Ex5

Wassnaa Al-Mawee

2/07/2024

cran\_mirror <- "https://cran.rstudio.com/"  
options(repos = c(CRAN = cran\_mirror))  
install.packages("tidyr")

##   
## The downloaded binary packages are in  
## /var/folders/q\_/sg2ykfgs57z4jr3mr195l43c0000gn/T//RtmpOLxcEU/downloaded\_packages

## Accessing a relational database

* Install and load the dplyr, DBI, and RSQLite packages for accessing databases

install.packages(c("dplyr", "DBI", "RSQLite"))

##   
## The downloaded binary packages are in  
## /var/folders/q\_/sg2ykfgs57z4jr3mr195l43c0000gn/T//RtmpOLxcEU/downloaded\_packages

library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(DBI)  
library(RSQLite)

* Create a connection to the Chinook\_Sqlite.sqlite file.

con <- dbConnect(RSQLite::SQLite(), "/Users/harikrishnareddy/Desktop/R/WassnaaAl-Mawee/Ex5/Chinook\_Sqlite.sqlite")  
con

## <SQLiteConnection>  
## Path: /Users/harikrishnareddy/Desktop/R/WassnaaAl-Mawee/Ex5/Chinook\_Sqlite.sqlite  
## Extensions: TRUE

Use the dbListTables() function (passing in the connection) to get a list of tables in the database.

tables <- dbListTables(con)  
print(tables)

## [1] "Album" "Artist" "Customer" "Employee"   
## [5] "Genre" "Invoice" "InvoiceLine" "MediaType"   
## [9] "Playlist" "PlaylistTrack" "Track"

* Use the tbl()function to create a reference to the table of music genres. Print out the the table to confirm that you’ve accessed it.

genres <- tbl(con, "Genre")  
genres

## # Source: table<Genre> [?? x 2]  
## # Database: sqlite 3.45.0 [/Users/harikrishnareddy/Desktop/R/WassnaaAl-Mawee/Ex5/Chinook\_Sqlite.sqlite]  
## GenreId Name   
## <int> <chr>   
## 1 1 Rock   
## 2 2 Jazz   
## 3 3 Metal   
## 4 4 Alternative & Punk  
## 5 5 Rock And Roll   
## 6 6 Blues   
## 7 7 Latin   
## 8 8 Reggae   
## 9 9 Pop   
## 10 10 Soundtrack   
## # ℹ more rows

* Try to use View() to see the contents of the table. What happened?

head(genres)

## # Source: SQL [6 x 2]  
## # Database: sqlite 3.45.0 [/Users/harikrishnareddy/Desktop/R/WassnaaAl-Mawee/Ex5/Chinook\_Sqlite.sqlite]  
## GenreId Name   
## <int> <chr>   
## 1 1 Rock   
## 2 2 Jazz   
## 3 3 Metal   
## 4 4 Alternative & Punk  
## 5 5 Rock And Roll   
## 6 6 Blues

* Use the collect() function to actually load the genre table into memory as a data frame. View that data frame.

genres\_df <- collect(genres)  
genres\_df

## # A tibble: 25 × 2  
## GenreId Name   
## <int> <chr>   
## 1 1 Rock   
## 2 2 Jazz   
## 3 3 Metal   
## 4 4 Alternative & Punk  
## 5 5 Rock And Roll   
## 6 6 Blues   
## 7 7 Latin   
## 8 8 Reggae   
## 9 9 Pop   
## 10 10 Soundtrack   
## # ℹ 15 more rows

* Use dplyr’s count() function to see how many rows are in the genre table

genres\_count <- count(genres)  
genres\_count

## # Source: SQL [1 x 1]  
## # Database: sqlite 3.45.0 [/Users/harikrishnareddy/Desktop/R/WassnaaAl-Mawee/Ex5/Chinook\_Sqlite.sqlite]  
## n  
## <int>  
## 1 25

* Use the tbl() function to create a reference the table with track data.Print out the the table to confirm that you’ve accessed it.

tracks <- tbl(con, "Track")  
tracks

## # Source: table<Track> [?? x 9]  
## # Database: sqlite 3.45.0 [/Users/harikrishnareddy/Desktop/R/WassnaaAl-Mawee/Ex5/Chinook\_Sqlite.sqlite]  
## TrackId Name AlbumId MediaTypeId GenreId Composer Milliseconds Bytes  
## <int> <chr> <int> <int> <int> <chr> <int> <int>  
## 1 1 For Those A… 1 1 1 Angus Y… 343719 1.12e7  
## 2 2 Balls to th… 2 2 1 <NA> 342562 5.51e6  
## 3 3 Fast As a S… 3 2 1 F. Balt… 230619 3.99e6  
## 4 4 Restless an… 3 2 1 F. Balt… 252051 4.33e6  
## 5 5 Princess of… 3 2 1 Deaffy … 375418 6.29e6  
## 6 6 Put The Fin… 1 1 1 Angus Y… 205662 6.71e6  
## 7 7 Let's Get I… 1 1 1 Angus Y… 233926 7.64e6  
## 8 8 Inject The … 1 1 1 Angus Y… 210834 6.85e6  
## 9 9 Snowballed 1 1 1 Angus Y… 203102 6.60e6  
## 10 10 Evil Walks 1 1 1 Angus Y… 263497 8.61e6  
## # ℹ more rows  
## # ℹ 1 more variable: UnitPrice <dbl>

* Use dplyr functions to query for a list of artists in descending order by popularity in the database (e.g., the artist with the most tracks at the top)
* Start by filting for rows that have an artist listed (use is.na()), then group rows by the artist and count them. Finally, arrange the results.
* Use pipes to do this all as one statement without collecting the data into memory!

artist\_popularity <- tracks %>%  
 filter(!is.na(Composer)) %>%  
 group\_by(Composer) %>%  
 summarize(Count = n()) %>%  
 arrange(desc(Count))  
View(artist\_popularity)

* Use dplyr functions to query for the most popular *genre* in the library. You will need to count the number of occurrences of each genre, and join the two tables together in order to also access the genre name. Collect the resulting data into memory in order to access the specific row of interest

most\_popular\_genre <- genres %>%  
 left\_join(tracks, by = c("GenreId" = "GenreId")) %>%  
 count(GenreId, name = "GenreCount") %>%  
 arrange(desc(GenreCount)) %>%  
 collect() %>%  
 slice(1)  
most\_popular\_genre

## # A tibble: 1 × 2  
## GenreId GenreCount  
## <int> <int>  
## 1 1 1297

* Query for a list of the most popular artist for each genre in the library (a “representative” artist for each). Consider using multiple grouping operations. Note that you can only filter for a max() value if you’ve collected the data into memory.

# Remove the grouping structure before the join  
artist\_popularity <- tracks %>%  
 filter(is.na(Composer)) %>%  
 group\_by(GenreId, Composer) %>%  
 summarise(Count = n(), .groups = "drop") %>%  
 arrange(desc(Count)) %>%  
 group\_by(GenreId) %>%  
 slice\_max(Count) %>%  
 ungroup()  
  
genre\_artist\_popularity <- left\_join(artist\_popularity, genres, by = c("GenreId"))  
  
print(artist\_popularity)

## Warning: ORDER BY is ignored in subqueries without LIMIT  
## ℹ Do you need to move arrange() later in the pipeline or use window\_order() instead?

## # Source: SQL [?? x 3]  
## # Database: sqlite 3.45.0 [/Users/harikrishnareddy/Desktop/R/WassnaaAl-Mawee/Ex5/Chinook\_Sqlite.sqlite]  
## # Ordered by: desc(Count)  
## GenreId Composer Count  
## <int> <chr> <int>  
## 1 1 <NA> 168  
## 2 2 <NA> 51  
## 3 3 <NA> 44  
## 4 4 <NA> 31  
## 5 7 <NA> 309  
## 6 8 <NA> 27  
## 7 9 <NA> 26  
## 8 10 <NA> 16  
## 9 11 <NA> 15  
## 10 13 <NA> 3  
## # ℹ more rows

* Remember to disconnect from the database once you are done with it!

dbDisconnect(con)

# Working with data APIs

In this exercise, you’ll practice working with a data API: specifically, the [New York Times API](https://developer.nytimes.com/) for movie reviews. To learn more about the API, see the [developer console](https://developer.nytimes.com/movie_reviews_v2.json).

## Getting an API Key

You will need to register with the NYT site at [**https://developer.nytimes.com/signup**](https://developer.nytimes.com/signup) in order to get an API Key. Fill out the form (you can use fake information and a single-use email if you wish). Under “API” select **“Movie Reviews API”** (and note other APIs available for future projects)! The API key should be emailed to you once you sign up.

1. In order to utilize this key in your script, create a \*\*separate\* RScript called apikey.R inside this directory. Assign the key that was emailed to you to a variable:

nty\_apikey <- "dFwYLuZvyrAplw05W6VSBEbKQsJ3XKxX"

1. Use source() to load your API key variable from the apikey.R file you made. Make sure you’ve set your working directory!

rm(list = ls())  
suppressWarnings(source("apikey.R"))

1. Create a variable movie\_name that is the name of a movie of your choice.

setwd("/Users/harikrishnareddy/Desktop/R/WassnaaAl-Mawee/Ex5/")  
Article\_name <- "R research"

1. Construct an HTTP request to search for reviews for the given movie.

* The base URI is https://api.nytimes.com/svc/movies/v2/
* The resource is reviews/search.json
* See the interactive console for parameter details: <https://developer.nytimes.com/movie_reviews_v2.json>
* You should use YOUR api key (as the api-key parameter) and your movie\_name variable as the search query!

install.packages("httr")

##   
## The downloaded binary packages are in  
## /var/folders/q\_/sg2ykfgs57z4jr3mr195l43c0000gn/T//RtmpOLxcEU/downloaded\_packages

library("httr")  
url <- "https://api.nytimes.com/svc/search/v2/articlesearch.json"  
query\_params <- list(  
 q = Article\_name,   
 "api-key" = nty\_apikey   
)  
response <- GET(url, query = query\_params)

1. Send the HTTP Request to download the data. Then, extract the content and convert it from JSON

install.packages("jsonlite")

##   
## The downloaded binary packages are in  
## /var/folders/q\_/sg2ykfgs57z4jr3mr195l43c0000gn/T//RtmpOLxcEU/downloaded\_packages

library(httr)  
library(jsonlite)  
response\_text <- content(response, type = "text",encoding = "UTF-8")  
response\_data <- fromJSON(response\_text)

1. What kind of data structure did this produce? A data frame? A list?

class(response\_data)

## [1] "list"

1. Manually inspect the returned data and identify the content of interest (which are the movie reviews). Use functions such as names(), str(), etc.

names(response\_data)

## [1] "status" "copyright" "response"

1. Flatten the movie reviews content into a data structure called reviews

reviews <- flatten(response\_data$response$docs)

1. From the most recent review, store the headline, short summary, and link to the full article, each in their own variables

headline <- reviews[[18]][[1]]  
short\_summary <- reviews[[1]][[1]]  
full\_article <- reviews[[2]][[1]]

1. Create a list of the three pieces of information from above. Print out the list.

review\_info <- list(headline = headline, summary = summary, full\_article\_link = full\_article)  
print(review\_info)

## $headline  
## [1] "nyt://article/840fec7b-debd-56e0-82c6-3884925c1658"  
##   
## $summary  
## function (object, ...)   
## UseMethod("summary")  
## <bytecode: 0x120e8b5a8>  
## <environment: namespace:base>  
##   
## $full\_article\_link  
## [1] "https://www.nytimes.com/2023/12/06/business/dealbook/2024-election-media.html"