Need to Connect R with SQL

It is common for Data Analysts/Scientists to connect R with SQL. For that reason, there exist many different packages designed for different Databases like PostgreSQL, MySQL etc. My suggestion is to work with the **DBI** package which is compatible with almost all the Databases.

Example of How to Connect R with SQL

As always we will start directly with a concrete example. Note that **dbConnect()** creates a connection between your R session and an SQL database. We will need to define the **DBIdriver object**, for example for MySQL databases, you can build such a driver with **RMySQL::MySQL()** for **RPostgreSQL::PostgreSQL()** and so on. If the database is a remote database hosted on a server, you'll also have to specify the following arguments in dbConnect(): dbname, host, port, user and password.

```
library(DBI)

# Connect to the MySQL database: con
con <- dbConnect(RMySQL::MySQL(),
                 dbname = "mydb",
                 host = "https://predictivehacks.com/",
                 port = 3306,
                 user = "george",
                 password = "predictivehacks")

# # Get table names
tables <- dbListTables(con)

# Display structure of tables
str(tables)
```

How to Import Tables

This how you import one table

```
# Import the accounts table from mydb
accounts <- dbReadTable(con, "accounts ")
```

This how you import all the tables

```
# # Get table names
tables <- dbListTables(con)

# Import all tables
tables <- lapply(tables , dbReadTable, conn = con)
```

How to Import Data from Queries

We can do it with `dbGetQuery` or by sending a query to the database with `dbSendQuery`

```
my_table <- dbGetQuery(con, "SELECT country, count(*) as tims FROM accounts
WHERE region = 'EU'")
my_table
```

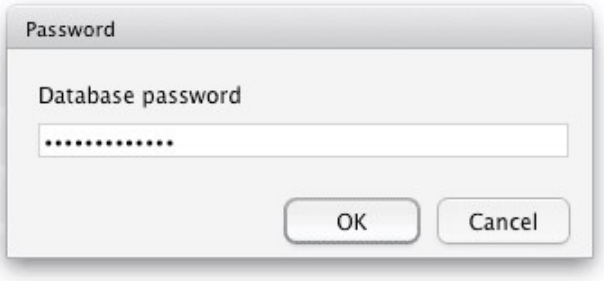
```
# Send query to the database
myquery <- dbSendQuery(con, "SELECT country, count(*) as tims FROM accounts
WHERE region = 'EU'")

# get the first two rows
dbFetch(myquery , n = 2)

# get all rows
dbFetch(myquery, n=-1 )


# Clear myquery
dbClearResult(myquery )
```

Remove the Credentials from your Code

You can find a detailed description of how you can avoid publishing credentials in your code.

Prompt for Credentials

The RStudio IDE's API can be used to prompt the user to enter the credentials in a popup box that masks what is typed:



https://db.rstudio.com/best-practices/managing-credentials/

So in our case would be:

```
library(DBI)

# Connect to the MySQL database: con
con <- dbConnect(RMySQL::MySQL(),
                 dbname = "mydb",
                 host = "https://predictivehacks.com/",
                 port = 3306,
                 user = rstudioapi::askForPassword("Database user"),
                 password = rstudioapi::askForPassword("Database password")
                )
```

Use Environment variables

Use Environment variables

The `.Renviron` file can be used to store the credentials, which can then be retrieved with `Sys.getenv()`. Here are the steps:

1. Create a new file defining the credentials:

```
userid = "username"
```

```
    pwd = "password"
```

2. Save it in your home directory with the file name `.Renviron`. If you are asked whether you want to save a file whose name begins with a dot, say **YES**.

3. Retrieve the credentials using `Sys.getenv()` while opening the connection:

```
library(DBI)

# Connect to the MySQL database: con
con <- dbConnect(RMySQL::MySQL(),
                 dbname = "mydb",
                 host = "https://predictivehacks.com/",
                 port = 3306,
                 user = Sys.getenv("userid"),
                 password = Sys.getenv("pwd")
                )
```

Work with a `config` file

In case we are dealing with many databases, and we have different credentials for each one, it is helpful to work with the `config` package that allows the connection code in R to reference an external file that defines values based on the environment. This process makes it easy to specify values to use for a connection locally and values to use after deployment.

For example:

```
library(DBI)
library(odbc)
library(config)

dw <- config::get("datawarehouse")

con <- dbConnect(
   Driver = dw$driver,
   Server = dw$server,
   UID    = dw$uid,
   PWD    = dw$pwd,
   Port   = dw$port,
   Database = dw$database
)
```

and the `config.yml` file:

```
default:
  datawarehouse:
    driver: 'Postgres'
    server: 'mydb-test.company.com'
    uid: 'local-account'
    pwd: 'my-password'  // not recommended, see alternatives below
    port: 5432
    database: 'regional-sales-sample'

rsconnect:
  datawarehouse:
    driver: 'PostgresPro'
    server: 'mydb-prod.company.com'
```

```
    uid: 'service-account'
    pwd: 'service-password' // not recommended, see alternatives below
    port: 5432
    database: 'regional-sales-full'
```

The `config` package determines the active configuration by looking at the `R_CONFIG_ACTIVE` environment variable. By default, RStudio Connect sets `R_CONFIG_ACTIVE` to the value `rsconnect`. In the config file above, the default datawarehouse values would be used locally and the datawarehouse values defined in the `rsconnect` section would be used on RStudio Connect. Administrators can optionally customize the name of the active configuration used in Connect .