R, Databases and Docker

Dipti Muni, Ian Frantz, John David Smith, Mary Anne Thygesen, M. Edward (Ed) Borasky, Scott Case, and Sophie Yang

2018-10-02

Contents

Chapter 1

Introduction

1.1 Using R to query a DBMS in your organization

- Large data stores in organizations are stored in databases that have specific access constraints and structural characteristics. Data documentation may be incomplete, often emphasizes operational issues rather than analytic ones, and often needs to be confirmed on the fly. Data volumes and query performance are important design constraints.
- R users frequently need to make sense of complex data structures and coding schemes to address incompletely formed questions so that exploratory data analysis has to be fast. Exploratory techniques for the purpose should not be reinvented (and so would benefit from more public instruction or discussion).
- Learning to navigate the interfaces (passwords, packages, etc.) between R and a database is difficult to simulate outside corporate walls. Resources for interface problem diagnosis behind corporate walls may or may not address all the issues that R users face, so a simulated environment is needed.

1.2 Docker as a tool for UseRs

Noam Ross's "Docker for the UseR" suggests that there are four distinct Docker use-cases for useRs.

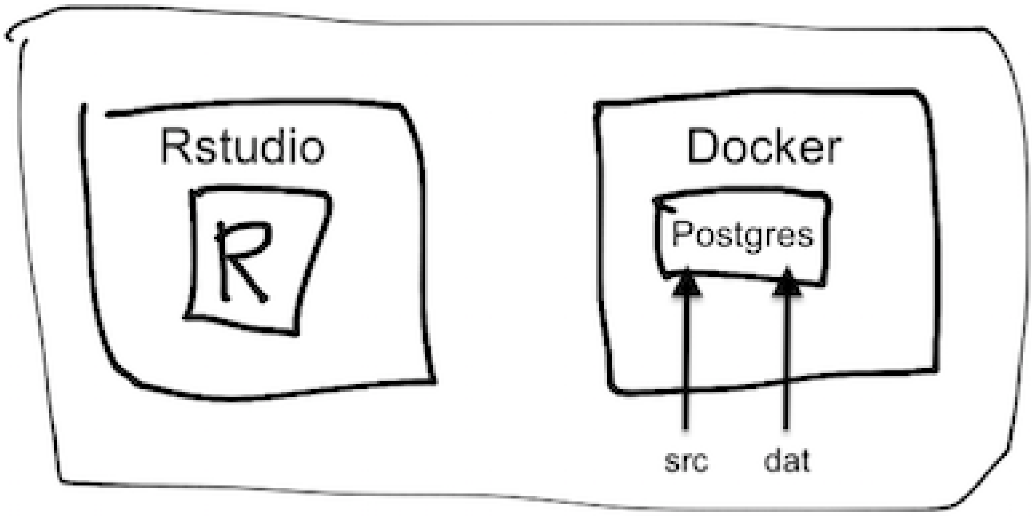
1. Make a fixed working environment for reproducible analysis
2. Access a service outside of R **(e.g., Postgres)**
3. Create an R based service (e.g., with `plumber`)
4. Send our compute jobs to the cloud with minimal reconfiguration or revision

This book explores #2 because it allows us to work on the database access issues described above and to practice on an industrial-scale DBMS.

- Docker is a relatively easy way to simulate the relationship between an R/RStudio session and a database – all on on a single machine, provided you have Docker installed and running.
- You may want to run PostgreSQL on a Docker container, avoiding any OS or system dependencies that might come up.

1.3 Docker and R on your machine

Here is how R and Docker fit on your operating system in this tutorial:



(This diagram needs to be updated as our directory structure evolves.)

1.4 Who are we?

- M. Edward (Ed) Borasky - @znmeb
- John David Smith - @smithjd
- Scott Came - @scottcame
- Ian Franz - @ianfrantz
- Sophie Yang - @SophieMYang
- Jim Tyhurst - @jimtyhurst

1.5 Prerequisites

You will need:

- A computer running Windows, MacOS, or Linux (Any Linux distro that will run Docker Community Edition, R and RStudio will work),
- R, and RStudio and
- Docker hosting.

The database we use is PostgreSQL 10, but you do not need to install that - it's installed via a Docker image. RStudio 1.2 is highly recommended but not required.

In addition to the current version of R and RStudio, you will need the following packages:

- tidyverse
- DBI
- RPostgres
- glue
- dbplyr

1.6 Install Docker

Install Docker. Installation depends on your operating system:

- On a Mac
- On UNIX flavors
- For Windows, consider these issues and follow these instructions.

1.7 Download the repo

First step: download this repo. It contains source code to build a Docker container that has the dvdrental database in PostgreSQL and shows how to interact with the database from R.

Chapter 2

Docker Hosting for Windows (02)

Skip these instructions if your computer has either OSX or a Unix variant.

2.1 Hardware requirements

You will need an Intel or AMD processor with 64-bit hardware and the hardware virtualization feature. Most machines you buy today will have that, but older ones may not. You will need to go into the BIOS / firmware and enable the virtualization feature. You will need at least 4 gigabytes of RAM!

2.2 Software requirements

You will need Windows 7 64-bit or later. If you can afford it, I highly recommend upgrading to Windows 10 Pro.

2.2.1 Windows 7, 8, 8.1 and Windows 10 Home (64 bit)

Install Docker Toolbox. The instructions are here: https://docs.docker.com/toolbox/toolbox_install_windows/. Make sure you try the test cases and they work!

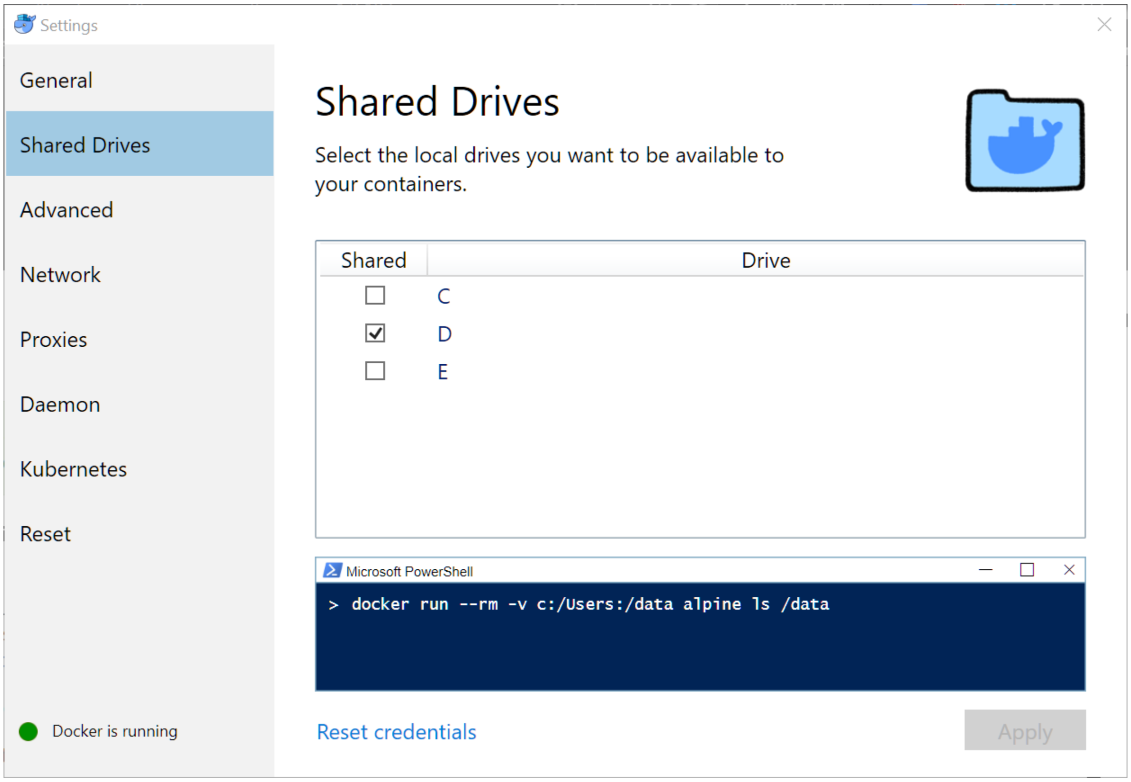
2.2.2 Windows 10 Pro

Install Docker for Windows *stable*. The instructions are here: https://docs.docker.com/docker-for-windows/install/#start-docker-for-windows. Again, make sure you try the test cases and they work.

2.3 Docker for Windows settings

2.3.1 Shared drives

If you're going to mount host files into container file systems (as we do in the following chapters), you need to set up shared drives. Open the Docker settings dialog and select `Shared Drives`. Check the drives you want to share. In this screenshot, the `D:` drive is my 1 terabyte hard drive.



Shared Drives

Select the local drives you want to be available to your containers.

Shared	Drive
☐	C
☑	D
☐	E

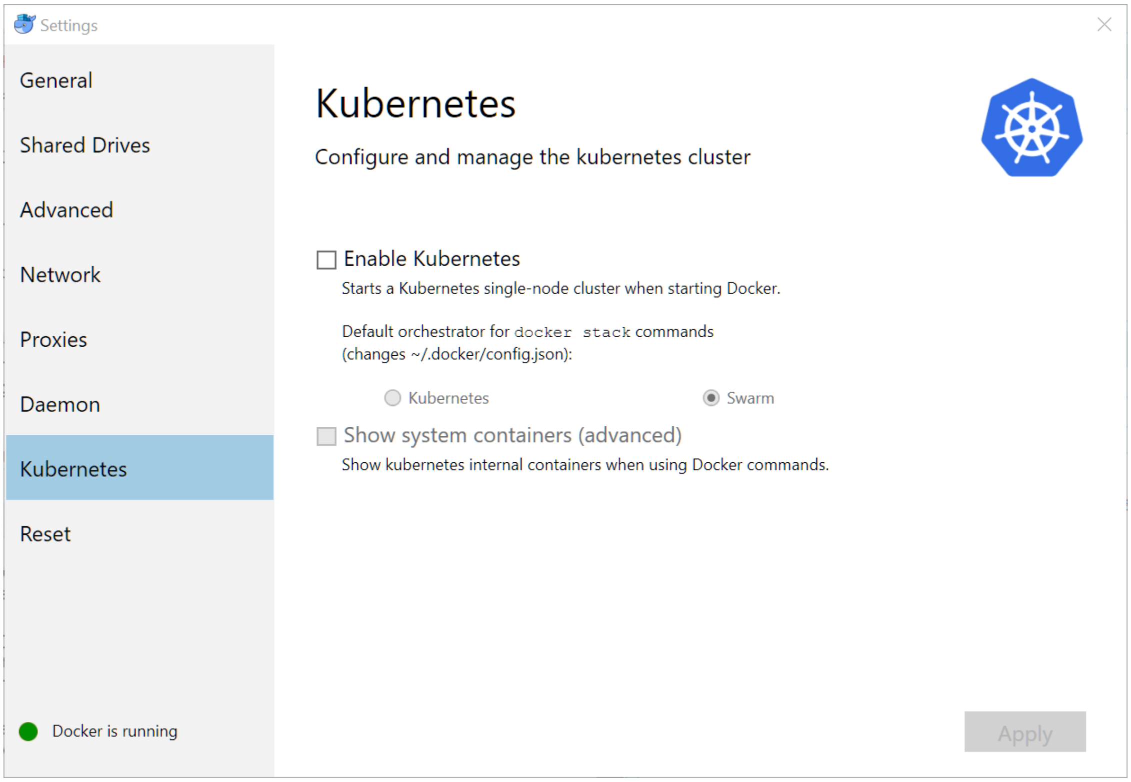
```
> docker run --rm -v c:/Users:/data alpine ls /data
```

2.3.2 Kubernetes

Kubernetes is a container orchestration / cloud management package that's a major DevOps tool. It's heavily supported by Red Hat and Google, and as a result is becoming a required skill for DevOps.

However, it's overkill for this project at the moment. So you should make sure it's not enabled.

Go to the `Kubernetes` dialog and make sure the `Enable Kubernetes` checkbox is cleared.



Kubernetes

Configure and manage the kubernetes cluster

□ Enable Kubernetes

Starts a Kubernetes single-node cluster when starting Docker.

Default orchestrator for `docker stack` commands (changes ~/.docker/config.json):

○ Kubernetes ● Swarm

□ Show system containers (advanced)

Show kubernetes internal containers when using Docker commands.

● Docker is running

2.4 Git, GitHub and line endings

Git was originally developed for Linux - in fact, it was created by Linus Torvalds to manage hundreds of different versions of the Linux kernel on different machines all around the world. As usage has grown, Git has achieved a huge following and is the version control system used by most large open source projects, including this one.

If you're on Windows, there are some things about Git and GitHub you need to watch. First of all, there are quite a few tools for running Git on Windows, but the RStudio default and recommended one is Git for Windows (https://git-scm.com/download/win).

By default, text files on Linux end with a single linefeed (`\n`) character. But on Windows, text files end with a carriage return and a line feed (`\r\n`). See https://en.wikipedia.org/wiki/Newline for the gory details.

Git defaults to checking files out in the native mode. So if you're on Linux, a text file will show up with the Linux convention, and if you're on Windows, it will show up with the Windows convention.

Most of the time this doesn't cause any problems. But Docker containers usually run Linux, and if you have files from a repository on Windows that you've sent to the container, the container may malfunction or give weird results. *This kind of situation has caused a lot of grief for contributors to this project, so beware.*

In particular, executable `sh` or `bash` scripts will fail in a Docker container if they have Windows line endings. You may see an error message with `\r` in it, which means the shell saw the carriage return (`\r`) and gave up. But often you'll see no hint at all what the problem was.

So you need a way to tell Git that some files need to be checked out with Linux line endings. See https://help.github.com/articles/dealing-with-line-endings/ for the details. Summary:

1. You'll need a `.gitattributes` file in the root of the repository.
2. In that file, all text files (scripts, program source, data, etc.) that are destined for a Docker container will need to have the designator `<spec> text eol=lf`, where `<spec>` is the file name specifier, for

example, `*.sh`.

This repo includes a sample: .gitattributes

Chapter 3

Learning Goals and Use Cases

3.1 Context: Why integrate R with databases using Docker? (03)

- Large data stores in organizations are stored in databases that have specific access constraints and structural characteristics.
- Learning to navigate the gap between R and the database is difficult to simulate outside corporate walls.
- R users frequently need to make sense of complex data structures using diagnostic techniques that should not be reinvented (and so would benefit from more public instruction and commentary).
- Docker is a relatively easy way to simulate the relationship between an R/Rstudio session and database – all on on a single machine.

3.2 Learning Goals

After working through this tutorial, you can expect to be able to:

- Run queries against PostgreSQL in an environment that simulates what you will find in a corporate setting.
- Understand some of the trade-offs between:
  1. queries aimed at exploration or informal investigation using dplyr; and
  2. those where performance is important because of the size of the database or the frequency with which a query is run.
- Rewrite dplyr queries as SQL and submit them directly.
- Gain some understanding of techniques for assessing query structure and performance.
- Set up a PostgreSQL database in a Docker environment.
- Understand enough about Docker to swap databases, e.g. Sports DB for the DVD rental database used in this tutorial. Or swap the database management system (DBMS), e.g. MySQL for PostgreSQL.

3.3 Use cases

Imagine that you have one of several roles at our fictional company **DVDs R Us** and that you need to:

- As a data scientist, I want to know the distribution of number of rentals per month per customer, so that the Marketing department can create incentives for customers in 3 segments: Frequent Renters, Average Renters, Infrequent Renters.
- As the Director of Sales, I want to see the total number of rentals per month for the past 6 months and I want to know how fast our customer base is growing/shrinking per month for the past 6 months.

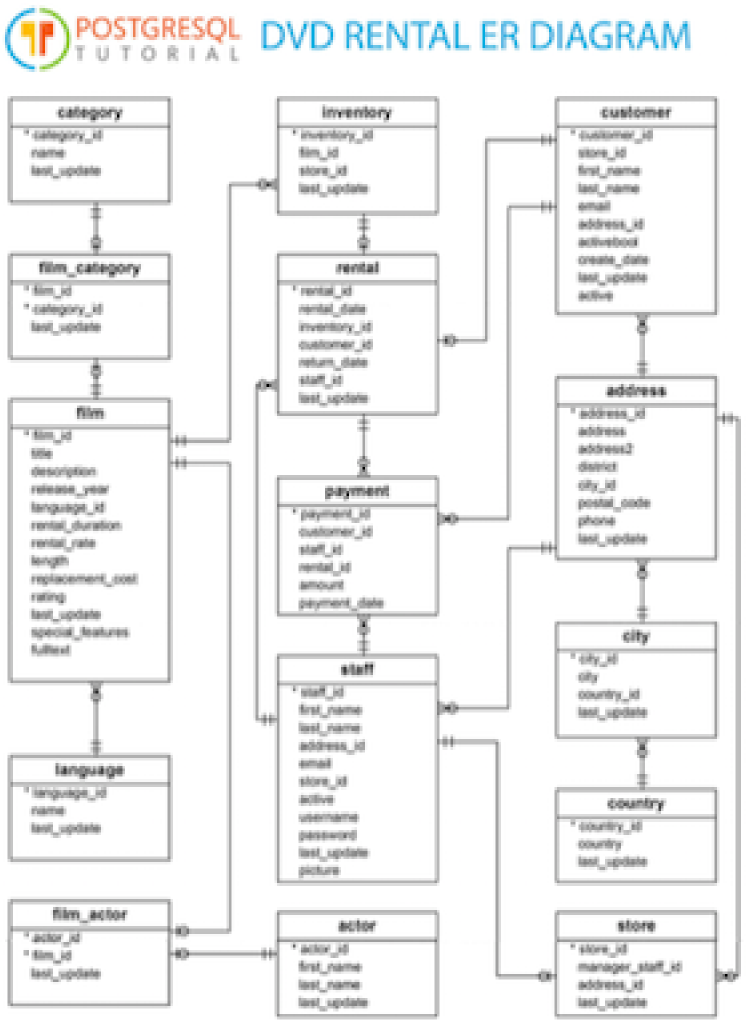


Figure 3.1: Entity Relationship diagram for the dvdrental database

- As the Director of Marketing, I want to know which categories of DVDs are the least popular, so that I can create a campaign to draw attention to rarely used inventory.
- As a shipping clerk, I want to add rental information when I fulfill a shipment order.
- As the Director of Analytics, I want to test as much of the production R code in my shop as possible against a new release of the DBMS that the IT department is implementing next month.
- etc.

3.4 Environment

This tutorial uses the Postgres version of "dvd rental" database, which can be downloaded here. Here's a glimpse of it's structure:

Chapter 4

Docker, Postgres, and R (04)

We always load the tidyverse and some other packages, but don't show it unless we are using packages other than `tidyverse`, `DBI`, `RPostgres`, and `glue`.

4.1 Verify that Docker is running

Docker commands can be run from a terminal (e.g., the Rstudio Terminal pane) or with a `system()` command. In this tutorial, we use `system2()` so that all the output that is created externally is shown. Note that `system2` calls are divided into several parts:

1. The program that you are sending a command to.
2. The parameters or commands that are being sent.
3. `stdout = TRUE, stderr = TRUE` are two parameters that are standard in this book, so that the command's full output is shown in the book.

The `docker version` command returns the details about the docker daemon that is running on your computer.

```
system2("docker", "version", stdout = TRUE, stderr = TRUE)
```

```
##  [1] "Client:"
##  [2] " Version:           18.06.1-ce"
##  [3] " API version:       1.38"
##  [4] " Go version:        go1.10.3"
##  [5] " Git commit:        e68fc7a"
##  [6] " Built:             Tue Aug 21 17:21:31 2018"
##  [7] " OS/Arch:           darwin/amd64"
##  [8] " Experimental:      false"
##  [9] ""
## [10] "Server:"
## [11] " Engine:"
## [12] "  Version:          18.06.1-ce"
## [13] "  API version:      1.38 (minimum version 1.12)"
## [14] "  Go version:       go1.10.3"
## [15] "  Git commit:       e68fc7a"
## [16] "  Built:            Tue Aug 21 17:29:02 2018"
## [17] "  OS/Arch:          linux/amd64"
## [18] "  Experimental:     true"
```

4.2 Clean up if appropriate

Remove the `cattle` and `sql-pet` containers if they exists (e.g., from a prior experiments).

```
if (system2("docker", "ps -a", stdout = TRUE) %>%
   grepl(x = ., pattern = 'cattle') %>%
   any()) {
     system2("docker", "rm -f cattle")
}
if (system2("docker", "ps -a", stdout = TRUE) %>%
   grepl(x = ., pattern = 'sql-pet') %>%
   any()) {
     system2("docker", "rm -f sql-pet")
}
```

The convention we use in this book is to assemble a command with `glue` so that the you can see all of its separate parts. The following chunk just constructs the command, but does not execute it. If you have problems executing a command, you can always copy the command and execute in your terminal session.

```
docker_cmd <- glue(
  "run ",      # Run is the Docker command.  Everything that follows are `docker run` parameters.
  "--detach ", # (or `-d`) tells Docker to disconnect from the terminal / program issuing the command
  "--name cattle ",       # tells Docker to give the container a name: `cattle`
  "--publish 5432:5432 ", # tells Docker to expose the Postgres port 5432 to the local network with 543
  " postgres:10 "  # tells Docker the image that is to be run (after downloading if necessary)
)

# We name containers `cattle` for "throw-aways" and `pet` for ones we treasure and keep around.  :-)
```

Submit the command constructed above:

```
# this is what you would submit from a terminal:
cat(glue(" docker ", docker_cmd))
```

```
##  docker run --detach --name cattle --publish 5432:5432  postgres:10
```

```
# this is how R submits it to Docker:
system2("docker", docker_cmd, stdout = TRUE, stderr = TRUE)
```

```
## [1] "12b9285bd4144529d98b79a9d723e4dd944e369956bd6ec81fc7f42f20fcb0c9"
```

Docker returns a long string of numbers. If you are running this command for the first time, Docker downloads the PostgreSQL image, which takes a bit of time.

The following command shows that a container named `cattle` is running `postgres:10`. `postgres` is waiting for a connection:

```
system2("docker", "ps", stdout = TRUE, stderr = TRUE)
```

```
## [1] "CONTAINER ID      IMAGE          COMMAND                CREATED               STATUS              PORTS
## [2] "12b9285bd414      postgres:10      \"docker-entrypoint.s…\"  Less than a second ago  Up Less than a
```

4.3 Connect, read and write to Postgres from R

4.3.1 Pause for some security considerations

We use the following `wait_for_postgres` function, which will repeatedly try to connect to PostgreSQL. PostgreSQL can take different amounts of time to come up and be ready to accept connections from R, depending on various factors that will be discussed later on.

```
#' Connect to Postgres, waiting if it is not ready
#'
#' @param user Username that will be found
#' @param password Password that corresponds to the username
#' @param dbname the name of the database in the database
#' @param seconds_to_test the number of iterations to try while waiting for Postgres to be ready
#' @export
wait_for_postgres <- function(user, password, dbname, seconds_to_test = 10) {
  for (i in 1:seconds_to_test) {
    db_ready <- DBI::dbCanConnect(RPostgres::Postgres(),
                                 host = "localhost",
                                 port = "5432",
                                 user = user,
                                 password = password,
                                 dbname = dbname)
    if ( !db_ready ) {Sys.sleep(1)}
    else {con <- DBI::dbConnect(RPostgres::Postgres(),
                               host = "localhost",
                               port = "5432",
                               user = user,
                               password = password,
                               dbname = dbname)
    }
    if (i == seconds_to_test & !db_ready) {con <- "There is no connection"}
  }
  con
}
```

When we call `wait_for_postgres` we'll use environment variables that R obtains from reading a file named `.Rprofile`. That file is not uploaded to Github and R looks for it in your default directory. To see whether you have already created that file, execute:

```
dir(path = "~", pattern = ".Rprofile", all.files = TRUE)
```

```
## [1] ".Rprofile"
```

It should contain lines such as:

```
  DEFAULT_POSTGRES_PASSWORD=postgres
  DEFAULT_POSTGRES_USER_NAME=postgres
```

Those are the default values for the username and password, but this approach demonstrates how they would be kept secret and not uploaded to Github or some other public location.

This is how the `wait_for_postgres` function is used:

```
con <- wait_for_postgres(user = Sys.getenv("DEFAULT_POSTGRES_USER_NAME"),
                         password = Sys.getenv("DEFAULT_POSTGRES_PASSWORD"),
                         dbname = "postgres",
                         seconds_to_test = 10)
```

Make sure that you can connect to the PostgreSQL database that you started earlier. If you have been executing the code from this tutorial, the database will not contain any tables yet:

```
dbListTables(con)
```

```
## character(0)
```

4.3.2 Alternative: put the database password in an environment file

The goal is to put the password in an untracked file that will **not** be committed in your source code repository. Your code can reference the name of the variable, but the value of that variable will not appear in open text in your source code.

We have chosen to call the file `dev_environment.csv` in the current working directory where you are executing this script. That file name appears in the `.gitignore` file, so that you will not accidentally commit it. We are going to create that file now.

You will be prompted for the database password. By default, a PostgreSQL database defines a database user named `postgres`, whose password is `postgres`. If you have changed the password or created a new user with a different password, then enter those new values when prompted. Otherwise, enter `postgres` and `postgres` at the two prompts.

In an interactive environment, you could execute a snippet of code that prompts the user for their username and password with the following snippet (which isn't run in the book):

```r
prompt_for_postgres <- function(seconds_to_test){
  for (i in 1:seconds_to_test) {
    db_ready <- DBI::dbCanConnect(RPostgres::Postgres(),
                                  host = "localhost",
                                  port = "5432",
                                  user = dplyr::filter(environment_variables, variable == "username")[,
                                  password = dplyr::filter(environment_variables, variable == "password
                                  dbname = "postgres")
    if ( !db_ready ) {Sys.sleep(1)}
    else {con <- DBI::dbConnect(RPostgres::Postgres(),
                                host = "localhost",
                                port = "5432",
                                user = dplyr::filter(environment_variables, variable == "username")[, "
                                password = dplyr::filter(environment_variables, variable == "password")
                                dbname = "postgres")
    }
    if (i == seconds_to_test & !db_ready) {con <- "there is no connection "}
  }
  con
}

DB_USERNAME <- trimws(readline(prompt = "username: "), which = "both")
DB_PASSWORD <- getPass::getPass(msg = "password: ")
environment_variables = data.frame(
  variable = c("username", "password"),
  value = c(DB_USERNAME, DB_PASSWORD),
  stringsAsFactors = FALSE)
write.csv(environment_variables, "./dev_environment.csv", row.names = FALSE)
```

Your password is still in plain text in the file, `dev_environment.csv`, so you should protect that file from exposure. However, you do not need to worry about committing that file accidentally to your git repository, because the name of the file appears in the `.gitignore` file.

For security, we use values from the `environment_variables` data.frame, rather than keeping the `username` and `password` in plain text in a source file.

4.3.3 Interact with Postgres

Write mtcars to PostgreSQL

```
dbWriteTable(con, "mtcars", mtcars, overwrite = TRUE)
```

List the tables in the PostgreSQL database to show that mtcars is now there:

```
dbListTables(con)
```

```
## [1] "mtcars"
```

```
# list the fields in mtcars:
dbListFields(con, "mtcars")
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

Download the table from the DBMS to a local data frame:

```
mtcars_df <- tbl(con, "mtcars")

# Show a few rows:
knitr::kable(head(mtcars_df))
```

mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
21.0	6	160	110	3.90	2.620	16.46	0	1	4	4
21.0	6	160	110	3.90	2.875	17.02	0	1	4	4
22.8	4	108	93	3.85	2.320	18.61	1	1	4	1
21.4	6	258	110	3.08	3.215	19.44	1	0	3	1
18.7	8	360	175	3.15	3.440	17.02	0	0	3	2
18.1	6	225	105	2.76	3.460	20.22	1	0	3	1

4.4 Clean up

Afterwards, always disconnect from the DBMS, stop the docker container and (optionally) remove it.

```
dbDisconnect(con)

# tell Docker to stop the container:
system2("docker", "stop cattle", stdout = TRUE, stderr = TRUE)
```

```
## [1] "cattle"
```

```
# Tell Docker to remove the container from it's library of active containers:
system2("docker", "rm cattle", stdout = TRUE, stderr = TRUE)
```

```
## [1] "cattle"
```

If we stop the docker container but don't remove it (with the rm cattle command), the container will persist and we can start it up again later with start cattle. In that case, mtcars would still be there and we could retrieve it from R again. Since we have now removed the cattle container, the whole database has been deleted. (There are enough copies of mtcars in the world, so no great loss.)

Chapter 5

A persistent database in Postgres in Docker - all at once (05)

5.1 Overview

You've already connected to PostgreSQL with R, now you need a "realistic" (`dvdrental`) database. We're going to demonstrate how to set one up, with two different approaches. This chapter and the next do the same job, illustrating the different approaches that you can take and helping you see the different points where you could swap what's provided here with a different DBMS or a different backup file or something else.

The code in this first version is recommended because it is an "all in one" approach. Details about how it works and how you might modify it are included below. There is another version in the the next chapter that you can use to investigate Docker commands and components.

Note that this approach relies on two files that have quote that's not shown here: dvdrental.Dockerfile and init-dvdrental.sh. They are discussed below.

Note that `tidyverse`, `DBI`, `RPostgres`, and `glue` are loaded.

5.2 Verify that Docker is up and running

```r
system2("docker", "version", stdout = TRUE, stderr = TRUE)
```

```
##  [1] "Client:"
##  [2] " Version:           18.06.1-ce"
##  [3] " API version:       1.38"
##  [4] " Go version:        go1.10.3"
##  [5] " Git commit:        e68fc7a"
##  [6] " Built:             Tue Aug 21 17:21:31 2018"
##  [7] " OS/Arch:           darwin/amd64"
##  [8] " Experimental:      false"
##  [9] ""
## [10] "Server:"
## [11] " Engine:"
## [12] "  Version:          18.06.1-ce"
## [13] "  API version:      1.38 (minimum version 1.12)"
## [14] "  Go version:       go1.10.3"
## [15] "  Git commit:       e68fc7a"
## [16] "  Built:            Tue Aug 21 17:29:02 2018"
## [17] "  OS/Arch:          linux/amd64"
## [18] "  Experimental:     true"
```

5.3 Clean up if appropriate

Remove the `sql-pet` container if it exists (e.g., from a prior run)

```
if (system2("docker", "ps -a", stdout = TRUE) %>%
   grepl(x = ., pattern = 'sql-pet') %>%
   any()) {
     system2("docker", "rm -f sql-pet")
}
```

5.4 Build the Docker Image

Build an image that derives from postgres:10, defined in `dvdrental.Dockerfile`, that is set up to restore and load the dvdrental db on startup. The dvdrental.Dockerfile is discussed below.

```
system2("docker",
        glue("build ", # tells Docker to build an image that can be loaded as a container
          "--tag postgres-dvdrental ", # (or -t) tells Docker to name the image
          "--file dvdrental.Dockerfile ", #(or -f) tells Docker to read `build` instructions from the d
          " . "),  # tells Docker to look for dvdrental.Dockerfile in the current directory
          stdout = TRUE, stderr = TRUE)
```

```
##  [1] "Sending build context to Docker daemon  3.025MB\r\r"
##  [2] "Step 1/4 : FROM postgres:10"
##  [3] " ---> ac25c2bac3c4"
##  [4] "Step 2/4 : WORKDIR /tmp"
##  [5] " ---> Using cache"
##  [6] " ---> 3f00a18e0bdf"
##  [7] "Step 3/4 : COPY init-dvdrental.sh /docker-entrypoint-initdb.d/"
##  [8] " ---> Using cache"
##  [9] " ---> 3453d61d8e3e"
## [10] "Step 4/4 : RUN apt-get -qq update &&  apt-get install -y -qq curl zip  > /dev/null 2>&1 &&  curl -Os
## [11] " ---> Using cache"
## [12] " ---> f5e93aa64875"
## [13] "Successfully built f5e93aa64875"
## [14] "Successfully tagged postgres-dvdrental:latest"
```

5.5 Run the Docker Image

Run docker to bring up postgres. The first time it runs it will take a minute to create the PostgreSQL environment. There are two important parts to this that may not be obvious:

- The `source=` parameter points to dvdrental.Dockerfile, which does most of the heavy lifting. It has detailed, line-by-line comments to explain what it is doing.

- *Inside* dvdrental.Dockerfile the command `COPY init-dvdrental.sh /docker-entrypoint-initdb.d/` copies init-dvdrental.sh from the local file system into the specified location in the Docker container. When the PostgreSQL Docker container initializes, it looks for that file and executes it.

Doing all of that work behind the scenes involves two layers of complexity. Depending on how you look at it, that may be more or less difficult to understand than the method shown in the next Chapter.

```
wd <- getwd()

docker_cmd <- glue(
  "run ",      # Run is the Docker command.  Everything that follows are `run` parameters.
  "--detach ", # (or `-d`) tells Docker to disconnect from the terminal / program issuing the command
  " --name sql-pet ",     # tells Docker to give the container a name: `sql-pet`
  "--publish 5432:5432 ", # tells Docker to expose the Postgres port 5432 to the local network with 543
  "--mount ", # tells Docker to mount a volume -- mapping Docker's internal file structure to the host
  "type=bind,", # tells Docker that the mount command points to an actual file on the host system
  'source="', # tells Docker where the local file will be found
  wd, '/",', # the current working directory, as retrieved above
  "target=/petdir", # tells Docker to refer to the current directory as "/petdir" in its file system
  " postgres-dvdrental" # tells Docker to run the image was built in the previous step
)

# if you are curious you can paste this string into a terminal window after the command 'docker':
docker_cmd
```

```
## run --detach  --name sql-pet --publish 5432:5432 --mount type=bind,source="/Users/jds/Documents/Library
```

```
system2("docker", docker_cmd, stdout = TRUE, stderr = TRUE)
```

```
## [1] "6326702dd14eaa6a5e1dd9755645f891075675bb1c93c203027d0c515eb2148e"
```

5.6 Connect to Postgres with R

Use the DBI package to connect to PostgreSQL. But first, wait for Docker & PostgreSQL to come up before connecting.

We have loaded the `wait_for_postgres` function behind the scenes.

```
con <- wait_for_postgres(user = Sys.getenv("DEFAULT_POSTGRES_USER_NAME"),
                         password = Sys.getenv("DEFAULT_POSTGRES_PASSWORD"),
                         dbname = "dvdrental",
                         seconds_to_test = 10)


# if (con == "it's not there") {stop()}

dbListTables(con)
```

```
##  [1] "actor_info"                 "customer_list"
##  [3] "film_list"                  "nicer_but_slower_film_list"
##  [5] "sales_by_film_category"     "staff"
##  [7] "sales_by_store"             "staff_list"
##  [9] "category"                   "film_category"
## [11] "country"                    "actor"
## [13] "language"                   "inventory"
## [15] "payment"                    "rental"
## [17] "city"                         "store"
## [19] "film"                         "address"
## [21] "film_actor"                   "customer"
```

```
dbListFields(con, "rental")
```

```
## [1] "rental_id"    "rental_date"  "inventory_id" "customer_id"
## [5] "return_date"  "staff_id"     "last_update"
```

```
dbDisconnect(con)

Sys.sleep(2) # Can take a moment to disconnect.
```

5.7 Stop and start to demonstrate persistence

Stop the container

```
system2('docker', 'stop sql-pet',
        stdout = TRUE, stderr = TRUE)
```

```
## [1] "sql-pet"
```

Restart the container and verify that the dvdrental tables are still there

```
system2("docker",  "start sql-pet", stdout = TRUE, stderr = TRUE)
```

```
## [1] "sql-pet"
```

```
con <- wait_for_postgres(user = Sys.getenv("DEFAULT_POSTGRES_USER_NAME"),
                         password = Sys.getenv("DEFAULT_POSTGRES_PASSWORD"),
                         dbname = "dvdrental",
                         seconds_to_test = 10)

glimpse(dbReadTable(con, "film"))
```

```
## Observations: 1,000
## Variables: 13
## $ film_id          <int> 133, 384, 8, 98, 1, 2, 3, 4, 5, 6, 7, 9, 10, ...
## $ title            <chr> "Chamber Italian", "Grosse Wonderful", "Airpo...
## $ description      <chr> "A Fateful Reflection of a Moose And a Husban...
## $ release_year     <int> 2006, 2006, 2006, 2006, 2006, 2006, 2006, 200...
## $ language_id      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ rental_duration  <int> 7, 5, 6, 4, 6, 3, 7, 5, 6, 3, 6, 3, 6, 6, 6, ...
## $ rental_rate      <dbl> 4.99, 4.99, 4.99, 4.99, 0.99, 4.99, 2.99, 2.9...
## $ length           <int> 117, 49, 54, 73, 86, 48, 50, 117, 130, 169, 6...
## $ replacement_cost <dbl> 14.99, 19.99, 15.99, 12.99, 20.99, 12.99, 18....
## $ rating           <chr> "NC-17", "R", "R", "PG-13", "PG", "G", "NC-17...
## $ last_update      <dttm> 2013-05-26 14:50:58, 2013-05-26 14:50:58, 20...
## $ special_features <chr> "{Trailers}", "{\"Behind the Scenes\"}", "{Tr...
## $ fulltext         <chr> "'chamber':1 'fate':4 'husband':11 'italian':...
```

5.8 Cleaning up

It's always good to have R disconnect from the database

```
dbDisconnect(con)
```

Stop the container and show that the container is still there, so can be started again.

```
system2('docker', 'stop sql-pet',
        stdout = TRUE, stderr = TRUE)
```

```
## [1] "sql-pet"
```

```
# show that the container still exists even though it's not running
psout <- system2("docker", "ps -a", stdout = TRUE)
psout[grepl(x = psout, pattern = 'sql-pet')]
```

```
## [1] "6326702dd14e      postgres-dvdrental  \"docker-entrypoint.s…\"  20 seconds ago     Exited (137) Les
```

Next time, you can just use this command to start the container:

```
system2("docker",  "start sql-pet", stdout = TRUE, stderr = TRUE)
```

And once stopped, the container can be removed with:

```
system2("docker",  "rm sql-pet", stdout = TRUE, stderr = TRUE)
```

5.9 Using the sql-pet container in the rest of the book

After this point in the book, we assume that Docker is up and that we can always start up our *sql-pet database* with:

```
system2("docker",  "start sql-pet", stdout = TRUE, stderr = TRUE)
```

Chapter 6

A persistent database in Postgres in Docker - piecemeal (06)

6.1 Overview

This chapter essentially repeats what was presented in the previous one, but does it in a step-by-step way that might be useful to understand how each of the steps involved in setting up a persistent PostgreSQL database works. If you are satisfied with the method shown in that chapter, skip this one for now.

Note that `tidyverse`, `DBI`, `RPostgres`, and `glue` are loaded.

6.2 Download the `dvdrental` backup file

The first step is to get a local copy of the `dvdrental` PostgreSQL restore file. It comes in a zip format and needs to be un-zipped. Use the `downloader` and `here` packages to keep track of things.

```
if (!require(downloader)) install.packages("downloader")
```

```
## Loading required package: downloader
```

```
if (!require(here)) install.packages("here")
```

```
## Loading required package: here

## here() starts at /Users/jds/Documents/Library/R/r-system/sql-pet/r-database-docker
```

```
library(downloader, here)

download("http://www.postgresqltutorial.com/wp-content/uploads/2017/10/dvdrental.zip", destfile = here(

unzip(here("dvdrental.zip"), exdir = here()) # creates a tar archhive named "dvdrental.tar"

file.remove(here("dvdrental.zip")) # the Zip file is no longer needed.
```

```
## [1] TRUE
```

6.3 Verify that Docker is up and running:

```
system2("docker", "version", stdout = TRUE, stderr = TRUE)
```

```
##  [1] "Client:"
##  [2] " Version:           18.06.1-ce"
##  [3] " API version:       1.38"
##  [4] " Go version:        go1.10.3"
##  [5] " Git commit:        e68fc7a"
##  [6] " Built:             Tue Aug 21 17:21:31 2018"
##  [7] " OS/Arch:           darwin/amd64"
##  [8] " Experimental:      false"
##  [9] ""
## [10] "Server:"
## [11] " Engine:"
## [12] "  Version:          18.06.1-ce"
## [13] "  API version:      1.38 (minimum version 1.12)"
## [14] "  Go version:       go1.10.3"
## [15] "  Git commit:       e68fc7a"
## [16] "  Built:            Tue Aug 21 17:29:02 2018"
## [17] "  OS/Arch:          linux/amd64"
## [18] "  Experimental:     true"
```

6.4 Clean up if appropriate

Remove the `sql-pet` container if it exists (e.g., from a prior run)

```
if (system2("docker", "ps -a", stdout = TRUE) %>%
   grepl(x = ., pattern = 'sql-pet') %>%
   any()) {
     system2("docker", "rm -f sql-pet")
}
```

6.5 Build the Docker Image

Build an image that derives from postgres:10. Connect the local and Docker directories that need to be shared. Expose the standard PostgreSQL port 5432.

” postgres-dvdrental” # tells Docker the image that is to be run (after downloading if necessary)

```
wd <- getwd()

docker_cmd <- glue(
  "run ",      # Run is the Docker command.  Everything that follows are `run` parameters.
  "--detach ", # (or `-d`) tells Docker to disconnect from the terminal / program issuing the command
  " --name sql-pet ",     # tells Docker to give the container a name: `sql-pet`
  "--publish 5432:5432 ", # tells Docker to expose the Postgres port 5432 to the local network with 543
  "--mount ", # tells Docker to mount a volume -- mapping Docker's internal file structure to the host
  'type=bind,source="', wd, '/",target=/petdir',
  " postgres:10 " # tells Docker the image that is to be run (after downloading if necessary)
)
cat('docker ',docker_cmd)
```

```
## docker  run --detach  --name sql-pet --publish 5432:5432 --mount type=bind,source="/Users/jds/Documents/
```

```
system2("docker", docker_cmd, stdout = TRUE, stderr = TRUE)
```

```
## [1] "569798b50767d3310e7369b2690248165b6ce5757df23f83368e43383cc1cd0e"
```

Peek inside the docker container and list the files in the `petdir` directory. Notice that `dvdrental.tar` is in both.

```
system2('docker', 'exec sql-pet ls petdir | grep "dvdrental.tar" ',
        stdout = TRUE, stderr = TRUE)
```

```
## [1] "dvdrental.tar"
```

```
dir(wd, pattern = "dvdrental.tar")
```

```
## [1] "dvdrental.tar"
```

6.6 Create the database and restore from the backup

We can execute programs inside the Docker container with the `exec` command. In this case we tell Docker to execute the `psql` program inside the `sql-pet` container and pass it some commands.

```
Sys.sleep(2)  # is this really needed?
# inside Docker, execute the postgress SQL command-line program to create the dvdrental database:
system2('docker', 'exec sql-pet psql -U postgres -c "CREATE DATABASE dvdrental;"',
        stdout = TRUE, stderr = TRUE)
```

```
## [1] "CREATE DATABASE"
```

The `psql` program repeats back to us what it has done, e.g., to create a database named `dvdrental`.

Next we execute a different program in the Docker container, `pg_restore`, and tell it where the restore file is located. If successful, the `pg_restore` just responds with a very laconic `character(0)`.

```
# restore the database from the .tar file
system2("docker", "exec sql-pet pg_restore -U postgres -d dvdrental petdir/dvdrental.tar", stdout = TRU
```

```
## character(0)
```

```
file.remove(here("dvdrental.tar")) # the tar file is no longer needed.
```

```
## [1] TRUE
```

6.7 Connect to the database with R

Use the DBI package to connect to PostgreSQL. But first, wait for Docker & PostgreSQL to come up before connecting.

We have loaded the `wait_for_postgres` function behind the scenes.

```
con <- wait_for_postgres(user = Sys.getenv("DEFAULT_POSTGRES_USER_NAME"),
                         password = Sys.getenv("DEFAULT_POSTGRES_PASSWORD"),
                         dbname = "dvdrental",
                         seconds_to_test = 10)

dbListTables(con)
```

```
##  [1] "actor_info"                 "customer_list"
##  [3] "film_list"                  "nicer_but_slower_film_list"
##  [5] "sales_by_film_category"     "staff"
##  [7] "sales_by_store"             "staff_list"
##  [9] "category"                   "film_category"
## [11] "country"                    "actor"
## [13] "language"                   "inventory"
## [15] "payment"                    "rental"
## [17] "city"                       "store"
## [19] "film"                       "address"
## [21] "film_actor"                 "customer"
```

```
dbListFields(con, "film")
```

```
##  [1] "film_id"          "title"            "description"
##  [4] "release_year"     "language_id"      "rental_duration"
##  [7] "rental_rate"      "length"           "replacement_cost"
## [10] "rating"           "last_update"      "special_features"
## [13] "fulltext"
```

```
dbDisconnect(con)
```

6.8 Stop and start to demonstrate persistence

Stop the container

```
system2('docker', 'stop sql-pet',
        stdout = TRUE, stderr = TRUE)
```

```
## [1] "sql-pet"
```

Restart the container and verify that the dvdrental tables are still there

```
system2("docker",  "start sql-pet", stdout = TRUE, stderr = TRUE)
```

```
## [1] "sql-pet"
```

```
con <- wait_for_postgres(user = Sys.getenv("DEFAULT_POSTGRES_USER_NAME"),
                         password = Sys.getenv("DEFAULT_POSTGRES_PASSWORD"),
                         dbname = "dvdrental",
                         seconds_to_test = 10)

glimpse(dbReadTable(con, "film"))
```

```
## Observations: 1,000
## Variables: 13
## $ film_id          <int> 133, 384, 8, 98, 1, 2, 3, 4, 5, 6, 7, 9, 10, ...
## $ title            <chr> "Chamber Italian", "Grosse Wonderful", "Airpo...
## $ description      <chr> "A Fateful Reflection of a Moose And a Husban...
## $ release_year     <int> 2006, 2006, 2006, 2006, 2006, 2006, 2006, 200...
## $ language_id      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...
## $ rental_duration  <int> 7, 5, 6, 4, 6, 3, 7, 5, 6, 3, 6, 3, 6, 6, 6, ...
## $ rental_rate      <dbl> 4.99, 4.99, 4.99, 4.99, 0.99, 4.99, 2.99, 2.9...
## $ length           <int> 117, 49, 54, 73, 86, 48, 50, 117, 130, 169, 6...
## $ replacement_cost <dbl> 14.99, 19.99, 15.99, 12.99, 20.99, 12.99, 18....
## $ rating           <chr> "NC-17", "R", "R", "PG-13", "PG", "G", "NC-17...
## $ last_update      <dttm> 2013-05-26 14:50:58, 2013-05-26 14:50:58, 20...
## $ special_features <chr> "{Trailers}", "{\"Behind the Scenes\"}", "{Tr...
## $ fulltext         <chr> "'chamber':1 'fate':4 'husband':11 'italian':...
```

6.9 Cleaning up

It's always good to have R disconnect from the database

```
dbDisconnect(con)
```

Stop the container and show that the container is still there, so can be started again.

```
system2('docker', 'stop sql-pet',
        stdout = TRUE, stderr = TRUE)
```

```
## [1] "sql-pet"
```

```
# show that the container still exists even though it's not running
psout <- system2("docker", "ps -a", stdout = TRUE)
psout[grepl(x = psout, pattern = 'sql-pet')]
```

```
## [1] "569798b50767      postgres:10      \"docker-entrypoint.s…\"  19 seconds ago    Exited (137) Less t
```

Next time, you can just use this command to start the container:

```
system2("docker",  "start sql-pet", stdout = TRUE, stderr = TRUE)
```

And once stopped, the container can be removed with:

```
system2("docker",  "rm sql-pet", stdout = TRUE, stderr = TRUE)
```

6.10 Using the `sql-pet` container in the rest of the book

After this point in the book, we assume that Docker is up and that we can always start up our *sql-pet database* with:

```
system2("docker",  "start sql-pet", stdout = TRUE, stderr = TRUE)
```

Chapter 7

Introduction: Postgres queries from R (10)

Note that `tidyverse`, `DBI`, `RPostgres`, `glue`, and `knitr` are loaded. Also, we've sourced the `[db-login-batch-code.R]('r-database-docker/book-src/db-login-batch-code.R')` file which is used to log in to PostgreSQL.

7.1 Basics

- keeping passwords secure
- Coverage in this book. There are many SQL tutorials that are available. For example, we are drawing some materials from a tutorial we recommend. In particular, we will not replicate the lessons there, which you might want to complete. Instead, we are showing strategies that are recommended for R users. That will include some translations of queries that are discussed there.

7.2 Ask yourself about what you are aiming for?

- differences between production and data warehouse environments
- learning to keep your DBAs happy
  - You are your own DBA in this simulation, so you can wreak havoc and learn from it, but you can learn to be DBA-friendly here.
  - in the end it's the subject-matter experts that understand your data, but you have to work with your DBAs first

7.3 Get some basic information about your database

Assume that the Docker container with PostgreSQL and the dvdrental database are ready to go.

```
system2("docker",  "start sql-pet", stdout = TRUE, stderr = TRUE)
```

```
## [1] "sql-pet"
```

```
con <- wait_for_postgres(user = Sys.getenv("DEFAULT_POSTGRES_USER_NAME"),
                         password = Sys.getenv("DEFAULT_POSTGRES_PASSWORD"),
                         dbname = "dvdrental",
                         seconds_to_test = 10)
```

Chapter 8

Simple queries (11)

8.1 Some extra handy libraries

Here are some packages that we find handy in the preliminary investigation of a database (or a problem that involves data from a database).

```
library(glue)
library(skimr)
```

8.2 Basic investigation

- Need both familiarity with the data and a focus question
  - An interative process
  - Each informs the other
- R tools for data investigation
  - glimpse
  - str
  - View and kable
- overview investigation: do you understand your data
  - documentation and its limits
  - what's *missing* from the database: (columns, records, cells)
- find out how the data is used by those who enter it and others who've used it before
  - why is there missing data?

8.3 Using Dplyr

We already started, but that's OK.

8.3.1 finding out what's in the database

- DBI / RPostgres packages
- R tools like glimpse, skimr, kable.
- Tutorials like: https://suzan.rbind.io/tags/dplyr/
- Benjamin S. Baumer, A Grammar for Reproducible and Painless Extract-Transform-Load Operations on Medium Data: https://arxiv.org/pdf/1708.07073

8.3.2 sample query

- rental
- date subset

8.3.3 Subset: only retrieve what you need

- Columns
- Rows
  - number of row
  - specific rows
- Counts & stats

8.3.4 Make the server do as much work as you can

discuss this simple example? http://www.postgresqltutorial.com/postgresql-left-join/

- dplyr joins on the server side
- Where you put `(collect(n = Inf))` really matters

8.4 What is dplyr sending to the server?

- show_query as a first draft

8.5 Writing SQL queries directly to the DBMS

- dbquery
- Glue for constructing SQL statements
  - parameterizing SQL queries

8.6 Chosing between dplyr and native SQL

- performance considerations: first get the right data, then worry about performance
- Trade offs between leaving the data in PostgreSQL vs what's kept in R:
  - browsing the data
  - larger samples and complete tables
  - using what you know to write efficient queries that do most of the work on the server

---

- left join staff

- left join customer

- dplyr joins in the R

Chapter 9

Leftovers (12)

Most of the content in this file has een moved elsewhere.

9.1 Some extra handy libraries

Here are some packages that we find handy in the preliminary investigation of a database (or a problem that involves data from a database).

```
library(glue)
```

```
##
## Attaching package: 'glue'
```

```
## The following object is masked from 'package:dplyr':
##
##     collapse
```

```
library(skimr)
```

9.2 More topics

- Check this against Aaron Makubuya's workshop at the Cascadia R Conf.

9.3 Standards for production jobs

- writing tests for your queries

Chapter 10

Joins and complex queries (13)

Libraries loaded and functions are loaded

10.1 Verify Docker is up and running:

```
result <- system2("docker", "version", stdout = TRUE, stderr = TRUE)
result
```

```
##  [1] "Client:"
##  [2] " Version:           18.06.1-ce"
##  [3] " API version:       1.38"
##  [4] " Go version:        go1.10.3"
##  [5] " Git commit:        e68fc7a"
##  [6] " Built:             Tue Aug 21 17:21:31 2018"
##  [7] " OS/Arch:           darwin/amd64"
##  [8] " Experimental:      false"
##  [9] ""
## [10] "Server:"
## [11] " Engine:"
## [12] "  Version:          18.06.1-ce"
## [13] "  API version:      1.38 (minimum version 1.12)"
## [14] "  Go version:       go1.10.3"
## [15] "  Git commit:       e68fc7a"
## [16] "  Built:            Tue Aug 21 17:29:02 2018"
## [17] "  OS/Arch:          linux/amd64"
## [18] "  Experimental:     true"
```

verify pet DB is available, it may be stopped.

```
result <- system2("docker", "ps -a", stdout = TRUE, stderr = TRUE)
result
```

```
## [1] "CONTAINER ID    IMAGE          COMMAND            CREATED         STATUS         PORTS
## [2] "569798b50767     postgres:10      \"docker-entrypoint.s…\"  26 seconds ago    Up 5 seconds     0.
```

```
any(grepl('Up .+pet$',result))
```

```
## [1] TRUE
```

Start up the `docker-pet` container

```
result <- system2("docker", "start sql-pet", stdout = TRUE, stderr = TRUE)
result
```

```
## [1] "sql-pet"
```

now connect to the database with R

```
# need to wait for Docker & Postgres to come up before connecting.

con <- wait_for_postgres(user = Sys.getenv("DEFAULT_POSTGRES_USER_NAME"),
                         password = Sys.getenv("DEFAULT_POSTGRES_PASSWORD"),
                         dbname = "dvdrental",
                         seconds_to_test = 10)
```

```
## select examples
##    dbGetQuery returns the entire result set as a data frame.
##        For large returned datasets, complex or inefficient SQL statements, this may take a
##        long time.

##      dbSendQuery: parses, compiles, creates the optimized execution plan.
##          dbFetch: Execute optimzed execution plan and return the dataset.
##    dbClearResult:remove pending query results from the database to your R environment
```

How many customers are there in the DVD Rental System

```
rs1 <- dbGetQuery(con,'select * from customer;')
kable(head(rs1))
```

customer_id	store_id	first_name	last_name	email	address_id	activebool	crea
524	1	Jared	Ely	jared.ely@sakilacustomer.org	530	TRUE	200
1	1	Mary	Smith	mary.smith@sakilacustomer.org	5	TRUE	200
2	1	Patricia	Johnson	patricia.johnson@sakilacustomer.org	6	TRUE	200
3	1	Linda	Williams	linda.williams@sakilacustomer.org	7	TRUE	200
4	2	Barbara	Jones	barbara.jones@sakilacustomer.org	8	TRUE	200
5	1	Elizabeth	Brown	elizabeth.brown@sakilacustomer.org	9	TRUE	200

```
pco <- dbSendQuery(con,'select * from customer;')
rs2  <- dbFetch(pco)
dbClearResult(pco)
kable(head(rs2))
```

customer_id	store_id	first_name	last_name	email	address_id	activebool	crea
524	1	Jared	Ely	jared.ely@sakilacustomer.org	530	TRUE	200
1	1	Mary	Smith	mary.smith@sakilacustomer.org	5	TRUE	200
2	1	Patricia	Johnson	patricia.johnson@sakilacustomer.org	6	TRUE	200
3	1	Linda	Williams	linda.williams@sakilacustomer.org	7	TRUE	200
4	2	Barbara	Jones	barbara.jones@sakilacustomer.org	8	TRUE	200
5	1	Elizabeth	Brown	elizabeth.brown@sakilacustomer.org	9	TRUE	200

```
# insert yourself as a new customer
dbExecute(con
         ,"insert into customer
                (store_id,first_name,last_name,email,address_id
                ,activebool,create_date,last_update,active)
          values(2,'Sophie','Yang','dodreamdo@yahoo.com',1,TRUE,'2018-09-13','2018-09-13',1)
          "
         )
```

```
## [1] 0
## anti join -- Find customers who have never rented a movie.

rs <- dbGetQuery(con,
                 "select c.first_name
                        ,c.last_name
                        ,c.email
                    from customer c
                         left outer join rental r
                              on c.customer_id = r.customer_id
                   where r.rental_id is null;
                 "
                 )
head(rs)
```

```
##   first_name last_name                email
## 1     Sophie      Yang dodreamdo@yahoo.com
```

```
## how many films and languages exist in the DVD rental application
rs <- dbGetQuery(con,
                "      select 'film' table_name,count(*) count from film
                 union select 'language' table_name,count(*) count from language
               ;
                "
                )
head(rs)
```

```
##   table_name count
## 1       film  1000
## 2   language     6
```

```
## what is the film distribution based on language

rs <- dbGetQuery(con,
                "select l.language_id id
                       ,l.name
                       ,sum(case when f.language_id is not null then 1 else 0 end) total
                   from language l
                        full outer join film f
                             on l.language_id = f.language_id
                  group by l.language_id,l.name
                  order by l.name;
                 ;
                "
                )
head(rs)
```

```
##   id                  name total
## 1  1 English               1000
## 2  5 French                   0
## 3  6 German                   0
## 4  2 Italian                  0
## 5  3 Japanese                 0
## 6  4 Mandarin                 0
```

```
## Store analysis
### which store has had more rentals and income
```

```
rs <- dbGetQuery(con,
                "select *
                 from (      select 'actor' tbl_name,count(*) from actor
                       union select 'category' tbl_name,count(*) from category
                       union select 'film' tbl_name,count(*) from film
                       union select 'film_actor' tbl_name,count(*) from film_actor
                       union select 'film_category' tbl_name,count(*) from film_category
                       union select 'language' tbl_name,count(*) from language
                       union select 'inventory' tbl_name,count(*) from inventory
                       union select 'rental' tbl_name,count(*) from rental
                       union select 'payment' tbl_name,count(*) from payment
                       union select 'staff' tbl_name,count(*) from staff
                       union select 'customer' tbl_name,count(*) from customer
                       union select 'address' tbl_name,count(*) from address
                       union select 'city' tbl_name,count(*) from city
                       union select 'country' tbl_name,count(*) from country
                       union select 'store' tbl_name,count(*) from store
                       ) counts
                  order by tbl_name
                 ;
                "
                )
head(rs)
```

```
##   tbl_name count
## 1    actor   200
## 2  address   603
## 3 category    16
## 4     city   600
## 5  country   109
## 6 customer   600
```

```
## Store analysis
### which store has the largest income stream
rs <- dbGetQuery(con,
                "select store_id,sum(amount) amt,count(*) cnt
                   from payment p
                        join staff s
                          on p.staff_id = s.staff_id
                 group by store_id order by 2 desc
                 ;
                "
                )
head(rs)
```

```
##   store_id      amt  cnt
## 1        2 31059.92 7304
## 2        1 30252.12 7292
```

```
## Store analysis
### How many rentals have not been paid
### How many rentals have been paid
### How much has been paid
### What is the average price/movie
### Estimate the outstanding balance
```

```
rs <- dbGetQuery(con,
                "select sum(case when payment_id is null then 1 else 0 end) missing
                       ,sum(case when payment_id is not null then 1 else 0 end) found
                       ,sum(p.amount) amt
                       ,count(*) cnt
                       ,round(sum(p.amount)/sum(case when payment_id is not null then 1 else 0 end),2) a
                       ,round(round(sum(p.amount)/sum(case when payment_id is not null then 1 else 0 en
                                  * sum(case when payment_id is null then 1 else 0 end),2) est_balance
                   from rental r
                        left outer join payment p
                          on r.rental_id = p.rental_id
                 ;
                "
                )
head(rs)
```

```
##   missing found      amt   cnt avg_price est_balance
## 1    1452 14596 61312.04 16048       4.2      6098.4
```

```
### what is the actual outstanding balance

rs <- dbGetQuery(con,
                "select sum(f.rental_rate) open_amt,count(*) count
                   from rental r
                        left outer join payment p
                          on r.rental_id = p.rental_id
                        join inventory i
                          on r.inventory_id = i.inventory_id
                        join film f
                          on i.film_id = f.film_id
                  where p.rental_id is null
                 ;"
                )
head(rs)
```

```
##   open_amt count
## 1  4297.48  1452
```

```
### Rank customers with highest open amounts

rs <- dbGetQuery(con,
                "select c.customer_id,c.first_name,c.last_name,sum(f.rental_rate) open_amt,count(*) cou
                   from rental r
                        left outer join payment p
                          on r.rental_id = p.rental_id
                        join inventory i
                          on r.inventory_id = i.inventory_id
                        join film f
                          on i.film_id = f.film_id
                        join customer c
                          on r.customer_id = c.customer_id
                  where p.rental_id is null
                  group by c.customer_id,c.first_name,c.last_name
                  order by open_amt desc
                  limit 25
```

```
                 ;"
                )
head(rs)
```

```
##   customer_id first_name last_name open_amt count
## 1         293        Mae  Fletcher    35.90    10
## 2         307     Joseph       Joy    31.90    10
## 3         316     Steven    Curley    31.90    10
## 4         299      James    Gannon    30.91     9
## 5         274      Naomi  Jennings    29.92     8
## 6         326       Jose    Andrew    28.93     7
```

```
### what film has been rented the most
rs <- dbGetQuery(con,
                "select i.film_id,f.title,rental_rate,sum(rental_rate) revenue,count(*) count  --16044
                   from rental r
                        join inventory i
                          on r.inventory_id = i.inventory_id
                        join film f
                          on i.film_id = f.film_id
                 group by i.film_id,f.title,rental_rate
                 order by count desc
                 ;"
                )
head(rs)
```

```
##   film_id               title rental_rate revenue count
## 1     103  Bucket Brotherhood        4.99  169.66    34
## 2     738    Rocketeer Mother        0.99   32.67    33
## 3     382      Grit Clockwork        0.99   31.68    32
## 4     767       Scalawag Duck        4.99  159.68    32
## 5     489      Juggler Hardly        0.99   31.68    32
## 6     730 Ridgemont Submarine        0.99   31.68    32
```

```
### what film has been generated the most revenue assuming all amounts are collected
rs <- dbGetQuery(con,
                "select i.film_id,f.title,rental_rate
                       ,sum(rental_rate) revenue,count(*) count  --16044
                   from rental r
                        join inventory i
                          on r.inventory_id = i.inventory_id
                        join film f
                          on i.film_id = f.film_id
                 group by i.film_id,f.title,rental_rate
                 order by revenue desc
                 ;"
                )
head(rs)
```

```
##   film_id              title rental_rate revenue count
## 1     103 Bucket Brotherhood        4.99  169.66    34
## 2     767      Scalawag Duck        4.99  159.68    32
## 3     973          Wife Turn        4.99  154.69    31
## 4      31      Apache Divine        4.99  154.69    31
## 5     369  Goodfellas Salute        4.99  154.69    31
## 6    1000          Zorro Ark        4.99  154.69    31
```

```
### which films are in one store but not the other.
rs <- dbGetQuery(con,
                "select coalesce(i1.film_id,i2.film_id) film_id
                       ,f.title,f.rental_rate,i1.store_id,i1.count,i2.store_id,i2.count
                   from     (select film_id,store_id,count(*) count
                               from inventory where store_id = 1
                             group by film_id,store_id) as i1
                         full outer join
                            (select film_id,store_id,count(*) count
                               from inventory where store_id = 2
                             group by film_id,store_id
                            ) as i2
                           on i1.film_id = i2.film_id
                         join film f
                           on coalesce(i1.film_id,i2.film_id) = f.film_id
                  where i1.film_id is null or i2.film_id is null
                 order by f.title  ;
               "
                )
head(rs)
```

```
##   film_id               title rental_rate store_id count store_id..6
## 1       2      Ace Goldfinger        4.99       NA  <NA>           2
## 2       3    Adaptation Holes        2.99       NA  <NA>           2
## 3       5         African Egg        2.99       NA  <NA>           2
## 4       8     Airport Pollock        4.99       NA  <NA>           2
## 5      13         Ali Forever        4.99       NA  <NA>           2
## 6      20 Amelie Hellfighters        4.99        1     3          NA
##   count..7
## 1        3
## 2        4
## 3        3
## 4        4
## 5        4
## 6     <NA>
```

```
# Compute the outstanding balance.
rs <- dbGetQuery(con,
                "select sum(f.rental_rate) open_amt,count(*) count
                   from rental r
                        left outer join payment p
                          on r.rental_id = p.rental_id
                        join inventory i
                          on r.inventory_id = i.inventory_id
                        join film f
                          on i.film_id = f.film_id
                  where p.rental_id is null
                 ;"
                )
head(rs)
```

```
##   open_amt count
## 1  4297.48  1452
```

list what's there

```
dbListTables(con)
```

```
##  [1] "actor_info"                 "customer_list"
##  [3] "film_list"                  "nicer_but_slower_film_list"
##  [5] "sales_by_film_category"     "staff"
##  [7] "sales_by_store"             "staff_list"
##  [9] "category"                   "film_category"
## [11] "country"                    "actor"
## [13] "language"                   "inventory"
## [15] "payment"                    "rental"
## [17] "city"                       "store"
## [19] "film"                       "address"
## [21] "film_actor"                 "customer"
```

Clean up

```
# dbRemoveTable(con, "cars")
# dbRemoveTable(con, "mtcars")
# dbRemoveTable(con, "cust_movies")

# diconnect from the db
dbDisconnect(con)

result <- system2("docker", "stop sql-pet", stdout = TRUE, stderr = TRUE)
result
```

```
## [1] "sql-pet"
```

Chapter 11

Postgres Examples, part B (14)

11.1 Verify Docker is up and running:

```
result <- system2("docker", "version", stdout = TRUE, stderr = TRUE)
result
```

```
##  [1] "Client:"
##  [2] " Version:           18.06.1-ce"
##  [3] " API version:       1.38"
##  [4] " Go version:        go1.10.3"
##  [5] " Git commit:        e68fc7a"
##  [6] " Built:             Tue Aug 21 17:21:31 2018"
##  [7] " OS/Arch:           darwin/amd64"
##  [8] " Experimental:      false"
##  [9] ""
## [10] "Server:"
## [11] " Engine:"
## [12] "  Version:          18.06.1-ce"
## [13] "  API version:      1.38 (minimum version 1.12)"
## [14] "  Go version:       go1.10.3"
## [15] "  Git commit:       e68fc7a"
## [16] "  Built:            Tue Aug 21 17:29:02 2018"
## [17] "  OS/Arch:          linux/amd64"
## [18] "  Experimental:     true"
```

verify pet DB is available, it may be stopped.

```
result <- system2("docker", "ps -a", stdout = TRUE, stderr = TRUE)
result
```

```
## [1] "CONTAINER ID    IMAGE          COMMAND            CREATED         STATUS              PORTS
## [2] "569798b50767     postgres:10      \"docker-entrypoint.s…\"  30 seconds ago    Exited (0) 2 second
```

```
any(grepl('Up .+pet$',result))
```

```
## [1] FALSE
```

Start up the `docker-pet` container

```
result <- system2("docker", "start sql-pet", stdout = TRUE, stderr = TRUE)
result
```

```
## [1] "sql-pet"
```

now connect to the database with R

```
con <- wait_for_postgres(user = Sys.getenv("DEFAULT_POSTGRES_USER_NAME"),
                         password = Sys.getenv("DEFAULT_POSTGRES_PASSWORD"),
                         dbname = "dvdrental",
                         seconds_to_test = 10)
```

All of the material from this file has moved to files 71, 72, and 73.

Clean up

```
# dbRemoveTable(con, "cars")
# dbRemoveTable(con, "mtcars")
# dbRemoveTable(con, "cust_movies")

# diconnect from the db
dbDisconnect(con)

result <- system2("docker", "stop sql-pet", stdout = TRUE, stderr = TRUE)
result
```

```
## [1] "sql-pet"
```

Chapter 12

Getting metadata about and from the database (21)

Note that `tidyverse`, `DBI`, `RPostgres`, `glue`, and `knitr` are loaded. Also, we've sourced the `db-login-batch-code.R` file which is used to log in to PostgreSQL.

12.1 Look at the data and its metadata

Assume that the Docker container with PostgreSQL and the dvdrental database are ready to go.

```
system2("docker",  "start sql-pet", stdout = TRUE, stderr = TRUE)
```

```
## [1] "sql-pet"
```

```
con <- wait_for_postgres(user = Sys.getenv("DEFAULT_POSTGRES_USER_NAME"),
                         password = Sys.getenv("DEFAULT_POSTGRES_PASSWORD"),
                         dbname = "dvdrental",
                         seconds_to_test = 10)
```

So far in this books we've most often looked at the data by listing a few observations or using a tool like `glimpse`.

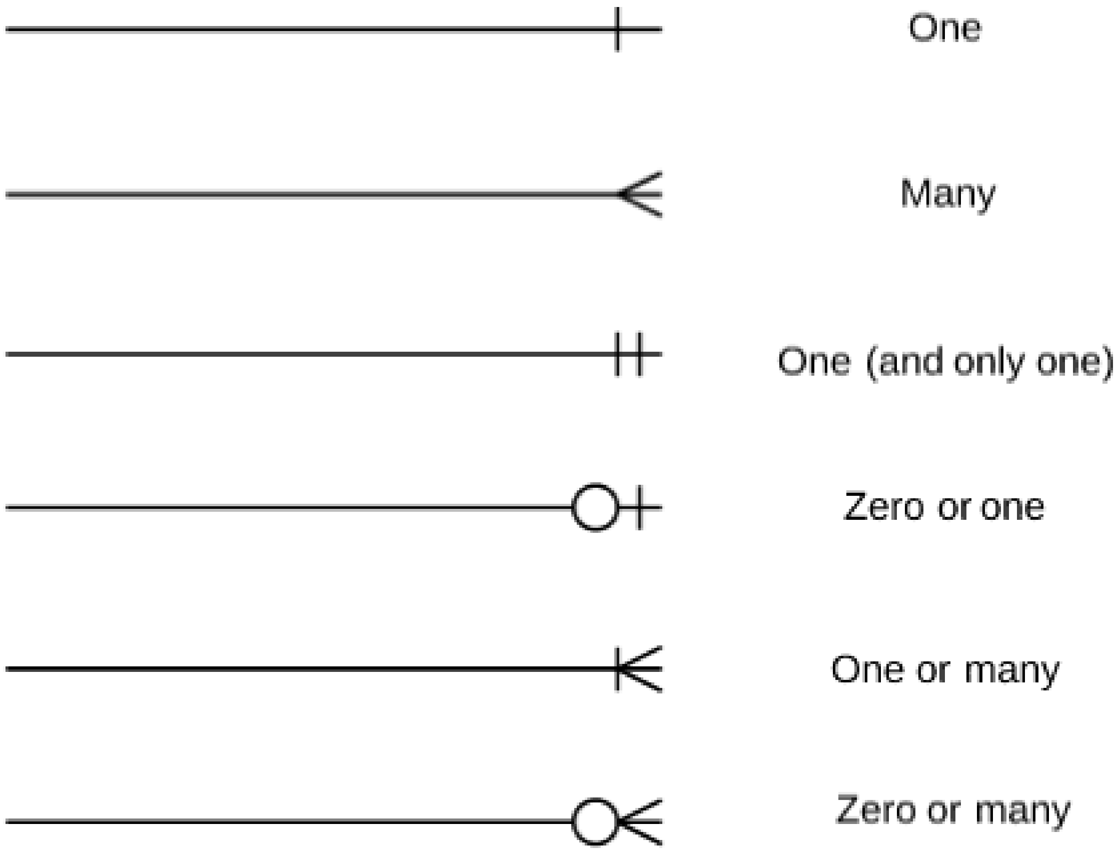
```
rental <- tbl(con, "rental")

kable(head(rental))
```

rental_id	rental_date	inventory_id	customer_id	return_date	staff_id	last_update
2	2005-05-24 22:54:33	1525	459	2005-05-28 19:40:33	1	2006-02-16 02:30:53
3	2005-05-24 23:03:39	1711	408	2005-06-01 22:12:39	1	2006-02-16 02:30:53
4	2005-05-24 23:04:41	2452	333	2005-06-03 01:43:41	2	2006-02-16 02:30:53
5	2005-05-24 23:05:21	2079	222	2005-06-02 04:33:21	1	2006-02-16 02:30:53
6	2005-05-24 23:08:07	2792	549	2005-05-27 01:32:07	1	2006-02-16 02:30:53
7	2005-05-24 23:11:53	3995	269	2005-05-29 20:34:53	2	2006-02-16 02:30:53

```
glimpse(rental)
```

```
## Observations: ??
## Variables: 7
## $ rental_id    <int> 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 1...
## $ rental_date  <dttm> 2005-05-24 22:54:33, 2005-05-24 23:03:39, 2005-0...
## $ inventory_id <int> 1525, 1711, 2452, 2079, 2792, 3995, 2346, 2580, 1...
## $ customer_id  <int> 459, 408, 333, 222, 549, 269, 239, 126, 399, 142,...
## $ return_date  <dttm> 2005-05-28 19:40:33, 2005-06-01 22:12:39, 2005-0...
## $ staff_id     <int> 1, 1, 2, 1, 1, 2, 2, 1, 2, 2, 2, 1, 1, 1, 2, 1, 2...
## $ last_update  <dttm> 2006-02-16 02:30:53, 2006-02-16 02:30:53, 2006-0...
```

For large or complex databases, however, you need to use both the available documentation for your database (e.g., the dvdrental dataase) and the other empirical tools that are available. For example it's worth learning to interpret the symbols in an Entity Relationship Diagram:



The `information_schema` is a trove of information *about* the database. Its format is more or less consistent across the different SQL implementations that are available. Here we explore some of what's available using several different methods. Postgres stores a lot of metadata.

12.1.1 The information_schema with dbplyr

For this chapter R needs the `dbplyr` package to access alternate schemas. A schema is an object that contains one or more tables. Most often there will be a default schema, but to access the metadata, you need to explicitly specify which schema contains the data you want.

```
library(dbplyr)
```

```
##
## Attaching package: 'dbplyr'

## The following objects are masked from 'package:dplyr':
```

```
##
##     ident, sql
```

```
columns_info_schema_table <- tbl(con, in_schema("information_schema", "columns"))

columns_info_schema_info <- columns_info_schema_table %>%
  select(table_schema, table_name, column_name, data_type, ordinal_position,
         column_default, character_maximum_length) %>%
  collect(n = Inf)

columns_info_schema_info
```

```
## # A tibble: 1,855 x 7
##    table_schema table_name column_name data_type ordinal_position
##    <chr>        <chr>      <chr>       <chr>                <int>
##  1 pg_catalog   pg_proc    proname     name                     1
##  2 pg_catalog   pg_proc    pronamespa~ oid                      2
##  3 pg_catalog   pg_proc    proowner    oid                      3
##  4 pg_catalog   pg_proc    prolang     oid                      4
##  5 pg_catalog   pg_proc    procost     real                     5
##  6 pg_catalog   pg_proc    prorows     real                     6
##  7 pg_catalog   pg_proc    provariadic oid                      7
##  8 pg_catalog   pg_proc    protransfo~ regproc                  8
##  9 pg_catalog   pg_proc    proisagg    boolean                  9
## 10 pg_catalog   pg_proc    proiswindow boolean                 10
## # ... with 1,845 more rows, and 2 more variables: column_default <chr>,
## #   character_maximum_length <int>
```

For the moment we're going to drop everything except the columns that are in the `public` schema.

```
public_table_columns <- columns_info_schema_info %>%
  filter(table_schema == "public") %>%
  select(-table_schema)

public_table_columns
```

```
## # A tibble: 128 x 6
##    table_name column_name data_type ordinal_position column_default
##    <chr>      <chr>       <chr>                <int> <chr>
##  1 customer   store_id    smallint                 2 <NA>
##  2 customer   first_name  characte~                3 <NA>
##  3 customer   last_name   characte~                4 <NA>
##  4 customer   email       characte~                5 <NA>
##  5 customer   address_id  smallint                 6 <NA>
##  6 customer   active      integer                 10 <NA>
##  7 customer   customer_id integer                  1 nextval('cust~
##  8 customer   activebool  boolean                  7 true
##  9 customer   create_date date                     8 ('now'::text)~
## 10 customer   last_update timestam~                9 now()
## # ... with 118 more rows, and 1 more variable:
## #   character_maximum_length <int>
```

Pull out some rough-and-ready but useful statistics about your database. Since we are in SQL-land we talk about variables as `columns`.

Start with a list of tables names and a count of the number of columns that each one contains.

```
public_table_columns %>%
  count(table_name, sort = TRUE) %>%
  rename(number_of_columns = n) %>%
  as.data.frame() # we want to look at all of them, so bypass the nice tibble row limit
```

```
##                    table_name number_of_columns
## 1                        film                13
## 2                       staff                11
## 3                    customer                10
## 4               customer_list                 9
## 5                     address                 8
## 6                   film_list                 8
## 7  nicer_but_slower_film_list                 8
## 8                  staff_list                 8
## 9                      rental                 7
## 10                    payment                 6
## 11                      actor                 4
## 12                 actor_info                 4
## 13                       city                 4
## 14                  inventory                 4
## 15                      store                 4
## 16                   category                 3
## 17                    country                 3
## 18                 film_actor                 3
## 19              film_category                 3
## 20                   language                 3
## 21             sales_by_store                 3
## 22     sales_by_film_category                 2
```

How many column names are shared across tables (or duplicated)?

```
public_table_columns %>% count(column_name, sort = TRUE) %>% filter(n > 1)
```

```
## # A tibble: 34 x 2
##    column_name     n
##    <chr>       <int>
##  1 last_update    14
##  2 address_id      4
##  3 film_id         4
##  4 first_name      4
##  5 last_name       4
##  6 name            4
##  7 store_id        4
##  8 actor_id        3
##  9 address         3
## 10 category        3
## # ... with 24 more rows
```

How many column names are unique?

```
public_table_columns %>% count(column_name) %>% filter(n > 1)
```

```
## # A tibble: 34 x 2
##    column_name     n
##    <chr>       <int>
##  1 active          2
##  2 actor_id        3
##  3 actors          2
##  4 address         3
##  5 address_id      4
##  6 category        3
##  7 category_id     2
##  8 city            3
##  9 city_id         2
## 10 country         3
## # ... with 24 more rows
```

What data types are found in the database?

```
public_table_columns %>% count(data_type)
```

```
## # A tibble: 13 x 2
##    data_type                       n
##    <chr>                       <int>
##  1 ARRAY                           1
##  2 boolean                         2
##  3 bytea                           1
##  4 character                       1
##  5 character varying              36
##  6 date                            1
##  7 integer                        22
##  8 numeric                         7
##  9 smallint                       25
## 10 text                           11
## 11 timestamp without time zone    17
## 12 tsvector                        1
## 13 USER-DEFINED                    3
```

12.1.2 Submitting SQL statements directly

```
table_schema_query  <- glue("SELECT ",
  "table_name, column_name, data_type, ordinal_position, column_default, character_maximum_length",
  " FROM information_schema.columns ",
  "WHERE table_schema = 'public'")

  rental_meta_data  <- dbGetQuery(con, table_schema_query)

glimpse(rental_meta_data)
```

```
## Observations: 128
## Variables: 6
## $ table_name               <chr> "actor_info", "actor_info", "actor_in...
## $ column_name              <chr> "actor_id", "first_name", "last_name"...
## $ data_type                <chr> "integer", "character varying", "char...
## $ ordinal_position         <int> 1, 2, 3, 4, 1, 2, 3, 4, 5, 6, 7, 8, 9...
## $ column_default           <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, N...
## $ character_maximum_length <int> NA, 45, 45, NA, NA, NA, 50, 10, 20, 5...
```

```
## Get list of database objects
rs <- dbGetQuery(con
                 ,"select table_catalog,table_schema,table_name,table_type
```

```
                     from information_schema.tables
                    where table_schema not in ('pg_catalog','information_schema')
                   order by table_name
                  ;"
                  )
# Get list of tables
kable(dbListTables(con))
```

x
actor_info
customer_list
film_list
nicer_but_slower_film_list
sales_by_film_category
staff
sales_by_store
staff_list
category
film_category
country
actor
language
inventory
payment
rental
city
store
film
address
film_actor
customer

```
rs <- dbGetQuery(con
                 ,"select table_catalog||'.'||table_schema||'.'||table_name table_name
                         ,column_name,ordinal_position seq --,data_type
                         ,case when data_type = 'character varying'
                               then data_type || '('|| character_maximum_length||')'
                               when data_type = 'real'
                               then data_type || '(' || numeric_precision ||','||numeric_precision_radi
                               else data_type
                          end data_type
--                         ,character_maximum_length,numeric_precision,numeric_precision_radix
                     from information_schema.columns
                    where table_name in (select table_name
                                           from information_schema.tables
                                         where table_schema not in ('pg_catalog','information_schema')
                                         )
                   order by table_name,ordinal_position;

                  ;"
                  )
kable(head(rs, n = 20))
```

table_name	column_name	seq	data_type
dvdrental.public.actor	actor_id	1	integer
dvdrental.public.actor	first_name	2	character varying(45)
dvdrental.public.actor	last_name	3	character varying(45)
dvdrental.public.actor	last_update	4	timestamp without time zone
dvdrental.public.actor_info	actor_id	1	integer
dvdrental.public.actor_info	first_name	2	character varying(45)
dvdrental.public.actor_info	last_name	3	character varying(45)
dvdrental.public.actor_info	film_info	4	text
dvdrental.public.address	address_id	1	integer
dvdrental.public.address	address	2	character varying(50)
dvdrental.public.address	address2	3	character varying(50)
dvdrental.public.address	district	4	character varying(20)
dvdrental.public.address	city_id	5	smallint
dvdrental.public.address	postal_code	6	character varying(10)
dvdrental.public.address	phone	7	character varying(20)
dvdrental.public.address	last_update	8	timestamp without time zone
dvdrental.public.category	category_id	1	integer
dvdrental.public.category	name	2	character varying(25)
dvdrental.public.category	last_update	3	timestamp without time zone
dvdrental.public.city	city_id	1	integer

There are {r dim(rs)[1]} rows in the catalog.

```
rs <- dbGetQuery(con,
"
--SELECT conrelid::regclass as table_from
select table_catalog||'.'||table_schema||'.'||table_name table_name
,conname,pg_catalog.pg_get_constraintdef(r.oid, true) as condef
FROM information_schema.columns c,pg_catalog.pg_constraint r
WHERE 1 = 1 --r.conrelid = '16485'
  AND r.contype  in ('f','p') ORDER BY 1
;"
)


kable(head(rs))
```

table_name	conname	condef
dvdrental.information_schema.administrable_role_authorizations	actor_pkey	PRIMARY KEY (actor_id)
dvdrental.information_schema.administrable_role_authorizations	actor_pkey	PRIMARY KEY (actor_id)
dvdrental.information_schema.administrable_role_authorizations	actor_pkey	PRIMARY KEY (actor_id)
dvdrental.information_schema.administrable_role_authorizations	country_pkey	PRIMARY KEY (country_id)
dvdrental.information_schema.administrable_role_authorizations	country_pkey	PRIMARY KEY (country_id)
dvdrental.information_schema.administrable_role_authorizations	country_pkey	PRIMARY KEY (country_id)

```
rs <- dbGetQuery(con,
"select conrelid::regclass as table_from
      ,c.conname
      ,pg_get_constraintdef(c.oid)
  from pg_constraint c
  join pg_namespace n on n.oid = c.connamespace
 where c.contype in ('f','p')
   and n.nspname = 'public'
order by conrelid::regclass::text, contype DESC;
")
kable(head(rs))
```

table_from	conname	pg_get_constraintdef
actor	actor_pkey	PRIMARY KEY (actor_id)
address	address_pkey	PRIMARY KEY (address_id)
address	fk_address_city	FOREIGN KEY (city_id) REFERENCES city(city_id)
category	category_pkey	PRIMARY KEY (category_id)
city	city_pkey	PRIMARY KEY (city_id)
city	fk_city	FOREIGN KEY (country_id) REFERENCES country(country_id)

```
dim(rs)[1]
```

```
## [1] 33
```

```
rs <- dbGetQuery(con,
"SELECT r.*,
  pg_catalog.pg_get_constraintdef(r.oid, true) as condef
FROM pg_catalog.pg_constraint r
WHERE 1=1 --r.conrelid = '16485' AND r.contype = 'f' ORDER BY 1;
")

head(rs)
```

```
##                         conname connamespace contype condeferrable
## 1 cardinal_number_domain_check        12703       c         FALSE
## 2              yes_or_no_check        12703       c         FALSE
## 3                   year_check         2200       c         FALSE
## 4                   actor_pkey         2200       p         FALSE
## 5                 address_pkey         2200       p         FALSE
## 6                category_pkey         2200       p         FALSE
##   condeferred convalidated conrelid contypid conindid confrelid
## 1       FALSE         TRUE        0    12716        0         0
## 2       FALSE         TRUE        0    12724        0         0
## 3       FALSE         TRUE        0    16397        0         0
## 4       FALSE         TRUE    16420        0    16555         0
## 5       FALSE         TRUE    16461        0    16557         0
## 6       FALSE         TRUE    16427        0    16559         0
##   confupdtype confdeltype confmatchtype conislocal coninhcount
## 1                                             TRUE           0
## 2                                             TRUE           0
## 3                                             TRUE           0
## 4                                             TRUE           0
## 5                                             TRUE           0
## 6                                             TRUE           0
##   connoinherit conkey confkey conpfeqop conppeqop conffeqop conexclop
## 1        FALSE   <NA>    <NA>      <NA>      <NA>      <NA>      <NA>
## 2        FALSE   <NA>    <NA>      <NA>      <NA>      <NA>      <NA>
## 3        FALSE   <NA>    <NA>      <NA>      <NA>      <NA>      <NA>
## 4         TRUE    {1}    <NA>      <NA>      <NA>      <NA>      <NA>
## 5         TRUE    {1}    <NA>      <NA>      <NA>      <NA>      <NA>
## 6         TRUE    {1}    <NA>      <NA>      <NA>      <NA>      <NA>
##
## 1
## 2 {SCALARARRAYOPEXPR :opno 98 :opfuncid 67 :useOr true :inputcollid 100 :args ({RELABELTYPE :arg {COERCET
## 3                                                                         {BOOLEXPR :boolop and :args
## 4
## 5
## 6
##                                                                                   consrc
## 1                                                                           (VALUE >= 0)
## 2 ((VALUE)::text = ANY ((ARRAY['YES'::character varying, 'NO'::character varying])::text[]))
## 3                                              ((VALUE >= 1901) AND (VALUE <= 2155))
## 4                                                                                   <NA>
## 5                                                                                   <NA>
## 6                                                                                   <NA>
##                                                                                   condef
## 1                                                                     CHECK (VALUE >= 0)
## 2 CHECK (VALUE::text = ANY (ARRAY['YES'::character varying, 'NO'::character varying]::text[]))
## 3                                              CHECK (VALUE >= 1901 AND VALUE <= 2155)
## 4                                                               PRIMARY KEY (actor_id)
## 5                                                             PRIMARY KEY (address_id)
## 6                                                            PRIMARY KEY (category_id)
```

Chapter 13

Explain queries (71)

- examining dplyr queries (show_query on the R side v EXPLAIN on the PostgreSQL side)

Start up the `docker-pet` container

```
result <- system2("docker", "start sql-pet", stdout = TRUE, stderr = TRUE)
result
```

```
## [1] "sql-pet"
```

now connect to the database with R

```
con <- wait_for_postgres(user = Sys.getenv("DEFAULT_POSTGRES_USER_NAME"),
                         password = Sys.getenv("DEFAULT_POSTGRES_PASSWORD"),
                         dbname = "dvdrental",
                         seconds_to_test = 10)
```

13.1 Performance considertations

```
## Explain a dplyr join

## Explain the quivalent SQL join
rs1 <- dbGetQuery(con
                 ,"SELECT c.*
                     FROM pg_catalog.pg_class c
                     JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
                    WHERE  n.nspname = 'public'
                      AND  c.relname = 'cust_movies'
                      AND  c.relkind = 'r'
                   ;
                 "
                 )
head(rs1)
```

```
##  [1] relname             relnamespace        reltype
##  [4] reloftype           relowner            relam
##  [7] relfilenode         reltablespace       relpages
## [10] reltuples           relallvisible       reltoastrelid
## [13] relhasindex         relisshared         relpersistence
## [16] relkind             relnatts            relchecks
## [19] relhasoids          relhaspkey          relhasrules
## [22] relhastriggers      relhassubclass      relrowsecurity
## [25] relforcerowsecurity relispopulated      relreplident
## [28] relispartition      relfrozenxid        relminmxid
## [31] relacl              reloptions          relpartbound
## <0 rows> (or 0-length row.names)
```

This came from 14-sql_pet-examples-part-b.Rmd

```
rs1 <- dbGetQuery(con,
                "explain select r.*
                   from rental r
                 ;"
                )
head(rs1)
```

```
##                                                       QUERY PLAN
## 1 Seq Scan on rental r  (cost=0.00..310.44 rows=16044 width=36)
```

```
rs2 <- dbGetQuery(con,
                "explain select count(*) count
                   from rental r
                        left outer join payment p
                          on r.rental_id = p.rental_id
                    where p.rental_id is null
                 ;")
head(rs2)
```

```
##                                                                              QUERY PLAN
## 1                                      Aggregate  (cost=2086.78..2086.80 rows=1 width=8)
## 2                              ->  Merge Anti Join  (cost=0.57..2066.73 rows=8022 width=0)
## 3                                                Merge Cond: (r.rental_id = p.rental_id)
## 4         ->  Index Only Scan using rental_pkey on rental r  (cost=0.29..1024.95 rows=16044 width=4)
## 5      ->  Index Only Scan using idx_fk_rental_id on payment p  (cost=0.29..819.23 rows=14596 width=4)
```

```
rs3 <- dbGetQuery(con,
                "explain select sum(f.rental_rate) open_amt,count(*) count
                   from rental r
                        left outer join payment p
                          on r.rental_id = p.rental_id
                        join inventory i
                          on r.inventory_id = i.inventory_id
                        join film f
                          on i.film_id = f.film_id
                    where p.rental_id is null
                 ;")
head(rs3)
```

```
##                                                                  QUERY PLAN
## 1                        Aggregate  (cost=2353.64..2353.65 rows=1 width=40)
## 2                  ->  Hash Join  (cost=205.14..2313.53 rows=8022 width=12)
## 3                                        Hash Cond: (i.film_id = f.film_id)
## 4                   ->  Hash Join  (cost=128.64..2215.88 rows=8022 width=2)
## 5                              Hash Cond: (r.inventory_id = i.inventory_id)
## 6               ->  Merge Anti Join  (cost=0.57..2066.73 rows=8022 width=4)
```

```
rs4 <- dbGetQuery(con,
                "explain select c.customer_id,c.first_name,c.last_name,sum(f.rental_rate) open_amt,coun
```

```
                   from rental r
                        left outer join payment p
                          on r.rental_id = p.rental_id
                        join inventory i
                          on r.inventory_id = i.inventory_id
                        join film f
                          on i.film_id = f.film_id
                        join customer c
                          on r.customer_id = c.customer_id
                  where p.rental_id is null
                  group by c.customer_id,c.first_name,c.last_name
                  order by open_amt desc
                 ;"
                )
head(rs4)
```

```
##                                                           QUERY PLAN
## 1                 Sort  (cost=2452.49..2453.99 rows=599 width=260)
## 2                              Sort Key: (sum(f.rental_rate)) DESC
## 3     ->  HashAggregate  (cost=2417.37..2424.86 rows=599 width=260)
## 4                                         Group Key: c.customer_id
## 5         ->  Hash Join  (cost=227.62..2357.21 rows=8022 width=232)
## 6                       Hash Cond: (r.customer_id = c.customer_id)
```

13.2 Clean up

```
# dbRemoveTable(con, "cars")
# dbRemoveTable(con, "mtcars")
# dbRemoveTable(con, "cust_movies")

# diconnect from the db
dbDisconnect(con)

result <- system2("docker", "stop sql-pet", stdout = TRUE, stderr = TRUE)
result
```

```
## [1] "sql-pet"
```

Chapter 14

SQL queries behind the scenes (72)

Start up the `docker-pet` container

```
result <- system2("docker", "start sql-pet", stdout = TRUE, stderr = TRUE)
result
```

```
## [1] "sql-pet"
```

now connect to the database with R

```
con <- wait_for_postgres(user = Sys.getenv("DEFAULT_POSTGRES_USER_NAME"),
                         password = Sys.getenv("DEFAULT_POSTGRES_PASSWORD"),
                         dbname = "dvdrental",
                         seconds_to_test = 10)
```

14.1 SQL Execution Steps

- Parse the incoming SQL query
- Compile the SQL query
- Plan/optimize the data acquisition path
- Execute the optimized query / acquire and return data

```
dbWriteTable(con, "mtcars", mtcars, overwrite = TRUE)
rs <- dbSendQuery(con, "SELECT * FROM mtcars WHERE cyl = 4")
dbFetch(rs)
```

```
##     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 1  22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 2  24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## 3  22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## 4  32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## 5  30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## 6  33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## 7  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## 8  27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## 9  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## 10 30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## 11 21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
dbClearResult(rs)
```

14.2 Passing values to SQL statements

```
#Pass one set of values with the param argument:
rs <- dbSendQuery(con,"SELECT * FROM mtcars WHERE cyl = 4")
dbFetch(rs)
```

```
##     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 1  22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## 2  24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## 3  22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## 4  32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## 5  30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## 6  33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## 7  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## 8  27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## 9  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## 10 30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## 11 21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

```
dbClearResult(rs)
```

14.3 Pass multiple sets of values with dbBind():

```
rs <- dbSendQuery(con, "SELECT * FROM mtcars WHERE cyl = $1")
dbBind(rs, list(6L)) # cyl = 6
dbFetch(rs)
```

```
##    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 1 21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## 2 21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## 3 21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## 4 18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## 5 19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## 6 17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## 7 19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
```

```
dbBind(rs, list(8L)) # cyl = 8
dbFetch(rs)
```

```
##     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## 1  18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## 2  14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## 3  16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## 4  17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## 5  15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## 6  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## 7  10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## 8  14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## 9  15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## 10 15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## 11 13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## 12 19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## 13 15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## 14 15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
```

```
dbClearResult(rs)
```

14.4 Clean up

```
# dbRemoveTable(con, "cars")
dbRemoveTable(con, "mtcars")
# dbRemoveTable(con, "cust_movies")

# diconnect from the db
dbDisconnect(con)

result <- system2("docker", "stop sql-pet", stdout = TRUE, stderr = TRUE)
result
```

```
## [1] "sql-pet"
```

Chapter 15

Writing to the DBMS (73)

Start up the `docker-pet` container

```
result <- system2("docker", "start sql-pet", stdout = TRUE, stderr = TRUE)
result
```

```
## [1] "sql-pet"
```

now connect to the database with R

```
con <- wait_for_postgres(user = Sys.getenv("DEFAULT_POSTGRES_USER_NAME"),
                         password = Sys.getenv("DEFAULT_POSTGRES_PASSWORD"),
                         dbname = "dvdrental",
                         seconds_to_test = 10)
```

15.1 create a new table

This is an example from the DBI help file

```
dbWriteTable(con, "cars", head(cars, 3)) # "cars" is a built-in dataset, not to be confused with mtcars
dbReadTable(con, "cars")   # there are 3 rows
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
```

15.2 Modify an existing table

```
dbExecute(
  con,
  "INSERT INTO cars (speed, dist) VALUES (1, 1), (2, 2), (3, 3)"
)
```

```
## [1] 3
```

```
dbReadTable(con, "cars")   # there are now 6 rows
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     1    1
## 5     2    2
## 6     3    3
```

```
# Pass values using the param argument:
dbExecute(
  con,
  "INSERT INTO cars (speed, dist) VALUES ($1, $2)",
  param = list(4:7, 5:8)
)
```

```
## [1] 4
```

```
dbReadTable(con, "cars")   # there are now 10 rows
```

```
##    speed dist
## 1      4    2
## 2      4   10
## 3      7    4
## 4      1    1
## 5      2    2
## 6      3    3
## 7      4    5
## 8      5    6
## 9      6    7
## 10     7    8
```

15.3 Clean up

```
dbRemoveTable(con, "cars")

# diconnect from the db
dbDisconnect(con)

result <- system2("docker", "stop sql-pet", stdout = TRUE, stderr = TRUE)
result
```

```
## [1] "sql-pet"
```

Chapter 16

Other resources

16.1 Editing this book

- Here are instructions for editing this tutorial

16.2 Docker alternatives

- Choosing between Docker and Vagrant

16.3 Docker and R

- Noam Ross' talk on Docker for the UseR and his Slides give a lot of context and tips.
- Good Docker tutorials
  - An introductory Docker tutorial
  - A Docker curriculum
- Scott Came's materials about Docker and R on his website and at the 2018 UseR Conference focus on **R inside Docker**.
- It's worth studying the ROpensci Docker tutorial

16.4 Documentation Docker and Postgres

- The Postgres image documentation
- Dockerize PostgreSQL
- Postgres & Docker documentation
- Usage examples of Postgres with Docker

16.5 More Resources

- David Severski describes some key elements of connecting to databases with R for MacOS users
- This tutorial picks up ideas and tips from Ed Borasky's Data Science pet containers, which creates a framework based on that Hack Oregon example and explains why this repo is named pet-sql.

Chapter 17

Mapping your local environment (92)

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------------------------------------
## v ggplot2 3.0.0     v purrr   0.2.5
## v tibble  1.4.2     v dplyr   0.7.6
## v tidyr   0.8.1     v stringr 1.3.1
## v readr   1.1.1     v forcats 0.3.0
## -- Conflicts -----------------------------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(DBI)
library(RPostgres)
library(glue)
```

```
##
## Attaching package: 'glue'
## The following object is masked from 'package:dplyr':
##
##     collapse
```

```
library(knitr)
```

17.1 Environment Tools Used in this Chapter

Note that `tidyverse`, `DBI`, `RPostgres`, `glue`, and `knitr` are loaded. Also, we've sourced the `[db-login-batch-code.R]('r-database-docker/book-src/db-login-batch-code.R')` file which is used to log in to PostgreSQL.

library(rstudioapi)

The following code block defines Tool and versions for the graph that follows. The information order corresponds to the order shown in the graph.

```
library(DiagrammeR)

## OS information
os_lbl <- .Platform$OS.type
```

```r
os_ver <- 0
if (os_lbl == 'windows') {
  os_ver <- system2('cmd',stdout = TRUE) %>%
    grep(x = .,pattern = 'Microsoft Windows \\[',value = TRUE) %>%
    gsub(x = .,pattern = "^Microsoft.+Version |\\]", replace = '')
}

if (os_lbl == 'unix' || os_lbl == 'Linux' || os_lbl == 'Mac') {
  os_ver <- system2('uname', '-r', stdout = TRUE)
}

## Command line interface into Docker Apps
## CLI/system2
cli <- array(dim = 3)
cli[1] <- "docker [OPTIONS] COMMAND ARGUMENTS\n\nsystem2(docker,[OPTIONS,]\n, COMMAND,ARGUMENTS)"
cli[2] <- 'docker exec -it sql-pet bash\n\nsystem2(docker,exec -it sql-pet bash)'
cli[3] <- 'docker exec -ti sql-pet psql -a \n-p 5432 -d dvdrental -U postgres\n\nsystem2(docker,exec -t

# R Information
r_lbl       <- names(R.Version())[1:7]
r_ver       <- R.Version()[1:7]

# RStudio Information
rstudio_lbl <- c('RStudio version','Current program mode')
rstudio_ver <- c(as.character(rstudioapi::versionInfo()$version),rstudioapi::versionInfo()$mode)

# Docker Information
docker_lbl <- c('client version','server version')
docker_ver <- system2("docker", "version", stdout = TRUE) %>%
    grep(x = ., pattern = 'Version',value = TRUE) %>%
    gsub(x = ., pattern = ' +Version: +', replacement = '')

# Linux Information
linux_lbl <- 'Linux Version'
linux_ver <- system2('docker', 'exec -i sql-pet /bin/uname -r', stdout = TRUE)

# Postgres Information
con <- wait_for_postgres(user = Sys.getenv("DEFAULT_POSTGRES_USER_NAME"),
                         password = Sys.getenv("DEFAULT_POSTGRES_PASSWORD"),
                         dbname = "dvdrental",
                         seconds_to_test = 10)

postgres_ver <- dbGetQuery(con,"select version()") %>%
  gsub(x = ., pattern = '\\(.*$', replacement = '')
```

The following code block uses the data generated from the previous code block as input to the subgraphs, the ones outlined in red. The application nodes are the parents of the subgraphs and are not outlined in reds. The `Environment` application node represents the machine you are running the tutorial on and hosts the sub-applications.

Note that the '@@' variables are populated at the end of the `Environment` definition following the `## @@1 - @@5` source data comment.

```r
grViz("
digraph Envgraph {
```

```
# graph, node, and edge definitions
graph [compound = true, nodesep = .5, ranksep = .25,
       color = red]

node [fontname = Helvetica, fontcolor = darkslategray,
      shape = rectangle, fixedsize = true, width = 1,
      color = darkslategray]

edge [color = grey, arrowhead = none, arrowtail = none]

# subgraph for Environment information
subgraph cluster1 {
  node [fixedsize = true, width = 3]
  '@@1-1'
}

# subgraph for R information
subgraph cluster2 {
  node [fixedsize = true, width = 3]
  '@@2-1' -> '@@2-2' -> '@@2-3' -> '@@2-4'
  '@@2-4' -> '@@2-5' -> '@@2-6' -> '@@2-7'
}

# subgraph for RStudio information
subgraph cluster3 {
  node [fixedsize = true, width = 3]
  '@@3-1' -> '@@3-2'
}

# subgraph for Docker information
subgraph cluster4 {
  node [fixedsize = true, width = 3]
  '@@4-1' -> '@@4-2'
}

# subgraph for Docker-Linux information
subgraph cluster5 {
  node [fixedsize = true, width = 3]
  '@@5-1'
}

# subgraph for Docker-Postgres information
subgraph cluster6 {
  node [fixedsize = true, width = 3]
  '@@6-1'
}

# subgraph for Docker-Postgres information
subgraph cluster7 {
  node [fixedsize = true, height = 1.25, width = 4.0]
  '@@7-1' -> '@@7-2' -> '@@7-3'
}
```

```
  CLI [label='CLI\nRStudio system2',height = .75,width=3.0, color = 'blue' ]
  Environment             [label = 'Linux,Mac,Windows',width = 2.5]
  Environment -> R
  Environment -> RStudio
  Environment -> Docker

  Environment -> '@@1'    [lhead = cluster1] # Environment Information
  R           -> '@@2-1'  [lhead = cluster2] # R Information
  RStudio     -> '@@3'    [lhead = cluster3] # RStudio Information
  Docker      -> '@@4'    [lhead = cluster4] # Docker Information
  Docker      -> '@@5'    [lhead = cluster5] # Docker-Linux Information
  Docker      -> '@@6'    [lhead = cluster6] # Docker-Postgres Information

  '@@1' -> CLI
  CLI         -> '@@7'    [lhead = cluster7] # CLI
  '@@7-2'     -> '@@5'
  '@@7-3'     -> '@@6'
}
[1]: paste0(os_lbl,     ':\\n', os_ver)
[2]: paste0(r_lbl,      ':\\n', r_ver)
[3]: paste0(rstudio_lbl,':\\n', rstudio_ver)
[4]: paste0(docker_lbl, ':\\n', docker_ver)
[5]: paste0(linux_lbl,  ':\\n', linux_ver)
[6]: paste0('PostgreSQL:\\n', postgres_ver)
[7]: cli
")
```

One sub-application not shown above is your local console/terminal/cli application. In the tutorial, fully constructed docker commands are printed out and then executed. If for some reason the executed docker command fails, one can copy and paste it into your local terminal window to see additional error information. Failures seem more prevalent in the Windows environment.

17.2 Communicating with Docker Applications

In this tutorial, the two main ways to interface with the applications in the Docker container are through the CLI or the RStudio `system2` command. The blue box in the diagram above represents these two interfaces.