A tutorial for querying with SQL syntax within R

Version 1.0.0

Konstantinos I. Bougioukas

21/10/2021

Contents

Objectives of the lessons

- Access a database from R using DBI.
- Run SQL queries within R using RSQLite.

Packages to download

We need to install the following packages to execute the full tutorial:

- `{tidyverse}`
- `{here}`
- `{DBI}`
- `{dbplyr}`
- `{RSQLite}`

1 Introduction to relational databases

So far, we have dealt with datasets that easily fit into our computer's memory. But what about datasets that are too large for our computer to handle as a whole? In this case, storing the data outside of R and organizing it in a database is helpful. Connecting to the database allows us to retrieve only the chunks needed for the current analysis.

A very common form of data storage is the relational database. There are many relational database management systems (RDBMS), such as `SQLite`, `MySQL`, `PostgreSQL`, `Oracle`, and many more. These different RDBMS each have their own advantages and limitations. Almost all employ `SQL` (structured query language) to pull data from the database.

Thankfully, several packages have been written that allows R to connect to relational databases and use the R programming language as the front end (what the user types in) to pull data from them. In this tutorial, we will give examples of how to read and analyzed data using R with SQLite RDBMS.

1.1 Data from the Portal Project

The Portal Project is a long-term ecological study being conducted near Portal, AZ. Since 1977, the site has been a primary focus of research on interactions among rodents, ants and plants and their respective responses to climate.

Data from the Portal project is recorded in a relational database designed for reliable storage and rapid access to the bounty of information produced by this long-term ecological experiment.

The research site consists of many **plots** – patches of the Arizona desert that are intensively manipulated and repeatedly surveyed. The plots have some fixed characteristics, such as the type of manipulation, geographic location, aspect, etc.

The plots have a lot of dynamic characteristics, and those changes are recorded in repeated **surveys**. In particular, the animals captured during each survey are identified to **species**, weighed, and measured.

There is also the `portalr` R package that provides a collection of basic functions to summarize the Portal project data on rodents, plants, ants, and weather at our long-term field site in the Chihuahuan Desert.

1.2 Basic definitions in SQL

Database: a container/repository (usually a file or set of files) to store organized data; a set of relational information.

Tables: the information inside the database is organized in tables; a structured list of data

Database terminology builds on common ways of characterizing data files. The breakdown of a table into **records** (also named rows) or **fields** (also named columns, or variables) is familiar to anyone who's worked in spreadsheets. The descriptions below formalize these terms, and provide an example referencing the Portal mammals database.

1.3 Types of SQL statements (DDL vs. DML)

SQL statements fall into two different categories: Data Definition Language statements and Data Manipulation Language statements.

Data Definition Language (or DDL) statements are used to define, change, or drop database objects such as tables.

➔ Common DDL statement types include `CREATE`, `ALTER`, `TRUNCATE`, and `DROP`:

`CREATE`: which is used for creating tables and defining its columns;

`ALTER`: is used for altering tables including adding and dropping columns and modifying their datatypes;

`TRUNCATE`: is used for deleting data in a table but not the table itself;

`DROP`: is used for deleting tables.

Data Manipulation Language (or DML) statements are used to read and modify data in tables. These are also sometimes referred to as CRUD operations, that is, Create, Read, Update and Delete rows in a table.

➔ Common DML statement types include `INSERT`, `SELECT`, `UPDATE`, and `DELETE`:

`INSERT`: is used for inserting a row or several rows of data into a table;

`SELECT`: reads or selects row or rows from a table;

`UPDATE`: edits row or rows in a table;

`DELETE`: removes a row or rows of data from a table.

In these notes we will primarly use the `SELECT` statement.

2 Connecting to a database

The first step from RStudio is creating a connection object that opens up a channel of communication to the database file. We will connect to our database using the `{DBI}` package. For the sake of example, we simply connect to an “in-memory” database, but a wide range of database connectors are available depending on where our data lives.

```
# connect to the database
mammals <- DBI::dbConnect(RSQLite::SQLite(),
                          here("data", "portal_mammals.sqlite"))
```

This command uses 2 packages that helps `dbplyr` and `dplyr` talk to the `SQLite` database. `{DBI}` is not something that we'll use directly as a user. It allows R to send commands to databases irrespective of the database management system used. The `{RSQLite}` allows R to interface with SQLite databases.

This command does not load the data into the R session (as the `read_csv()` function does). Instead, it merely instructs R to connect to the `SQLite` database contained in the `portal_mammals.sqlite` file.

Using a similar approach, we could connect to many other database management systems that are supported by R including `MySQL`, `PostgreSQL`, etc.

3 Explore the database

With the connection object availble, we can begin exploring the database.

First, we can list the tables at the connection:

```
# mammals SQLite database can contain multiple tables (datasets)
dbListTables(mammals)
```

```
## [1] "plots"   "species" "surveys"
```

or, alternately using the `src_dbi()` from {dbplyr}:

```
src_dbi(mammals)
```

```
## src:  sqlite 3.36.0 [D:\My_R\hsda\data\portal_mammals.sqlite]
## tbls: plots, species, surveys
```

Just like a spreadsheet with multiple worksheets, a SQLite database can contain multiple tables. In this case three of them are listed in the tbls row in the output above:

The three key tables in the relational database are:

- plots
- surveys
- species

Now, we're done with set-up. Let's explore the fields (variables) contained in a particular table:

```
# List the fields (variables) in a particular table
dbListFields(mammals, "plots")
```

```
## [1] "plot_id"   "plot_type"
```

```
dbListFields(mammals, "surveys")
```

```
## [1] "record_id"       "month"
## [3] "day"             "year"
## [5] "plot_id"         "species_id"
## [7] "sex"             "hindfoot_length"
## [9] "weight"
```

```
dbListFields(mammals, "species")
```

```
## [1] "species_id" "genus"      "species"
## [4] "taxa"
```

4 Simple database queries with SQL syntax

4.1 SELECT statement (select columns)

To connect to tables within a database, you can use the `tbl()` function from `{dplyr}`. This function can be used to send `SQL` queries to the database if it is combined with the `sql()` function. To demonstrate this functionality, let's select the columns `record_id`, `year`, `species_id`, and `plot_id` from the `surveys` table:

```
## SELECT statement
tbl(mammals, sql("SELECT record_id, year, species_id, plot_id
                  FROM surveys")) %>%
  head()
```

```
## # Source:   lazy query [?? x 4]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   record_id  year species_id plot_id
##       <int> <int> <chr>        <int>
## 1         1  1977 NL               2
## 2         2  1977 NL               3
## 3         3  1977 DM               2
## 4         4  1977 DM               7
## 5         5  1977 DM               3
## 6         6  1977 PF               1
```

Although it looks like we just got a data frame from the database, we didn't! It's a reference, showing us data that is still in the `SQLite` database (note the first two lines of the output). It does this because databases are often more efficient at selecting, filtering and joining large data sets than R. And typically, the database will not even be stored on our computer, but rather a more powerful machine somewhere on the web. So R is lazy and waits to bring this data into memory until we explicitly tell it to do so using the collect function from the `dbplyr` package.

Note that SQL is case insensitive, so capitalization only helps for readability and is a good style to adopt.

Alternatively, we can use `dbGetQuery()` function from {DBI} to pass SQL code to the database file:

```
dbGetQuery(mammals,
           "SELECT record_id, year, species_id, plot_id
           FROM surveys") %>%
  head()
```

```
##   record_id year species_id plot_id
## 1         1 1977         NL       2
## 2         2 1977         NL       3
## 3         3 1977         DM       2
## 4         4 1977         DM       7
## 5         5 1977         DM       3
## 6         6 1977         PF       1
```

Note that the same operation can be done using dplyr's verbs instead of writing SQL. First, we select the table on which to do the operations by creating the surveys object, and then we use the standard dplyr syntax as if it were a data frame:

```
surveys <- tbl(mammals, "surveys")
query1 <- surveys %>%
  select(year, species_id, plot_id) %>%
  head()
```

Additionally, we can use dplyr's `show_query()` function to show which SQL commands are actually sent to the database:

```
show_query(query1)
```

```
## <SQL>
## SELECT `year`, `species_id`, `plot_id`
## FROM `surveys`
## LIMIT 6
```

4.2 WHERE clause (filtering rows)

Relational operation helps us in restricting the result set by allowing us to use the clause `WHERE`. The `WHERE` clause always requires a predicate. A predicate is conditioned evaluates to true, false or unknown. Predicates are used in the search condition of the `WHERE` clause.

So, if we need to select the rows from the `surveys` table where the `species_id` is `DM` (Dipodomys), we use the `WHERE` clause with the predicate `species_id` equals `DM`. The result set is now restricted to rows whose condition evaluates to true.

```
## Filtering with WHERE statement and using basic operators
tbl(mammals, sql("SELECT record_id, year, species_id, plot_id
                 FROM surveys
                 WHERE species_id = 'DM'")) %>%
  head()
```

```
## # Source:   lazy query [?? x 4]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   record_id  year species_id plot_id
##       <int> <int> <chr>        <int>
## 1         3  1977 DM               2
## 2         4  1977 DM               7
## 3         5  1977 DM               3
## 4         8  1977 DM               1
## 5         9  1977 DM               1
## 6        12  1977 DM               7
```

Additionally, using `BETWEEN-AND` comparison operator we can filter a range of values:

```
# BETWEEN ... AND ....  operator
tbl(mammals, sql("SELECT record_id, year, species_id, plot_id
                 FROM surveys
```

```
                WHERE plot_id BETWEEN 3 AND 5")) %>%
  head()
```

```
## # Source:   lazy query [?? x 4]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   record_id  year species_id plot_id
##       <int> <int> <chr>        <int>
## 1         2  1977 NL               3
## 2         5  1977 DM               3
## 3        11  1977 DS               5
## 4        13  1977 DM               3
## 5        16  1977 DM               4
## 6        17  1977 DS               3
```

We can also find the missing values `NA` for a specific field:

```
# IS NULL operator
tbl(mammals, sql("SELECT record_id, year, species_id, plot_id
                FROM surveys
                WHERE species_id IS NULL")) %>%
  head()
```

```
## # Source:   lazy query [?? x 4]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   record_id  year species_id plot_id
##       <int> <int> <chr>        <int>
## 1       324  1977 <NA>             7
## 2       325  1977 <NA>            10
## 3       326  1977 <NA>            23
## 4       401  1977 <NA>             3
## 5       402  1977 <NA>            15
## 6       403  1977 <NA>            19
```

Other operators is `IN` that we can use for searching specific values:

```
# IN operator
tbl(mammals, sql("SELECT record_id, year, species_id, plot_id
                 FROM surveys
                 WHERE plot_id IN (1, 2, 7)")) %>%
  head()
```

```
## # Source:   lazy query [?? x 4]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   record_id  year species_id plot_id
##       <int> <int> <chr>        <int>
## 1         1  1977 NL               2
## 2         3  1977 DM               2
## 3         4  1977 DM               7
## 4         6  1977 PF               1
## 5         7  1977 PE               2
## 6         8  1977 DM               1
```

and `OR` operator

```
# OR operator
tbl(mammals, sql("SELECT record_id, year, species_id, plot_id
                 FROM surveys
                 WHERE species_id = 'NL' OR species_id = 'PF' ")) %>%
  head()
```

```
## # Source:   lazy query [?? x 4]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   record_id  year species_id plot_id
##       <int> <int> <chr>        <int>
## 1         1  1977 NL               2
```

```
## 2         2  1977 NL              3
## 3         6  1977 PF              1
## 4        10  1977 PF              6
## 5        19  1977 PF              4
## 6        22  1977 NL             15
```

We can also combine `OR` and `AND` operators:

```
# OR with AND (NOTE: use parenthesis, SQL processes AND before OR)
tbl(mammals, sql("SELECT record_id, year, species_id, plot_id
                  FROM surveys
                  WHERE (plot_id = 2 OR plot_id = 7) AND species_id = 'DM' ")) %>%
head()
```

```
## # Source:   lazy query [?? x 4]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   record_id  year species_id plot_id
##       <int> <int> <chr>        <int>
## 1         3  1977 DM               2
## 2         4  1977 DM               7
## 3        12  1977 DM               7
## 4        64  1977 DM               7
## 5        67  1977 DM               7
## 6        71  1977 DM               7
```

Another operator is `NOT` that excludes the records with a specific value:

```
# NOT operator (exclude records/rows)
tbl(mammals, sql("SELECT record_id, year, species_id, plot_id
                  FROM surveys
                  WHERE NOT species_id = 'PF' AND NOT species_id = 'DM' ")) %>%
  head()
```

```
## # Source:   lazy query [?? x 4]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   record_id  year species_id plot_id
##       <int> <int> <chr>        <int>
## 1         1  1977 NL               2
## 2         2  1977 NL               3
## 3         7  1977 PE               2
## 4        11  1977 DS               5
## 5        17  1977 DS               3
## 6        18  1977 PP               2
```

4.3 Additional useful expressions, `COUNT` and `LIMIT`

Next, we'll briefly present a few additional useful expressions that are used with select statements.

The first one is `COUNT` that is a built-in database function that retrieves the number of rows that match the query criteria.

```
## COUNT species_id = 'DM'
tbl(mammals, sql("SELECT COUNT (species_id)
                 FROM surveys
                 WHERE species_id = 'DM'"))
```

```
## # Source:   SQL [?? x 1]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   `COUNT (species_id)`
##                  <int>
## 1                10596
```

The second expression is `LIMIT` that is used for restricting the number of rows retrieved from the database. For example, we can retrieve just the first 10 rows from `surveys` table.

```
## LIMIT (just a few rows)
tbl(mammals, sql("SELECT *
                 FROM surveys
                 LIMIT 10 "))
```

```
## # Source:   SQL [?? x 9]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##    record_id month   day  year plot_id
##        <int> <int> <int> <int>   <int>
##  1         1     7    16  1977       2
##  2         2     7    16  1977       3
##  3         3     7    16  1977       2
##  4         4     7    16  1977       7
##  5         5     7    16  1977       3
##  6         6     7    16  1977       1
##  7         7     7    16  1977       2
##  8         8     7    16  1977       1
##  9         9     7    16  1977       1
## 10        10     7    16  1977       6
## # ... with 4 more variables:
## #   species_id <chr>, sex <chr>,
## #   hindfoot_length <int>, weight <int>
```

Note that if we use the asterisk in the `SELECT`, we request all columns of the table (instead of column names).

We can also retrieve just a few rows in the `surveys` table for a particular year such as 1985.

```
## WHERE and LIMIT (just a few rows for a particular year)
tbl(mammals, sql("SELECT *
                 FROM surveys
                 WHERE year = 1985 LIMIT 10 "))
```

```
## # Source:   SQL [?? x 9]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##    record_id month   day  year plot_id
##        <int> <int> <int> <int>   <int>
##  1      9790     1    19  1985      16
##  2      9791     1    19  1985      17
##  3      9792     1    19  1985       6
##  4      9793     1    19  1985      12
##  5      9794     1    19  1985      24
##  6      9795     1    19  1985      12
##  7      9796     1    19  1985       6
##  8      9797     1    19  1985      14
##  9      9798     1    19  1985       6
## 10      9799     1    19  1985      19
## # ... with 4 more variables:
## #   species_id <chr>, sex <chr>,
## #   hindfoot_length <int>, weight <int>
```

4.4 Sorting results sets with `ORDER BY` clause

```
## ORDER BY species_id
tbl(mammals, sql("SELECT record_id, year, species_id, plot_id
                 FROM surveys
                 ORDER BY species_id DESC")) %>%
  head()
```

```
## # Source:   lazy query [?? x 4]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   record_id  year species_id plot_id
##       <int> <int> <chr>        <int>
```

```
## 1     14250  1988 ZL              18
## 2     14351  1988 ZL              23
## 3     35512  2002 US              11
## 4     35513  2002 US              11
## 5     35528  2002 US              13
## 6     35544  2002 US              15
```

Of note, the `ORDER BY` clause must be always the last in a select statement.

4.5 Grouping Result Sets, `GROUP BY`

```
## Just COUNT sex
tbl(mammals, sql("SELECT COUNT (sex)
                 FROM surveys
                 "))
```

```
## # Source:   SQL [?? x 1]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   `COUNT (sex)`
##           <int>
## 1         33038
```

```
## COUNT BY GROUP
tbl(mammals, sql("SELECT sex, COUNT(sex)
                 AS count FROM surveys GROUP BY sex
                 "))
```

```
## # Source:   SQL [?? x 2]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   sex   count
```

```
##   <chr> <int>
## 1 <NA>      0
## 2 F     15690
## 3 M     17348
```

```
## COUNT BY GROUP and use HAVING clause for selection
tbl(mammals, sql("SELECT sex, COUNT(sex)
                 AS count FROM surveys GROUP BY sex
                 HAVING COUNT(sex) > 16000"))
```

```
## # Source:   SQL [?? x 2]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   sex   count
##   <chr> <int>
## 1 M     17348
```

4.6 Wildcards (using with string paterns) with `LIKE` predicate

```
tbl(mammals, sql("SELECT species_id, genus, taxa
                 FROM species"))
```

```
## # Source:   SQL [?? x 3]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##    species_id genus           taxa
##    <chr>      <chr>           <chr>
##  1 AB         Amphispiza      Bird
##  2 AH         Ammospermophilus Rodent
##  3 AS         Ammodramus      Bird
##  4 BA         Baiomys         Rodent
##  5 CB         Campylorhynchus Bird
```

```
##  6 CM         Calamospiza      Bird
##  7 CQ         Callipepla       Bird
##  8 CS         Crotalus         Reptile
##  9 CT         Cnemidophorus    Reptile
## 10 CU         Cnemidophorus    Reptile
## # ... with more rows
```

```
## grabs anything ending with letters 'za'
tbl(mammals, sql("SELECT species_id, genus, taxa
                 FROM species
                 WHERE genus LIKE '%za' "))
```

```
## # Source:   SQL [?? x 3]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   species_id genus       taxa
##   <chr>      <chr>       <chr>
## 1 AB         Amphispiza  Bird
## 2 CM         Calamospiza Bird
```

```
## grabs anything that starts with the letter 'A'
tbl(mammals, sql("SELECT species_id, genus, taxa
                 FROM species
                 WHERE genus LIKE 'A%' "))
```

```
## # Source:   SQL [?? x 3]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   species_id genus            taxa
##   <chr>      <chr>            <chr>
## 1 AB         Amphispiza       Bird
## 2 AH         Ammospermophilus Rodent
## 3 AS         Ammodramus       Bird
```

```
## grabs anything before and after the letters 'sp'
tbl(mammals, sql("SELECT species_id, genus, taxa
                 FROM species
                 WHERE genus LIKE '%sp%' "))
```

```
## # Source:   SQL [?? x 3]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   species_id genus            taxa
##   <chr>      <chr>            <chr>
## 1 AB         Amphispiza       Bird
## 2 AH         Ammospermophilus Rodent
## 3 CM         Calamospiza      Bird
## 4 SB         Spizella         Bird
## 5 SS         Spermophilus     Rodent
## 6 ST         Spermophilus     Rodent
## 7 US         Sparrow          Bird
```

```
## grabs anything that starts with 'c' and ends with 'us'
tbl(mammals, sql("SELECT species_id, genus, taxa
                 FROM species
                 WHERE genus LIKE 'c%us' "))
```

```
## # Source:   SQL [?? x 3]
## # Database: sqlite 3.36.0
## #   [D:\My_R\hsda\data\portal_mammals.sqlite]
##   species_id genus           taxa
##   <chr>      <chr>           <chr>
## 1 CB         Campylorhynchus Bird
## 2 CS         Crotalus        Reptile
## 3 CT         Cnemidophorus   Reptile
## 4 CU         Cnemidophorus   Reptile
## 5 CV         Crotalus        Reptile
```

```
## 6 PB          Chaetodipus     Rodent
## 7 PI          Chaetodipus     Rodent
## 8 PP          Chaetodipus     Rodent
## 9 PX          Chaetodipus     Rodent
```

5 Close the connection

Good housekeeping means always remembering to disconnect once you're done.

```
dbDisconnect(mammals) # closes our DB connection
```