Database and SQL

02-11-2020

```r
library(tidyverse)
library(DBI)
```

What is a database? It is what google says

> a structured set of data held in a computer, especially one that is accessible in various ways.

A relational database is a type of database that stores and provides access to data points that are related to one another. Relation databases are administrated by a Relational Database Management System (RDMBS). The data in RDBMS is stored in database objects called tables. A table is a collection of related data entries and it consists of columns and rows.

There are many RDMBS. - MySQL (owned by Oracle) - PostgreSQL (open source) - SQL Server (microsoft) - SQLite (open source, single file)

What is SQL? Structured Query Language (or SQL) is a standard language for accessing and manipulating relational databaes. However, each RDMBS may have their own extension of the SQL language and their implmentation may vary too.

Connect to a sqlite databse

We are going to use a popular database called Sakila https://www.jooq.org/sakila.

The Sakila database is a nicely normalised schema modelling a DVD rental store, featuring things like films, actors, film-actor relationships, and a central inventory table that connects films, stores, and rentals.

You could either download the file from lectures repo or from the dropbox link https://www.dropbox.com/s/7hmifspoj4jzsk1/sakila.sqlite?dl=0

The file format is `.sqlite` which is one of the very common relational database formats, espeically for simple problems.

```r
sakila <- dbConnect(RSQLite::SQLite(), dbname = "sakila.sqlite")
sakila %>% dbListTables()
```

```
##  [1] "actor"                  "address"                "category"
##  [4] "city"                   "country"                "customer"
##  [7] "customer_list"          "film"                   "film_actor"
## [10] "film_category"          "film_list"              "film_text"
## [13] "inventory"              "language"               "payment"
## [16] "rental"                 "sales_by_film_category" "sales_by_store"
## [19] "sqlite_sequence"        "staff"                  "staff_list"
## [22] "store"
```

How not to use SQL?

dplyr provides an excellent interface for users without any SQL background to query databases.

```
# number of rental transactions
sakila %>%
  tbl("rental") %>%
  count() %>%
  collect()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1 16044
```

sakila %>% tbl("rental") creates a virtual table rather loading the whole table into memory.

```
sakila %>%
  tbl("rental") %>%
  class()
```

```
## [1] "tbl_SQLiteConnection" "tbl_dbi"              "tbl_sql"
## [4] "tbl_lazy"             "tbl"
```

```
sakila %>%
  tbl("rental") %>%
  colnames()
```

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

Sakila queris

https://datamastery.gitlab.io/exercises/sakila-queries.html

- Which actors have the first name Scarlett?

```
sakila %>%
  tbl("actor") %>%
  filter(str_to_lower(first_name) == str_to_lower("Scarlett")) %>%
  collect()
```

```
## # A tibble: 2 x 4
##   actor_id first_name last_name last_update
##      <int> <chr>      <chr>     <chr>
## 1       81 SCARLETT   DAMON     2019-04-11 18:11:48
## 2      124 SCARLETT   BENING    2019-04-11 18:11:48
```

Suppose we want to make the result a bit more beautiful.

```
sakila %>%
  tbl("actor") %>%
  filter(str_to_lower(first_name) == str_to_lower("Scarlett")) %>%
  collect() %>%
  mutate(first_name = str_to_title(first_name), last_name = str_to_title(last_name))
```

```
## # A tibble: 2 x 4
##   actor_id first_name last_name last_update
##      <int> <chr>      <chr>     <chr>
## 1       81 Scarlett   Damon     2019-04-11 18:11:48
## 2      124 Scarlett   Bening    2019-04-11 18:11:48
```

- Which actors have the last name Johansson?

```
sakila %>%
  tbl("actor") %>%
  filter(str_to_lower(last_name) == "johansson") %>%
  collect()
```

```
## # A tibble: 3 x 4
##   actor_id first_name last_name last_update
##      <int> <chr>      <chr>     <chr>
## 1        8 MATTHEW    JOHANSSON 2019-04-11 18:11:48
## 2       64 RAY        JOHANSSON 2019-04-11 18:11:48
## 3      146 ALBERT     JOHANSSON 2019-04-11 18:11:48
```

- How many distinct actors last names are there?

```
sakila %>%
  tbl("actor") %>%
  summarize(n = n_distinct(last_name)) %>%
  collect()
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1   121
```

- Which last names are not repeated?

```
sakila %>%
  tbl("actor") %>%
  group_by(last_name) %>%
  count() %>%
  filter(n == 1) %>%
  collect()
```

```
## # A tibble: 66 x 2
## # Groups:   last_name [66]
##    last_name     n
##    <chr>     <int>
##  1 ASTAIRE       1
##  2 BACALL        1
##  3 BALE          1
##  4 BALL          1
##  5 BARRYMORE     1
##  6 BASINGER      1
##  7 BERGEN        1
##  8 BERGMAN       1
##  9 BIRCH         1
## 10 BLOOM         1
## # ... with 56 more rows
```

- Which last names appear more than once?

```
sakila %>%
  tbl("actor") %>%
  group_by(last_name) %>%
  count() %>%
  filter(n > 1) %>%
  collect()
```

```
## # A tibble: 55 x 2
## # Groups:   last_name [55]
##    last_name     n
##    <chr>     <int>
##  1 AKROYD        3
##  2 ALLEN         3
##  3 BAILEY        2
##  4 BENING        2
##  5 BERRY         3
##  6 BOLGER        2
##  7 BRODY         2
##  8 CAGE          2
##  9 CHASE         2
## 10 CRAWFORD      2
## # ... with 45 more rows
```

- Which actor has appeared in the most films?

```
sakila %>%
  tbl("film_actor") %>%
  group_by(actor_id) %>%
  count() %>%
  arrange(desc(n)) %>%
  head(1) %>%
  left_join(tbl(sakila, "actor"), by = 'actor_id') %>%
  collect()
```

```
## # A tibble: 1 x 5
## # Groups:   actor_id [1]
##   actor_id     n first_name last_name last_update
##      <int> <int> <chr>      <chr>     <chr>
## 1      107    42 GINA       DEGENERES 2019-04-11 18:11:48
```

- What is that average running time of all the films in the sakila DB?

```
sakila %>%
  tbl("film") %>%
  summarize(m = mean(length)) %>%
  collect()
```

```
## Warning: Missing values are always removed in SQL.
## Use `mean(x, na.rm = TRUE)` to silence this warning
## This warning is displayed only once per session.
```

```
## # A tibble: 1 x 1
##       m
##   <dbl>
## 1  115.
```

- What is the average running time of films by category?

```
sakila %>%
  tbl("film") %>%
  left_join(tbl(sakila, "film_category"), by = "film_id") %>%
  group_by(category_id) %>%
  summarize(mean_length = mean(length)) %>%
  left_join(tbl(sakila, "category"), by = "category_id") %>%
  select(name, mean_length)
```

```
## # Source:   lazy query [?? x 2]
## # Database: sqlite 3.30.1
## #   [/Users/Randy/Dropbox/Winter2020/STA141B/sta141b-lectures/02-11/sakila.sqlite]
##    name        mean_length
##    <chr>             <dbl>
##  1 Action             112.
##  2 Animation          111.
##  3 Children           110.
##  4 Classics           112.
##  5 Comedy             116.
##  6 Documentary        109.
##  7 Drama              121.
##  8 Family             115.
##  9 Foreign            122.
## 10 Games              128.
## # ... with more rows
```

- Is ‘Unforgiven Zoolander’ available for rent from Store 1?

```
uz <- sakila %>%
  tbl("film") %>%
  filter(str_to_lower(title) == str_to_lower('Unforgiven Zoolander')) %>%
  select(film_id)
all_inventories_of_store1 <- sakila %>%
  tbl("inventory") %>%
  filter(store_id == 1) %>%
  select(film_id, inventory_id, store_id)
not_yet_returned <- sakila %>%
  tbl("rental") %>%
  filter(is.na(return_date)) %>%
  select(inventory_id)
uz %>%
  left_join(all_inventories_of_store1) %>%
  anti_join(not_yet_returned) %>%
  count() %>%
  collect()
```

```
## Joining, by = "film_id"

## Joining, by = "inventory_id"

## # A tibble: 1 x 1
##       n
##   <int>
## 1     2
```

SQL

We just see some example queries of a relational database. Beind the scene, we are using a language called SQL. For example, in the last query, the SQL used is

```
uz %>%
  left_join(all_inventories_of_store1) %>%
  anti_join(not_yet_returned) %>%
  count() %>%
  show_query()
```

```
## Joining, by = "film_id"

## Joining, by = "inventory_id"

## <SQL>
## SELECT COUNT() AS `n`
## FROM (SELECT * FROM (SELECT `LHS`.`film_id` AS `film_id`, `RHS`.`inventory_id` AS `inventory_id`, `R
## FROM (SELECT `film_id`
## FROM `film`
## WHERE (LOWER(`title`) = LOWER('Unforgiven Zoolander'))) AS `LHS`
## LEFT JOIN (SELECT `film_id`, `inventory_id`, `store_id`
## FROM `inventory`
## WHERE (`store_id` = 1.0)) AS `RHS`
## ON (`LHS`.`film_id` = `RHS`.`film_id`)
## ) AS `LHS`
## WHERE NOT EXISTS (
##   SELECT 1 FROM (SELECT `inventory_id`
## FROM `rental`
## WHERE (((`return_date`) IS NULL))) AS `RHS`
##   WHERE (`LHS`.`inventory_id` = `RHS`.`inventory_id`)
## ))
```

Why learning SQL when there is dplyr?

- SQL is everywhere (used in python, php, etc..)
- Job interviews

First we will need to connect to the database,

In R,

```
sakila <- dbConnect(RSQLite::SQLite(), dbname = "sakila.sqlite")
sakila %>% dbGetQuery("SELECT COUNT() AS `n` FROM `rental`")
```

```
##       n
## 1 16044
```

We could also make SQL query by sql block. In here, we are using the connection `sakila`. The result will be printed directly.

```
SELECT COUNT() AS `n` FROM `rental`;
```

Table 1: 1 records

n
16044

We could store the output to `rental_count`

```
SELECT COUNT() AS `n` FROM `rental`;
```

```
rental_count
```

For comparsion, in Python, we use

```
import sqlite3
sakila = sqlite3.connect('sakila.sqlite.sqlite')
c = sakila.cursor()
c.execute("SELECT COUNT() AS `n` FROM `rental`")
c.fetchall()
```

It is not stricly neccsaary to use backticks to quote identifiers and use upper cased keywords.

```
select count() as n from rental;
```

Table 2: 1 records

n
16044

However, it is a good pratice to use backticks to quote identifiers because some databases, for example, postgresql would otherwise convert the upper case identifiers to lower case identifiers

SELECT

The SELECT statement is pretty much the `select()` function in `dplyr`.

```
SELECT `last_name` FROM `actor`;
```

Table 3: Displaying records 1 - 10

last_name
AKROYD
AKROYD
AKROYD
ALLEN
ALLEN
ALLEN
ASTAIRE
BACALL
BAILEY
BAILEY

```
SELECT LOWER(`last_name`) as `family_name` FROM `actor`;
```

Table 4: Displaying records 1 - 10

family_name
akroyd
akroyd
akroyd
allen
allen
allen
astaire
bacall
bailey
bailey

```
SELECT * FROM `actor`;
```

Table 5: Displaying records 1 - 10

actor_id	first_name	last_name	last_update
1	PENELOPE	GUINESS	2019-04-11 18:11:48
2	NICK	WAHLBERG	2019-04-11 18:11:48
3	ED	CHASE	2019-04-11 18:11:48
4	JENNIFER	DAVIS	2019-04-11 18:11:48

actor_id	first_name	last_name	last_update
5	JOHNNY	LOLLOBRIGIDA	2019-04-11 18:11:48
6	BETTE	NICHOLSON	2019-04-11 18:11:48
7	GRACE	MOSTEL	2019-04-11 18:11:48
8	MATTHEW	JOHANSSON	2019-04-11 18:11:48
9	JOE	SWANK	2019-04-11 18:11:48
10	CHRISTIAN	GABLE	2019-04-11 18:11:48

```
SELECT `rental_id`, `last_update` FROM `rental`;
```

Table 6: Displaying records 1 - 10

rental_id	last_update
1	2019-04-11 18:11:49
2	2019-04-11 18:11:49
3	2019-04-11 18:11:49
4	2019-04-11 18:11:49
5	2019-04-11 18:11:49
6	2019-04-11 18:11:49
7	2019-04-11 18:11:49
8	2019-04-11 18:11:49
9	2019-04-11 18:11:49
10	2019-04-11 18:11:49

ORDER BY Clause

It is equivalent to `arrange()` in ‘dplyr“

```
SELECT * FROM `actor` ORDER BY `LAST_NAME`;
```

Table 7: Displaying records 1 - 10

actor_id	first_name	last_name	last_update
58	CHRISTIAN	AKROYD	2019-04-11 18:11:48
92	KIRSTEN	AKROYD	2019-04-11 18:11:48
182	DEBBIE	AKROYD	2019-04-11 18:11:48
118	CUBA	ALLEN	2019-04-11 18:11:48
145	KIM	ALLEN	2019-04-11 18:11:48
194	MERYL	ALLEN	2019-04-11 18:11:48
76	ANGELINA	ASTAIRE	2019-04-11 18:11:48
112	RUSSELL	BACALL	2019-04-11 18:11:48
67	JESSICA	BAILEY	2019-04-11 18:11:48
190	AUDREY	BAILEY	2019-04-11 18:11:48

DISTINCT

DISTINCT operator to remove duplicates from a result set. It is equivalent to `distinct()` function in `dplyr`.

```
SELECT DISTINCT `last_name` FROM `actor`;
```

Table 8: Displaying records 1 - 10

last_name
AKROYD
ALLEN
ASTAIRE
BACALL
BAILEY
BALE
BALL
BARRYMORE
BASINGER
BENING

```
sakila %>%
  tbl("actor") %>%
  distinct(last_name)
```

```
## # Source:   lazy query [?? x 1]
## # Database: sqlite 3.30.1
## #   [/Users/Randy/Dropbox/Winter2020/STA141B/sta141b-lectures/02-11/sakila.sqlite]
##    last_name
##    <chr>
##  1 AKROYD
##  2 ALLEN
##  3 ASTAIRE
##  4 BACALL
##  5 BAILEY
##  6 BALE
##  7 BALL
##  8 BARRYMORE
##  9 BASINGER
## 10 BENING
## # ... with more rows
```

LIMIT

```
SELECT * FROM `actor` LIMIT 2;
```

Table 9: 2 records

actor_id	first_name	last_name	last_update
1	PENELOPE	GUINESS	2019-04-11 18:11:48
2	NICK	WAHLBERG	2019-04-11 18:11:48

```r
sakila %>%
  tbl("actor") %>%
  head(2)
```

```
## # Source:   lazy query [?? x 4]
## # Database: sqlite 3.30.1
## #   [/Users/Randy/Dropbox/Winter2020/STA141B/sta141b-lectures/02-11/sakila.sqlite]
##   actor_id first_name last_name last_update
##      <int> <chr>      <chr>     <chr>
## 1        1 PENELOPE   GUINESS   2019-04-11 18:11:48
## 2        2 NICK       WAHLBERG  2019-04-11 18:11:48
```

WHERE

It is equivalent to `filter()` in `dplyr`.

Normally, strings are quoted in single quotes.

```sql
SELECT * FROM `film` WHERE `rating` == 'PG' AND `length` > 90;
```

film_id	title	description
6	AGENT TRUMAN	A Intrepid Panorama of a Robot And a Boy who must Escape a Sumo Wrestler i
12	ALASKA PHANTOM	A Fanciful Saga of a Hunter And a Pastry Chef who must Vanquish a Boy in Au
13	ALI FOREVER	A Action-Packed Drama of a Dentist And a Crocodile who must Battle a Femini
19	AMADEUS HOLY	A Emotional Display of a Pioneer And a Technical Writer who must Battle a Ma
37	ARIZONA BANG	A Brilliant Panorama of a Mad Scientist And a Mad Cow who must Meet a Pion
41	ARSENIC INDEPENDENCE	A Fanciful Documentary of a Mad Cow And a Womanizer who must Find a Den
65	BEHAVIOR RUNAWAY	A Unbelieveable Drama of a Student And a Husband who must Outrace a Sumo
72	BILL OTHERS	A Stunning Saga of a Mad Scientist And a Forensic Psychologist who must Chal
74	BIRCH ANTITRUST	A Fanciful Panorama of a Husband And a Pioneer who must Outgun a Dog in A
84	BOILED DARES	A Awe-Inspiring Story of a Waitress And a Dog who must Discover a Dentist in

```r
sakila %>%
  tbl("film") %>%
  filter(rating == "PG" && length > 90)
```

```
## # Source:   lazy query [?? x 13]
## # Database: sqlite 3.30.1
## #   [/Users/Randy/Dropbox/Winter2020/STA141B/sta141b-lectures/02-11/sakila.sqlite]
##    film_id title description release_year language_id original_langua~
##      <int> <chr> <chr>       <chr>              <int>            <int>
##  1       6 AGEN~ A Intrepid~ 2006                   1               NA
##  2      12 ALAS~ A Fanciful~ 2006                   1               NA
##  3      13 ALI ~ A Action-P~ 2006                   1               NA
##  4      19 AMAD~ A Emotiona~ 2006                   1               NA
##  5      37 ARIZ~ A Brillian~ 2006                   1               NA
##  6      41 ARSE~ A Fanciful~ 2006                   1               NA
##  7      65 BEHA~ A Unbeliev~ 2006                   1               NA
##  8      72 BILL~ A Stunning~ 2006                    1               NA
##  9      74 BIRC~ A Fanciful~ 2006                    1               NA
## 10      84 BOIL~ A Awe-Insp~ 2006                    1               NA
## # ... with more rows, and 7 more variables: rental_duration <int>,
## #   rental_rate <dbl>, length <int>, replacement_cost <dbl>, rating <chr>,
## #   special_features <chr>, last_update <chr>
```

The IN and LIKE operators

```
-- sqlite LIKE operator is case insensitive in default
-- you could make it case sensitive by using
PRAGMA case_sensitive_like = false;
```

```
SELECT * FROM `film` WHERE `rating` IN ('PG', 'PG-13') AND `title` LIKE "%car%";
```

film_id	title	description
121	CAROL TEXAS	A Astounding Character Study of a Composer And a Student who must Overcome a C
122	CARRIE BUNCH	A Amazing Epistle of a Student And a Astronaut who must Discover a Frisbee in The
644	OSCAR GOLD	A Insightful Tale of a Database Administrator And a Dog who must Face a Madman
768	SCARFACE BANG	A Emotional Yarn of a Teacher And a Girl who must Find a Teacher in A Baloon Fac
857	STRICTLY SCARFACE	A Touching Reflection of a Crocodile And a Dog who must Chase a Hunter in An Aba

CASE

Similar to `case_when()` in `dplyr`.

```
SELECT
  film_id,
  title,
  CASE
    WHEN length < 60 THEN 'short'
    WHEN length < 90 THEN 'mid'
    ELSE 'long'
  END length
FROM `film`;
```

Table 12: Displaying records 1 - 10

film_id	title	length
1	ACADEMY DINOSAUR	mid
2	ACE GOLDFINGER	short
3	ADAPTATION HOLES	short
4	AFFAIR PREJUDICE	long
5	AFRICAN EGG	long
6	AGENT TRUMAN	long
7	AIRPLANE SIERRA	mid
8	AIRPORT POLLOCK	short
9	ALABAMA DEVIL	long

film_id	title	length
10	ALADDIN CALENDAR	mid

JOIN operations

Reference

- SQL Tutorial https://www.sqltutorial.org/