# Querying a MySQL database

This lab was originally created by Profs Ben Baumer and Jordan Crouser. I've edited it slightly.

SQL is a longstanding database querying language. It is a loosely-implemented standard. We will be using MySQL.

To facilitate our connection to the MySQL database server, we will need to install the `RMySQL` package.

```
# do NOT install RMySQL if you are on the RStudio Server
install.packages("RMySQL")
```

**Goal**: by the end of this lab, you will be able to write basic `SELECT` queries in SQL and retrieve the results into R.

## Connecting to MySQL

The data we will be using is stored on a server in Bass Hall. It's called `scidb.smith.edu`. We can connect through the `dbConnect()` function provided by the `DBI` package (which is loaded automatically when you load `RMySQL`). You will also need the `RMySQL` package installed.

```
library(tidyverse)
library(RMySQL)
db <- dbConnect(
  MySQL(),
  host = "scidb.smith.edu",
  user = "sds192",
  password = "DSismfc@S",
  dbname = "imdb"
  )
knitr::opts_chunk$set(connection= 'db', max.print = 5)
```

This chunk of code will allow you to connect to `scidb`. Note that this creates a database connection object named `db`, which has the class `MySQLConnection`.

```r
class(db)
```

```
[1] "MySQLConnection"
attr(,"package")
[1] "RMySQL"
```

Also, we set the `connection` parameter for all future chunks in this R Markdown file. Note also that the `max.print` argument sets the maximum number of results printed by each query.

Each of the following chunks makes use of the SQL engine functionality in `knitr`. You may want to read about this. Each of the following chunks is an `sql` chunk – as opposed to an `r` chunk!

To retrieve the results from a query in R, use the `dbGetQuery()` function from the `DBI` package (which is automatically loaded when you load `RMySQL`). Its first argument is a database connection object, and the second argument is an SQL query as a character vector.

## Retrieving data

We want to be able to see which type of databases exist on the server. We can do this with SHOW DATABASES

```sql
SHOW DATABASES;
```

Table 1: Displaying records 1 - 5

| Database |
|---|
| airlines |
| citibike |
| fec |
| imdb |
| information_schema |

You don't actually need the ; at the end of the quiery above for MySQL, but for other SQL dialects you do so it doesn't hurt.

We want to use the imdb databse and we have to tell our db connection that.

```
USE imdb;
```

Its fine that there are 0 rows, this is just how we tell R which db we want to use.

Let's look at the tables available in imdb.

```
SHOW TABLES;
```

Table 2: Displaying records 1 - 5

| Tables_in_imdb |
|---|
| aka_name |
| aka_title |
| cast_info |
| char_name |
| comp_cast_type |

See the kind_type table? That one shows what type of movie it is.

This query returns the list of kinds of "movies" stored in the IMDB. We are selecting everything with * from the kind_type table.

```
SELECT * FROM kind_type;
```

Table 3: 7 records

| id | kind |
|---|---|
| 1 | movie |
| 2 | tv series |
| 3 | tv movie |
| 4 | video movie |
| 5 | tv mini series |
| 6 | video game |
| 7 | episode |

Of course, you will often want to store the result of your query as a data frame. This can be achieved by setting the `output.var` argument in the chunk. Here we retrieve the list of different types of information stored in the database, save it as a data frame in R, and show the first few rows.

```
## output.var="info_types" in this chunk

SELECT * FROM info_type;
```

We now have a `data.frame` called `info_types` in our environment.

```
# Note: this is an r chunk s we use R instead of SQL.

head(info_types)
```

```
  id         info
1  1     runtimes
2  2   color info
3  3       genres
4  4    languages
5  5 certificates
6  6    sound mix
```

That's all you need to know about how to get data from MySQL into R. The rest of this lab consists of practicing writing SQL queries. It may be useful to reference the full documentation for `SELECT` queries.

For example, let's say I wanted to find information on the wacky Bill Murray Movie Groundhog Day.

The titles are stored in the `title` field (i.e. column) in the `title` table. [Note: your professor is not responsible for naming these tables and fields!] Each row in the `title` table corresponds to a single movie, but of course, we need to restrict the rows we retrieve to only those where the `title` field equals `Groundhog Day`. The following query achieves this.

> Note: SQL does not require the `==` for testing equality, since you aren't ever changing the data.
>
> Note: You have to use `'` single quotes since you are working within a `"` double-quoted string.

In the chunk below we select every column from the title-table where the title-variable equals 'Groundhog Day'

```
SELECT *
FROM title
WHERE title = 'Groundhog Day';
```

Table 4: Displaying records 1 - 5

| id | title | imdb_index | kind_id | production_year | imdb_id | phonetic_code | episode_of_id | season_nr | episode_nr | series_years | md5sum |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 19605 | Groundhog Day | NA | 7 | 2014 | NA | G6532 | 19602 | 1 | 2 | NA | a5e203197e1aa883f7884eb89e92 |
| 27895 | Groundhog Day | NA | 7 | 2008 | NA | G6532 | 27822 | 1 | 48 | NA | 74ebd1bfceb83d4bdae480326d0 |
| 387076 | Groundhog Day | NA | 7 | 2011 | NA | G6532 | 387052 | 1 | 2 | NA | b8d621787a0ea75c76eff6b8a6a8 |
| 384111 | Groundhog Day | NA | 7 | 2016 | NA | G6532 | 384073 | 1 | 11 | NA | bcef9ee95ae2bb82eca4ae234108 |
| 337084 | Groundhog Day | NA | 7 | 2016 | NA | G6532 | 337080 | 1 | 5 | NA | fcaddb8be4ab7b9c5ca9f27bb325 |

That retrieved a lot of movies! Let's see if we can refine our query. First, movies (as opposed to TV episodes, etc.) have the `kind_id` value of `1`.

```sql
SELECT *
FROM title
WHERE title = 'Groundhog Day'
AND kind_id = 1;
```

Table 5: 1 records

| id | title | imdb_index | kind_id | production_year | imdb_id | phonetic_code | episode_of_id | season_nr | episode_nr | series_years | md5sum |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3664274 | Groundhog Day | NA | 1 | 1993 | NA | G6532 | NA | NA | NA | NA | 2f0a563d0b0a1f57a19385de5a87 |

Now we have the result that I want.

Imagine that I didn't know the full title of the movie I could soften my query by searching for the phrase `Groundhog` within the title. We can do this using the `LIKE` function along with some wildcards (`%` in SQL).

```sql
SELECT *
FROM title
WHERE title LIKE '%Groundhog%'
AND kind_id = 1;
```

Table 6: 4 records

| id | title | imdb_index | kind_id | production_year | imdb_id | phonetic_code | episode_of_id | season_nr | episode_nr | series_years | md5sum |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3664274 | Groundhog Day | NA | 1 | 1993 | NA | G6532 | NA | NA | NA | NA | 2f0a563d0b0a1f57a19385de5a87 |
| 3664277 | Groundhogs | NA | 1 | 2015 | NA | G6532 | NA | NA | NA | NA | 7b71cb8ae79de1171a71f95d2e50 |
| 3664273 | Groundhog | NA | 1 | 2017 | NA | G6532 | NA | NA | NA | NA | 5e7183dbeb6c28fb6445c4013b2l |
| 3664276 | Groundhogs Day | NA | 1 | 2016 | NA | G6532 | NA | NA | NA | NA | c45dd6456b9787e5f71144d6c3a |

Pretend I'm still not sure which of the above four movies is the real Groundhog Day movie I'm interested in, but I'm sure its the first movie that came out. I could put them in order with the code below.

```
SELECT *
FROM title
WHERE title LIKE '%Groundhog%'
  AND kind_id = 1
ORDER BY production_year;
```

Table 7: 4 records

| id | title | imdb_index | kind_id | production_year | imdb_id | phonetic_code | episode_of_id | season_nr | episode_nr | series_years | md5sum |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3664274 | Groundhog Day | NA | 1 | 1993 | NA | G6532 | NA | NA | NA | NA | 2f0a563d0b0a1f57a19385de5a87 |
| 3664277 | Groundhogs | NA | 1 | 2015 | NA | G6532 | NA | NA | NA | NA | 7b71cb8ae79de1171a71f95d2e50 |
| 3664276 | Groundhogs Day | NA | 1 | 2016 | NA | G6532 | NA | NA | NA | NA | c45dd6456b9787e5f71144d6c3a |
| 3664273 | Groundhog | NA | 1 | 2017 | NA | G6532 | NA | NA | NA | NA | 5e7183dbeb6c28fb6445c4013b2l |

Finally I can select just the three columns I'm interested in. Also notice that I am renaming the title table as t. So I select the columns t.title and t.production_year. This is called creating an alias. The convention is table.variable. This will be useful when joining tables.

```
SELECT t.id, t.title, t.production_year
FROM title AS t
WHERE title LIKE '%Groundhog%'
  AND t.kind_id = 1
ORDER BY t.production_year;
```

Table 8: 4 records

| id | title | production_year |
|---|---|---|
| 3664274 | Groundhog Day | 1993 |
| 3664277 | Groundhogs | 2015 |
| 3664276 | Groundhog's Day | 2016 |
| 3664273 | Groundhog | 2017 |

Its the first Groundhog day that came out in 1993 with ID 3664274.

## Exercise:

Find the original Ghostbusters in the `title` table.

```
SELECT *
FROM title
WHERE title = 'Ghostbusters'
AND kind_id = 1;
```

Table 9: 3 records

| id | title | imdb_index | kind_id | production_year | imdb_id | phonetic_code | episode_of_id | season_nr | episode_nr | series_years | md5sum |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 3644554 | Ghostbusters | NA | 1 | 2016 | NA | G2312 | NA | NA | NA | NA | 9dc8686712d01fc31d597c8aa189 |
| 3644555 | Ghostbusters | NA | 1 | 1984 | NA | G2312 | NA | NA | NA | NA | 595c84db94e698e6002097cd7c2b |
| 3644556 | Ghostbusters | NA | 1 | NA | NA | G2312 | NA | NA | NA | NA | d5eb40dc8725d1041e0260e7baa |

Now lets consider the name table

```
DESCRIBE name
```

Table 10: Displaying records 1 - 5

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| id | int | NO | PRI | NA | auto_increment |
| name | text | NO | MUL | NA | |
| imdb_index | varchar(12) | YES | | NA | |
| imdb_id | int | YES | MUL | NA | |
| gender | varchar(1) | YES | | NA | |

## Exercise:

Find Andie MacDowell's `id` in the `name` table.

> Note: that names are listed last name first and seperated by a comma (eg Murray, Bill)

```
SELECT * FROM name
WHERE name = 'MacDowell, Andie';
```

Table 11: 1 records

| id | name | imdb_index | imdb_id | gender | name_pcode_cf | name_pcode_nf | surname_pcode | md5sum |
|---|---|---|---|---|---|---|---|---|
| 3479179 | MacDowell, Andie | NA | NA | f | M2345 | A5352 | M234 | b73def7bf61212e32f9c6bd3c39e1c00 |

### Joining tables

In the IMDB, the `title` table contains information about movies, the `name` table contains the names of people, the `char_name` table contains information about the names of characters, and the `cast_info` table contains information about which people played which roles in which movies. Linking the tables together is essential in order to extract information from the database.

Since we already know that the ID of *Groundhog Day* is `3664274`, we can use that to find all of the cast assignments.

```
SELECT *
FROM cast_info
WHERE movie_id = 3664274;
```

Table 12: Displaying records 1 - 5

| id | person_id | movie_id | person_role_id | note | nr_order | role_id |
|---|---|---|---|---|---|---|
| 118176 | 17392 | 3664274 | 376 | NA | 43 | 1 |
| 1764292 | 232622 | 3664274 | 352541 | (uncredited) | NA | 1 |
| 2746369 | 354870 | 3664274 | 191351 | NA | 13 | 1 |
| 3167452 | 400710 | 3664274 | 140101 | NA | 34 | 1 |
| 4584838 | 577860 | 3664274 | 98 | (uncredited) | NA | 1 |

Note that this returns a list of person-role pairs. person_id is unique for the actors and person_role_id is unique for the role played.

1. Find all the rows in `cast_info` that correspond to Andie MacDowell as an actress.

We first need to understand the columns that are in cast_info.

```
DESCRIBE cast_info
```

Table 13: Displaying records 1 - 5

| Field | Type | Null | Key | Default | Extra |
|---|---|---|---|---|---|
| id | int | NO | PRI | NA | auto_increment |
| person_id | int | NO | MUL | NA | |
| movie_id | int | NO | MUL | NA | |
| person_role_id | int | YES | MUL | NA | |
| note | text | YES | | NA | |

The id from the name table should match the person_id from the cast_info table. So we can join on that. Notice that the names of the tables are being shorted again as n for name and ci for cast_info.

```
##SAMPLE SOLUTION

SELECT n.name, ci.role_id
FROM cast_info AS ci
JOIN name AS n ON n.id = ci.person_id
WHERE ci.person_id = 3479179;
```

Table 14: Displaying records 1 - 5

| name | role_id |
|---|---|
| MacDowell, Andie | 2 |
| MacDowell, Andie | 2 |
| MacDowell, Andie | 2 |
| MacDowell, Andie | 2 |
| MacDowell, Andie | 2 |

## Find all the people who acted in Groundhog Day

Next, we can join the `cast_info` table on the `name` table to recover the names of the actors in Groundhog Day.

```
SELECT n.name, ci.role_id
FROM cast_info AS ci
JOIN name n ON n.id = ci.person_id
WHERE movie_id = 3664274;
```

Table 15: Displaying records 1 - 5

| name | role_id |
|---|---|
| Adler, Roger | 1 |
| Blakeslee, Douglas | 1 |
| Campbell, Ken Hudson | 1 |
| Chaiyabhat, Shaun | 1 |
| DeGuide, Tony | 1 |

Note how we have used table aliases to save some typing.

### Joining more than two tables

Add the names of the characters she played to the list of Andie MacDowell 's roles from the previous exercise.

We have to do a second join to get the names of the roles Andie MacDowell has played.

```
## SAMPLE SOLUTION
SELECT n.name, ci.role_id, cn.name
FROM cast_info ci
JOIN name n ON n.id = ci.person_id
JOIN char_name cn ON cn.id = ci.person_role_id
WHERE ci.person_id = 3479179;
```

Table 16: Displaying records 1 - 5

| name | role_id | name |
|---|---|---|
| MacDowell, Andie | 2 | Herself |
| MacDowell, Andie | 2 | Herself - Presenter |

| name | role_id | name |
|---|---|---|
| MacDowell, Andie | 2 | Herself - audience member |
| MacDowell, Andie | 2 | Herself |
| MacDowell, Andie | 2 | Helen Kalahan |

### Exercise

Find Annie Davis's full filmography, in chronological order. Include each movie's `title`, `production_year`, and the name of the character that she played.

Hint: You will need the cast_ing, title, and char_name tables.

```
SELECT t.title, t.production_year, cn.name
FROM cast_info ci
JOIN title t ON ci.movie_id = t.id
JOIN char_name cn ON cn.id = ci.person_role_id
WHERE ci.person_id = 3479179
  AND t.kind_id = 1
ORDER BY production_year;
```

Table 17: Displaying records 1 - 5

| title | production_year | name |
|---|---|---|
| Greystoke: The Legend of Tarzan, Lord of the Apes | 1984 | Miss Jane Porter |
| St. Elmo's Fire | 1985 | Dale Biberman |
| Sex, Lies, and Videotape | 1989 | Ann Bishop Mullany |
| Green Card | 1990 | BrontÃ« |
| The Object of Beauty | 1991 | Tina |

## Exercise

Pull up the titles of the movies that were made in my birth year 1984. There are more than 1000 so we'll put a limit on the number of movies returned.

```
SELECT t.title, t.production_year
FROM title AS t
WHERE production_year = 1984
```

```
LIMIT 100;
```

Table 18: Displaying records 1 - 5

| title | production_year |
|---|---:|
| (1984-11-04) | 1984 |
| Game 2 | 1984 |
| (#1.1) | 1984 |
| A Winter Harvest | 1984 |
| Once a Hero | 1984 |

## Exercise

Pull the one movie that Andie MacDowell was in in 1984 and bring it into R as a dataframe. Hint: We found it above.

```
SELECT t.title, t.production_year
FROM cast_info AS ci
JOIN title AS t ON ci.movie_id = t.id
WHERE t.production_year = 1984
AND ci.person_id = 3479179
AND t.kind_id = 1;
```

Table 19: 1 records

| title | production_year |
|---|---:|
| Greystoke: The Legend of Tarzan, Lord of the Apes | 1984 |

## Exercise

This website lists Nicole Kidman as the most prolific actress. Find the name and production year of all of the movies she has been in ordered by production_year descending

```
SELECT * FROM name
WHERE name = 'Kidman, Nicole';
```

Table 20: 1 records

| id | name | imdb_index | imdb_id | gender | name_pcode_cf | name_pcode_nf | surname_pcode | md5sum |
|---|---|---|---|---|---|---|---|---|
| 3353717 | Kidman, Nicole | NA | NA | f | K3524 | N2423 | K35 | 69a5531f1e2c7311fcd6c17f1124b397 |

```
SELECT t.title, t.production_year
FROM cast_info AS ci
JOIN title AS t ON ci.movie_id = t.id
WHERE ci.person_id = 3353717
AND t.kind_id = 1
ORDER BY  t.production_year DESC;
```

Table 21: Displaying records 1 - 5

| title | production_year |
|---|---|
| Destroyer | 2018 |
| Aquaman | 2018 |
| Boy Erased | 2018 |
| How to Talk to Girls at Parties | 2017 |
| The Upside | 2017 |

## Exercise

Challenge: Look up the COUNT() and GROUP function for MySQL to answer the question, which year did Nicole Kidman act in the most movies?

```
SELECT  t.production_year, COUNT(t.production_year) AS movies
FROM cast_info AS ci
JOIN title AS t ON ci.movie_id = t.id
WHERE ci.person_id = 3353717
AND t.kind_id = 1
GROUP BY t.production_year
ORDER BY movies DESC;
```

Table 22: Displaying records 1 - 5

| production_year | movies |
|---:|---:|
| 2007 | 8 |
| 2015 | 5 |
| 2014 | 5 |
| 2017 | 5 |
| 2006 | 5 |