SQL and Relational Databases

Kevin DelRosso

November 30, 2015

Setting up the workspace

```
rm(list = ls())
setwd("~/Desktop/STA_141/assignment_5/")

require(scales) # for changing alpha level in plots
require(png)
require(grid)
require(magrittr) # for updating row names (while returning object)
require(lattice)
require(RSQLite) # also DBI package

con = dbConnect( SQLite(), "./lean_imdbpy.db" )

save_plot = function(file_name, ht = 600, wd = 1000) {
    png(file_name, width = wd, height = ht, pointsize = 16)
    file_name
}
```

Exploring the database

To get a feel for the data, let's first look at the first couple rows on every table and show all the small tables. In the kind_type table we find that id = 1 is for movie, in the info_type table we find that id = 3 is for genres, and in the role_type table we find that 1 and 2 are for actor and actress respectfully.

```
# get the first 3 rows of every table in the database
tables = dbListTables( con )
table_summary = lapply( tables, function(tt) {
    query = sprintf( " SELECT * FROM %s LIMIT 3; ", tt )
    dbGetQuery( con, query )
} )

# make TRUE to show these results
names(table_summary) = tables
if( FALSE ) table_summary

# get the number of rows in each table
query = "
SELECT *
FROM sqlite_sequence;
"
dbGetQuery( con, query )
```

```
##             name      seq
## 1      kind_type        7
## 2      role_type       12
## 3      info_type      113
```

```
## 4           title  3527732
## 5            name  5375509
## 6       cast_info 48845586
## 7        aka_name  1097570
## 8       aka_title   461245
## 9      movie_info 19395234
## 10    person_info  3536735
## 11  movie_keyword  6026853
## 12        keyword   182363
## 13 movie_info_idx  1993130
```

```
# kind_type has 7 rows, let's see them all. We find id = 1 for movie
query = "
SELECT *
FROM kind_type;
"
kind = dbGetQuery( con, query )

# similarly role_type has 12 rows, we see id = 1 & 2 for actor & actress
query = "
SELECT *
FROM role_type
LIMIT 7;
"
role = dbGetQuery( con, query )

cbind( kind, role )
```

```
##   id           kind id            role
## 1  1          movie  1           actor
## 2  2      tv series  2         actress
## 3  3       tv movie  3        producer
## 4  4    video movie  4          writer
## 5  5 tv mini series  5  cinematographer
## 6  6     video game  6        composer
## 7  7        episode  7 costume designer
```

```
# in the info type we see id = 3 is for genres
query = "
SELECT *
FROM info_type
LIMIT 5;
"
# dbGetQuery( con, query )
```

1. How many actors are there in the database? How many movies?

To find the number of actors, we'll run a subquery to get the id from the role_type table corresponding to actor and actress. We found this before to be 1 and 2, but this way we can determine these values programmatically We put this subquery in the WITH clause and filtered the cast_info table to include only actors and actresses. We did essentially the same thing to determine the number of movies. Note, we only included titles with the kind = 'movie'.

```
##########
# Part 1 #
##########
query = "
WITH actor AS (
    SELECT id
    FROM role_type
    WHERE role IN ('actor', 'actress')
    )

SELECT COUNT(1) AS number_of_actors
FROM cast_info
WHERE role_id IN actor;
"
num_actors = as.numeric( dbGetQuery( con, query ) )

query = "
WITH movie AS (
    SELECT id
    FROM kind_type
    WHERE kind = 'movie'
    )

SELECT COUNT(1) AS number_of_movies
FROM title
WHERE kind_id IN movie;
"
num_movies = as.numeric( dbGetQuery( con, query ) )

vect_names = c( "number_of_actors", "number_of_movies" )
setNames( c(num_actors, num_movies), vect_names )
```

```
## number_of_actors number_of_movies
##         26221787           878800
```

2. What time period does the database cover?

We can find the range of years covered by the database by using min/max on the production_year column of the title table.

```
##########
# Part 2 #
##########

query = "
SELECT
```

```
    MIN(production_year) AS min_year,
    MAX(production_year) AS max_year
FROM title;
"
dbGetQuery( con, query )
```

```
##   min_year max_year
## 1     1874     2025
```

3. What proportion of the actors are female? male?

We first check the distinct values in the gender column, and we find it contains some NULL values. Using COALESCE we can fill in these values with 'unknown' and then get the proportion of gender in the categories 'm', 'f', and 'unknown'. We could also just exclude these NULL values, and find the proportion of those identified genders. Either way, we see that there are approximately twice as many males as females. Note, we included all people in the name database, not just actors.

```
##########
# Part 3 #
##########

# including NULL values as gender unknown
query = "
WITH count_table AS (
    SELECT
        new_gender AS gender,
        COUNT(1) AS cnt
    FROM (
        SELECT COALESCE(gender, 'unknown') AS new_gender
        FROM name )
    GROUP BY new_gender
    )

SELECT
    gender,
    cnt * 1.0 / (SELECT SUM(cnt) FROM count_table) AS proportion
FROM count_table;
"
dbGetQuery( con, query )
```

```
##    gender proportion
## 1       f  0.2302708
## 2       m  0.4207343
## 3 unknown  0.3489949
```

```
# excluding the NULL values completely
query = "
WITH count_table AS (
    SELECT
        gender,
        COUNT(1) AS cnt
    FROM name
```

```
    WHERE gender IS NOT NULL
    GROUP BY gender
    )

SELECT
    gender,
    cnt * 1.0 / (SELECT SUM(cnt) FROM count_table) AS proportion
FROM count_table;
"
dbGetQuery( con, query )
```

```
##   gender proportion
## 1      f  0.3537159
## 2      m  0.6462841
```

4. What proportion of the entries in the movies table are actual movies and what proportion are television series, etc.?

We'll need to join the title table with the kind_id table, and then group by the kind. We find that over 60% of the titles are tv episodes, while approximately 25% are movies.

```
##########
# Part 4 #
##########

query = "
SELECT
    kind,
    COUNT(1) * 1.0 / (SELECT COUNT(1) FROM title) AS proportion
FROM title t
JOIN kind_type k
ON t.kind_id = k.id
GROUP BY kind
ORDER BY proportion DESC;
"
dbGetQuery( con, query )
```

```
##          kind  proportion
## 1     episode 0.635583712
## 2       movie 0.249111894
## 3 video movie 0.041563815
## 4   tv series 0.035273371
## 5    tv movie 0.034126175
## 6  video game 0.004341033
```

5. How many genres are there? What are their names/descriptions?

We can use a subquery in the WITH clause to programmatically find the id from the info_type table associated with the genre. Using this value, we can then get at unique genres from the movie_info table. We find there are 32 genres, and their names as shown below.

```
##########
# Part 5 #
##########

query = "
WITH info_genre AS (
    SELECT id
    FROM info_type
    WHERE info = 'genres'
    )

SELECT DISTINCT info AS genre
FROM movie_info
WHERE info_type_id IN info_genre
ORDER BY genre;
"
genre = dbGetQuery( con, query )$genre

genre
```

```
##  [1] "Action"       "Adult"        "Adventure"    "Animation"
##  [5] "Biography"    "Comedy"       "Commercial"   "Crime"
##  [9] "Documentary"  "Drama"        "Erotica"      "Experimental"
## [13] "Family"       "Fantasy"      "Film-Noir"    "Game-Show"
## [17] "History"      "Horror"       "Lifestyle"    "Music"
## [21] "Musical"      "Mystery"      "News"         "Reality-TV"
## [25] "Romance"      "Sci-Fi"       "Short"        "Sport"
## [29] "Talk-Show"    "Thriller"     "War"          "Western"
```

```
length(genre)
```

```
## [1] 32
```

6. List the 10 most common genres of movies, showing the number of movies in each of these genres.

Similar to part 5, we'll now join the movie_info, info_type, kind_type, and title tables, filtering the results on the info = 'genres' and the kind = 'movie'. Then, instead of getting the unique genres lets group by the genre and get the top ten with the highest counts. We see that short, drama, comedy and documentary are the four most common.

```
##########
# Part 6 #
##########

query = "
SELECT
    m.info AS genre,
    COUNT(1) AS movie_count
FROM movie_info m
JOIN info_type i
JOIN kind_type k
```

```
JOIN title t
ON
    i.id = m.info_type_id AND
    k.id = t.kind_id AND
    t.id = m.movie_id AND
    i.info = 'genres' AND
    k.kind = 'movie'

GROUP BY m.info
ORDER BY movie_count DESC
LIMIT 10;
"
dbGetQuery( con, query )
```

```
##          genre movie_count
## 1        Short      470488
## 2        Drama      269898
## 3       Comedy      180315
## 4  Documentary      145018
## 5      Romance       52324
## 6     Thriller       51961
## 7       Action       45077
## 8       Horror       38620
## 9    Animation       38461
## 10       Crime       33010
```

7. Find all movies with the keyword 'space'. How many are there? What are the years these were released? and who were the top 5 actors in each of these movies?

First, let's create a table called movie_actor, which joins the title, cast_info, and name tables, while only including movies and actors / actresses. We chose to limit the data to only include billing position between 1 and 200 in order to speed up the computations. This made the movie_actor table roughly half the size will keeping the most relevant actors.

Next, lets join the movie_keyword and keyword tables and include only movies with a keyword which includes 'space'. To get the top 5 actors, we joined this table with the newly created movie_actor table, but only included actors with a billing between 1 and 5. This temp table is called space and contains 1201 results. In order to conveniently display all the actors names on a single row, we self joined the space table five times, matched each join on the same movie, and concatenated the actors in descending billing order.

As a quick check, *10 Things I Hate About You* is one of the first results shown and indeed Heath Ledger, Julia Stiles, and Joseph Gordon-Levitt play the three biggest roles in that movie.

```
##########
# Part 7 #
##########

# dbGetQuery( con, "DROP TABLE IF EXISTS movie_actor;" )

query = "
CREATE TABLE movie_actor AS

    WITH actor AS (
        SELECT id
```

```
        FROM role_type
        WHERE role IN ('actor', 'actress')
    ), movie AS (
        SELECT id
        FROM kind_type
        WHERE kind = 'movie'
    )

    SELECT DISTINCT
        movie_id,
        title,
        production_year,
        person_id,
        nr_order,
        name
    FROM title t
    JOIN cast_info c
    JOIN name n
    ON
        t.id = c.movie_id AND
        n.id = c.person_id AND
        t.kind_id IN movie AND
        c.role_id IN actor AND
        nr_order BETWEEN 1 AND 200;
"
# dbGetQuery( con, query )

query = "
WITH space AS (
    SELECT DISTINCT t3.*
    FROM movie_keyword t1
    JOIN (
        SELECT *
        FROM keyword
        WHERE keyword LIKE '%space%' ) t2
    JOIN movie_actor t3
    ON
        t1.keyword_id = t2.id AND
        t1.movie_id = t3.movie_id AND
        nr_order BETWEEN 1 AND 5
    ),
    final_table AS (
        SELECT DISTINCT
            t1.title,
            t1.production_year,
            t1.name || '; ' ||
            t2.name || '; ' ||
            t3.name || '; ' ||
            t4.name || '; ' ||
            t5.name AS top_five_actors
        FROM space t1
        JOIN space t2
        JOIN space t3
```

```
        JOIN space t4
        JOIN space t5
        ON
            t1.movie_id = t2.movie_id AND
            t1.movie_id = t3.movie_id AND
            t1.movie_id = t4.movie_id AND
            t1.movie_id = t5.movie_id AND
            t1.nr_order < t2.nr_order AND
            t2.nr_order < t3.nr_order AND
            t3.nr_order < t4.nr_order AND
            t4.nr_order < t5.nr_order
    )

SELECT
    *,
    ( SELECT COUNT(1) FROM final_table ) AS space_keyword_cnt
FROM final_table;
"
space_df = dbGetQuery( con, query )
head(space_df, 5)
```

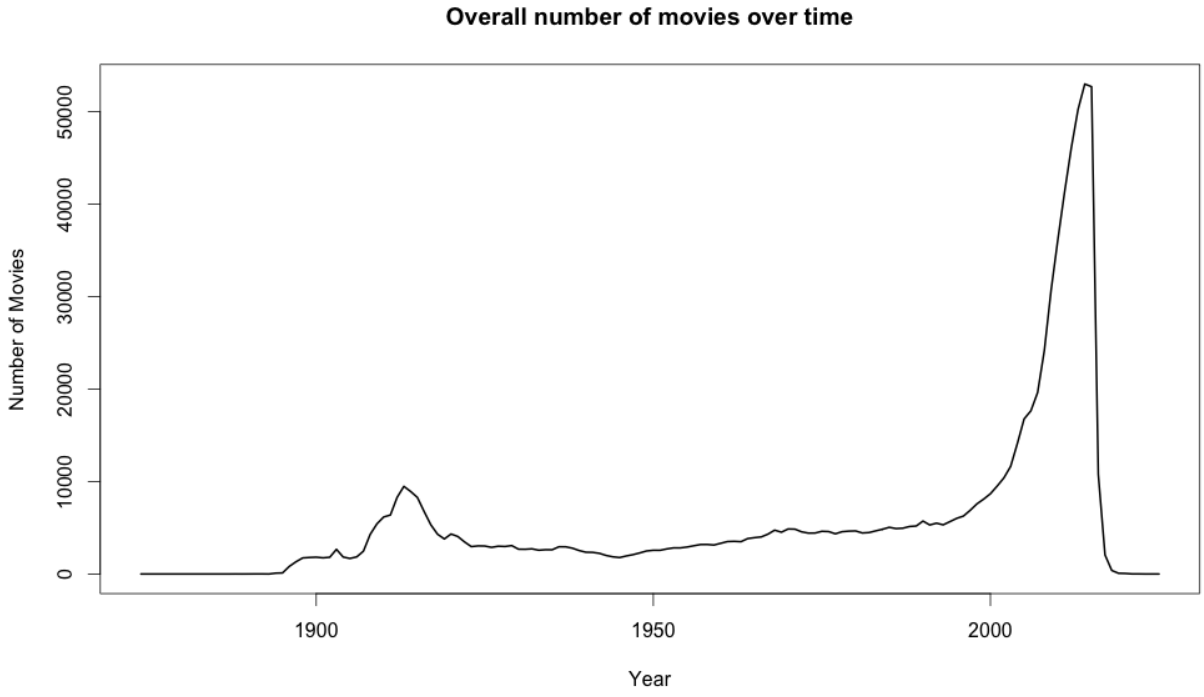
```
##                               title production_year
## 1 (T)Raumschiff Surprise - Periode 1            2004
## 2       ...4 ...3 ...2 ...1 ...morte            1967
## 3                002 operazione Luna            1965
## 4         10 + ½ + 1 = Making of 11½            2010
## 5         10 Things I Hate About You            1999
##                                                                                top_five_actors
## 1               Herbig, Michael; Kavanian, Rick; Tramitz, Christian; Kling, Anja; Schweiger, Til
## 2                   Jeffries, Lang; Persson, Essy; Dávila, Luis; Braun, Pinkas; Sibaldi, Stefano
## 3                 Franchi, Franco; Ingrassia, Ciccio; Randall, Mónica; Sini, Linda; Silva, María
## 4 Tamburini, Riccardo; Bartocci, Giacomo R.; Ananasso, Alessandro; Ceron, Corrado; Marini, Miriam
## 5          Ledger, Heath; Stiles, Julia; Gordon-Levitt, Joseph; Oleynik, Larisa; Krumholtz, David
##   space_keyword_cnt
## 1              1201
## 2              1201
## 3              1201
## 4              1201
## 5              1201
```

8. Has the number of movies in each genre changed over time? Plot the overall number of movies in each year over time, and for each genre.

The overall number of movies has an interesting looking graph. We see a small peak around 1913, however the number of movies then decreased for many years likely due to events such as the World Wars, Great Depression, and the invention of the television. Starting just before 1950, the number of movies begins to gradually increase again until about the year 2000, when the number of movies starts to increase at a near exponential rate. Note, the drop off at the far right is due to several future movies being included in the database.

```
##########
# Part 8 #
##########
```

```
# total movies over time
query = "
SELECT production_year AS year, COUNT(1) AS year_cnt
FROM title t
JOIN kind_type k
ON
    kind_id = k.id AND
    kind = 'movie' AND
    production_year IS NOT NULL
GROUP BY production_year;
"
df_year = dbGetQuery( con, query )

image = save_plot("./images/problem_8a.png")

plot( df_year$year, df_year$year_cnt, type = "l", lwd = 2,
xlab = "Year", ylab = "Number of Movies",
main = "Overall number of movies over time")
invisible( dev.off() )

grid.raster( readPNG(image) )
```

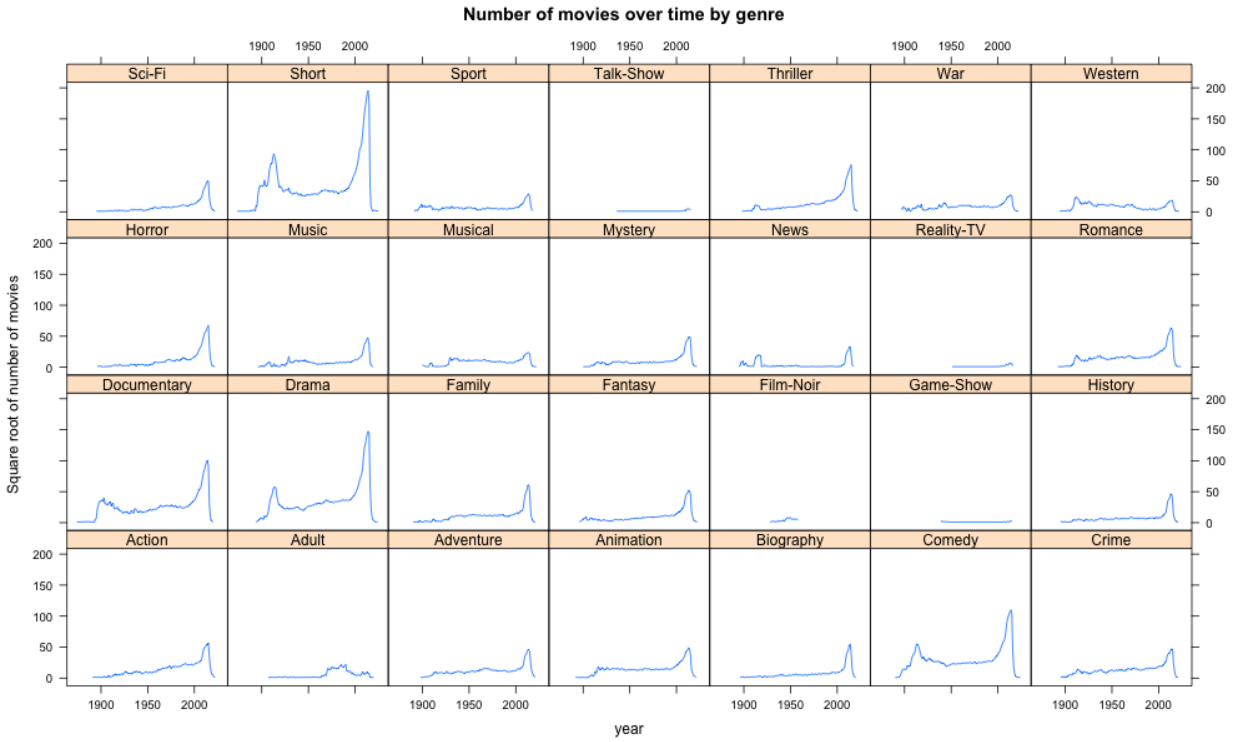


Looking at the number of movies over time by genre, we first notice the number of movies within each genre are vastly different. Short has the most, and is nearly double drama, which is in-turn nearly double comedy. This causes issues with plotting our time series if we hope to observe any details. For this reason, we chose to plot the square root of the number of movies on the y-axis. This transformation allows us to see that genres like action, romance, and thriller have been steadily increasing each year, while sport, musical, and war have remained relatively constant. Western appear to be the only genre which has steadily decreased over the years.

```
# total movies over time by genre
query = "
WITH movie_genre AS (
        SELECT info, movie_id
        FROM movie_info
        WHERE info_type_id IN ( SELECT id FROM info_type WHERE info = 'genres' )
    )

SELECT
    production_year AS year,
    info AS genre,
    COUNT(1) AS year_cnt
FROM title t
JOIN movie_genre m
JOIN kind_type k
ON
    m.movie_id = t.id AND
    kind_id = k.id AND
    kind = 'movie' AND
    production_year IS NOT NULL
GROUP BY info, production_year;
"
df_year_genre = dbGetQuery( con, query )

# getting max count within each group, to illustrate the need to change scale on plots
max_by_group = aggregate( df_year_genre, list( df_year_genre$genre ), max )
max_by_group = max_by_group[ c("genre", "year_cnt") ]
ordering = order( max_by_group$year_cnt, decreasing = TRUE )
head( max_by_group[ ordering, ] )
```

```
##          genre year_cnt
## 23       Short    38209
## 9        Drama    21811
## 6       Comedy    12117
## 8  Documentary    10125
## 26    Thriller     5785
## 15      Horror     4614
```

```
image = save_plot("./images/problem_8b.png")

xyplot( sqrt(year_cnt) ~ year | genre, data = df_year_genre, type = "l",
        ylab = "Square root of number of movies",
        main = "Number of movies over time by genre")
invisible( dev.off() )

grid.raster( readPNG(image) )
```



9. Who are the actors that have been in the most movies? List the top 20.

Using the movie_actor table created earlier, we can easily find the actors that have been in the most movies using a group by to get the counts. This process is a bit more involved in R. First we'll read in the title, cast_info, name, kind_type, and role_type tables. We'll then join them to create the movie_actor table in R (called merge_df). We can then split by person_id, get the length of each split, and find the actor names which have the 20 largest counts.

Using SQL

```
##########
# Part 9 #
##########

query = "
SELECT
    name AS sql_name,
    COUNT(1) AS sql_movie_cnt
FROM movie_actor
GROUP BY person_id
ORDER BY sql_movie_cnt DESC
LIMIT 20;
"
sql_query = dbGetQuery( con, query )
```

Using R

```
# pull all relevant tables into R
title_df = dbReadTable( con, "title" )
cast_df = dbGetQuery( con, "SELECT * FROM cast_info WHERE nr_order BETWEEN 1 AND 200;" )
```

```r
# cast_df = dbReadTable( con, "cast_info" )
name_df = dbReadTable( con, "name" )
kind_df = dbReadTable( con, "kind_type" )
role_df = dbReadTable( con, "role_type" )

# only keep movies and actors / actresses
kind_movie = kind_df$id[ kind_df$kind == "movie" ]
role_actor = role_df$id[ role_df$role %in% c( "actor", "actress" ) ]
title_df = title_df[ title_df$kind_id == kind_movie, ]
cast_df = cast_df[ cast_df$role_id %in% role_actor, ]

# remove duplicate person_id, movie_id rows
cast_df = cast_df[ ,c("person_id", "movie_id", "nr_order") ]
cast_df = cast_df[ !duplicated( cast_df ), ]

# JOIN title, cast_info, and name tables
merge_df = merge( title_df, cast_df, by.x = "id", by.y = "movie_id" )
merge_df = merge( merge_df, name_df, by.x = "person_id", by.y = "id" )

# group by person_id and get name and count
group_by = split( merge_df$name, merge_df$person_id )
cnt = sapply( group_by, length )
cnt = head( sort( cnt, decreasing = TRUE ), 20 )
actor_names = sapply( group_by[ names(cnt) ], '[', 1 )

top_20 = data.frame( r_name = actor_names, r_movie_cnt = cnt )

cbind( set_rownames( sql_query, NULL ), set_rownames( top_20, NULL ) )
```

```
##              sql_name sql_movie_cnt             r_name r_movie_cnt
## 1       Garcia, Eddie           530       Garcia, Eddie         530
## 2          Blanc, Mel           487          Blanc, Mel         487
## 3       Diaz, Paquito           468       Diaz, Paquito         468
## 4    Richardson, Jack           457    Richardson, Jack         457
## 5          Moran, Lee           454          Moran, Lee         454
## 6       Shin, Sung-il           408       Shin, Sung-il         408
## 7       Hardy, Oliver           400       Hardy, Oliver         400
## 8  Kerrigan, J. Warren           375 Kerrigan, J. Warren         375
## 9        Lyons, Eddie           372        Lyons, Eddie         372
## 10      Ford, Francis           369       Ford, Francis         369
## 11        Todd, Harry           363         Todd, Harry         363
## 12     Kennedy, Edgar           360      Kennedy, Edgar         360
## 13    Pollard, 'Snub'           355     Pollard, 'Snub'         355
## 14      Franey, Billy           350       Franey, Billy         350
## 15      Alvarado, Max           345       Alvarado, Max         345
## 16       Cobb, Edmund           344        Cobb, Edmund         344
## 17      Potel, Victor           336       Potel, Victor         336
## 18  Rawlinson, Herbert           336  Rawlinson, Herbert         336
## 19    Washburn, Bryant           336    Washburn, Bryant         336
## 20     Hatton, Raymond           333     Hatton, Raymond         333
```

10. Who are the actors that have had the most number of movies with "top billing", i.e., billed as 1, 2 or 3? For each actor, also show the years these movies spanned?

For use in this problem (and the bonus) it'll be convenient to create a new table, which contains movie actors with billing as 1, 2 or 3. We then group by person_id as done before but also include the min and max year for these movies. We can accomplish this similary in R by using the *by* function to group by person_id, find the number of nrows and min / max year in each resulting data.frame, and then order to get the top 10 results.

Using SQL

```
# dbGetQuery( con, "DROP TABLE IF EXISTS top_billing;" )

query = "
CREATE TABLE top_billing AS

    SELECT *
    FROM movie_actor
    WHERE nr_order IN (1, 2, 3);
"
# dbGetQuery( con, query )

query = "
SELECT
    name AS sql_name,
    COUNT(1) AS sql_movie_cnt,
    MIN(production_year) AS sql_year_1,
    MAX(production_year) AS sql_year_2
FROM top_billing
GROUP BY person_id
ORDER BY sql_movie_cnt DESC
LIMIT 10
"
sql_query = dbGetQuery( con, query )
```

Using R

```
top_billing = merge_df[ merge_df$nr_order %in% 1:3, ]

group_by = by( top_billing, top_billing$person_id, function(df) {
    c( df$name[1], nrow(df), min( df$production_year ), max( df$production_year ) )
    })

group_by = as.data.frame( do.call( rbind, group_by ), stringsAsFactors = FALSE)

col_names = c( "r_name", "r_movie_cnt", "r_year_1", "r_year_2" )
colnames( group_by ) = col_names

group_by$r_movie_cnt = as.numeric( group_by$r_movie_cnt )
ordering = order( group_by$r_movie_cnt, decreasing = TRUE )
top_ten = head( group_by[ ordering, ], 10 )

# shorten names for display
sql_query$sql_name = substring( sql_query$sql_name, 1, 20 )
```

```
set_rownames( sql_query, NULL )
```

```
##                sql_name sql_movie_cnt sql_year_1 sql_year_2
## 1            Blanc, Mel           473       1944       2011
## 2         Shin, Sung-il           394       1960       1992
## 3   Kerrigan, J. Warren           370       1910       1934
## 4            Moran, Lee           368       1912       1933
## 5          Lyons, Eddie           354       1911       1924
## 6  Anderson, Gilbert M.           320       1904       1922
## 7         Hardy, Oliver           311       1914       1982
## 8       Pollard, 'Snub'           301       1915       1933
## 9      Richardson, Jack           294       1911       1929
## 10        Garcia, Eddie           292       1953       2013
```

```
set_rownames( top_ten, NULL )
```

```
##                               r_name r_movie_cnt r_year_1 r_year_2
## 1                          Blanc, Mel         473     1944     2011
## 2                       Shin, Sung-il         394     1960     1992
## 3                  Kerrigan, J. Warren         370     1910     1934
## 4                          Moran, Lee         368     1912     1933
## 5                        Lyons, Eddie         354     1911     1924
## 6  Anderson, Gilbert M. 'Broncho Billy'         320     1904     1922
## 7                        Hardy, Oliver         311     1914     1982
## 8                      Pollard, 'Snub'         301     1915     1933
## 9                     Richardson, Jack         294     1911     1929
## 10                       Garcia, Eddie         292     1953     2013
```

11. Who are the 10 actors that performed in the most movies within any given year? What are their names, the year they starred in these movies and the names of the movies?

We'll first find the 10 actors that performed in the most movies within any given year by grouping by both person_id and year (done in the top_ten table). We'll then join this top_ten table with the movie_actor table to get all the titles of the movies they were in. We get all of this info in a single query, but for display purposes, we'll subset the data to get a list with an element for each unique actor-year pair in the results. We can then show these 10 actors and years, as well as a small sample of the movie titles.

Interestingly, all of these actors where creating movies around 1913, which we recall from problem 8 was that first peak in number of movies by year plot. It makes sense these movies would be from a long time again, since at least today there are many more actors and movies take much longer to make. Though not ranked #1, amazingly J. Warren Kerrigan was in 250 total movies over 3 consecutive years.

Using SQL

```
query = "
WITH top_ten AS (
        SELECT
            person_id,
            production_year,
            COUNT(1) AS movie_cnt
        FROM movie_actor
        GROUP BY person_id, production_year
        ORDER BY movie_cnt DESC
```

```
        LIMIT 10
        )

SELECT DISTINCT
    title,
    name AS sql_name,
    t.production_year AS sql_year,
    movie_cnt AS sql_cnt,
    (name || t.production_year) AS name_year
FROM top_ten t
JOIN movie_actor m
ON
    t.person_id = m.person_id AND
    t.production_year = m.production_year;
"
most_movies = dbGetQuery( con, query )

# keeping the column of titles separately for more convenient display
keep_cols = c("sql_name", "sql_year", "sql_cnt")
sql_movies = most_movies[ !duplicated( most_movies[ ,keep_cols ]), keep_cols ]

sql_titles = split( most_movies$title, most_movies$name_year )
```

Using R

```
# grouping by actor and year
grouping = list( merge_df$person_id, merge_df$name, merge_df$production_year )
group_by = aggregate( merge_df$person_id, grouping, length )

# sort count decreasing but name in alphabetical order (for ties)
# this ordering is the way it's done in SQL
ordering = order( -group_by$x, group_by$Group.2 )
r_movies = head( group_by[ ordering, c("Group.2", "Group.3", "x") ], 10 )
col_names = c( "r_name", "r_year", "r_cnt" )
colnames( r_movies ) = col_names

cbind( set_rownames( sql_movies, NULL ), set_rownames( r_movies, NULL ) )
```

```
##              sql_name sql_year sql_cnt             r_name r_year r_cnt
## 1     Barnett, Chester     1913     104    Barnett, Chester   1913   104
## 2         White, Pearl     1913     102        White, Pearl   1913   102
## 3  Kerrigan, J. Warren     1912      97 Kerrigan, J. Warren   1912    97
## 4  Kerrigan, J. Warren     1911      85 Kerrigan, J. Warren   1911    85
## 5       Sterling, Ford     1913      74       Bush, Pauline   1912    74
## 6        Bush, Pauline     1912      74      Sterling, Ford   1913    74
## 7    Onoe, Matsunosuke     1917      73   Onoe, Matsunosuke   1917    73
## 8     Richardson, Jack     1912      72    Richardson, Jack   1912    72
## 9     Richardson, Jack     1913      70    Richardson, Jack   1913    70
## 10 Kerrigan, J. Warren     1913      68 Kerrigan, J. Warren   1913    68
```

```
# getting movie title from top actors (most movies in a year)
r_titles = merge( r_movies, merge_df, by.x = c("r_name", "r_year"),
```

```
                  by.y = c("name", "production_year") )

keep_cols = c( col_names, "title" )
ordering = order( -r_titles$r_cnt, r_titles$r_name, r_titles$title )
r_titles = r_titles[ ordering, keep_cols ]
r_titles$name_year = paste( r_titles$r_name, r_titles$r_year, sep = "_" )

# show first three titles from top 3 actors (most movies in a year)
head( by( r_titles, r_titles$name_year, function(df) df$title[ 1:3 ] ), 3 )
```

```
## $`Barnett, Chester_1913`
## [1] "A Bachelor's Finish" "A Call from Home"    "A Child's Influence"
##
## $`Bush, Pauline_1912`
## [1] "A Bad Investment"  "A Life for a Kiss" "After School"
##
## $`Kerrigan, J. Warren_1911`
## [1] "$5000 Reward, Dead or Alive" "A California Love Story"
## [3] "A Cowboy's Sacrifice"
```

```
head( lapply( sql_titles, '[', c(1:3) ), 3 )
```

```
## $`Barnett, Chester1913`
## [1] "A Bachelor's Finish" "A Call from Home"    "A Child's Influence"
##
## $`Bush, Pauline1912`
## [1] "A Bad Investment"  "A Life for a Kiss" "After School"
##
## $`Kerrigan, J. Warren1911`
## [1] "$5000 Reward, Dead or Alive" "A California Love Story"
## [3] "A Cowboy's Sacrifice"
```

12. Who are the 10 actors that have the most aliases (i.e., see the aka_names table).

In order to find the 10 actors with the most aliases, we first get the unique person_id / name combinations for all movie actors (from the movie_actor table). We then join the results with the aka_name table using the person_id and group by person_id (rather than name in case of actors with the same name). Finally we order the results and take the top ten. We find that Jesus Franco has the most aliases with 78.

Using SQL

```
query = "
WITH actor AS (
    SELECT DISTINCT
        person_id,
        name
    FROM movie_actor
    )

SELECT
    t1.name AS sql_name,
    COUNT(1) AS sql_alias_cnt
FROM actor t1
```

```
JOIN aka_name t2
ON
    t1.person_id = t2.person_id
GROUP BY t1.person_id
ORDER BY sql_alias_cnt DESC
LIMIT 10;
"
sql_query = dbGetQuery( con, query )
```

Using R

```
aka_name_df = dbReadTable( con, "aka_name" )

# get all unique name, person_id pairs
alias_df = merge_df[ c("name", "person_id") ]
alias_df = alias_df[ !duplicated( alias_df ), ]

# join with aka_name table
alias_df = merge( alias_df, aka_name_df, by = "person_id" )

# group by person_id, get the size of each group
id = alias_df$person_id
group_by = aggregate( id, list( alias_df$name.x, id ), length  )

# order results, fix column names, and take the top ten
ordering = order( -group_by$x, group_by$Group.1 )
r_query = head( group_by[ ordering, c("Group.1", "x") ], 10 )
colnames( r_query ) = c( "r_name", "r_alias_cnt" )

cbind( set_rownames( sql_query, NULL ), set_rownames( r_query, NULL ) )
```

```
##            sql_name sql_alias_cnt           r_name r_alias_cnt
## 1     Franco, Jesús            78    Franco, Jesús          78
## 2      D'Amato, Joe            71     D'Amato, Joe          71
## 3     Digard, Uschi            63    Digard, Uschi          63
## 4  Savage, Herschel            53 Savage, Herschel          53
## 5       Ho, Godfrey            50      Ho, Godfrey          50
## 6     Silvera, Joey            42    Silvera, Joey          42
## 7   León, Nathanael            39  León, Nathanael          39
## 8  Clark, Christoph            38 Clark, Christoph          38
## 9       Martin, Jon            36      Martin, Jon          36
## 10 Sarno, Joseph W.            36   Mrazkova, Jana          36
```

13. Networks: Pick a (lead) actor who has been in at least 20 movies. Find all of the other actors that have appeared in a movie with that person. For each of these, find all the people they have appeared in a move with it.

We'd like to create an actor network beginning with Edward Norton. Let's first get his id. We find that there are two actors with that name, but after further exploration we find the first (id = 1478340) is the more well known actor. Using a WITH clause, we'll create a series of tables to create the actor network. Explanation of the intermediate tables are as follows:

- The movie_actor_tmp table limits the movie_actor table (again the joined table of title, cast_info, and name which is limited to only movies and actors) to billing order between 1 and 5 inclusive.

- The with_edward table finds all actors which were also in a movie with Edward Norton, person_id = 1478340. Let's call these actors first connections. There are 74 of these first connections.
- The other_movie table finds all the movies of Edward Norton's first connections.
- The with_other table finds all actors who appeared in these first connection movies. Let's call these actors second connections. We then take the union of the pairs Edward Norton to first connections, and first connections to second connections, thus creating a table with two columns which we'll use to form the network.

```
query = "
SELECT id, name, imdb_index
FROM name
WHERE name LIKE 'norton, edward';
"
dbGetQuery( con, query )
```

```
##        id           name imdb_index
## 1 1478340 Norton, Edward          I
## 2 1478341 Norton, Edward        III
```

```
query = "
WITH movie_actor_tmp AS (
        SELECT DISTINCT *
        FROM movie_actor
        WHERE nr_order BETWEEN 1 AND 5
    ),
    with_edward AS (
        SELECT DISTINCT
            t1.movie_id AS movie_id,
            t1.person_id AS first_actors,
            t2.person_id AS original_person
        FROM movie_actor_tmp t1
        JOIN (SELECT * FROM movie_actor_tmp WHERE person_id = 1478340) t2
        ON
            t1.movie_id = t2.movie_id AND
            t1.person_id <> t2.person_id
    ),
    other_movie AS (
        SELECT DISTINCT
            t2.movie_id,
            first_actors,
            original_person
        FROM with_edward t1
        JOIN movie_actor_tmp t2
        ON first_actors = t2.person_id
    ),
    with_other AS (
        SELECT DISTINCT
            t1.person_id AS actor_1,
            first_actors AS actor_2
        FROM movie_actor_tmp t1
        JOIN other_movie t2
        ON
```

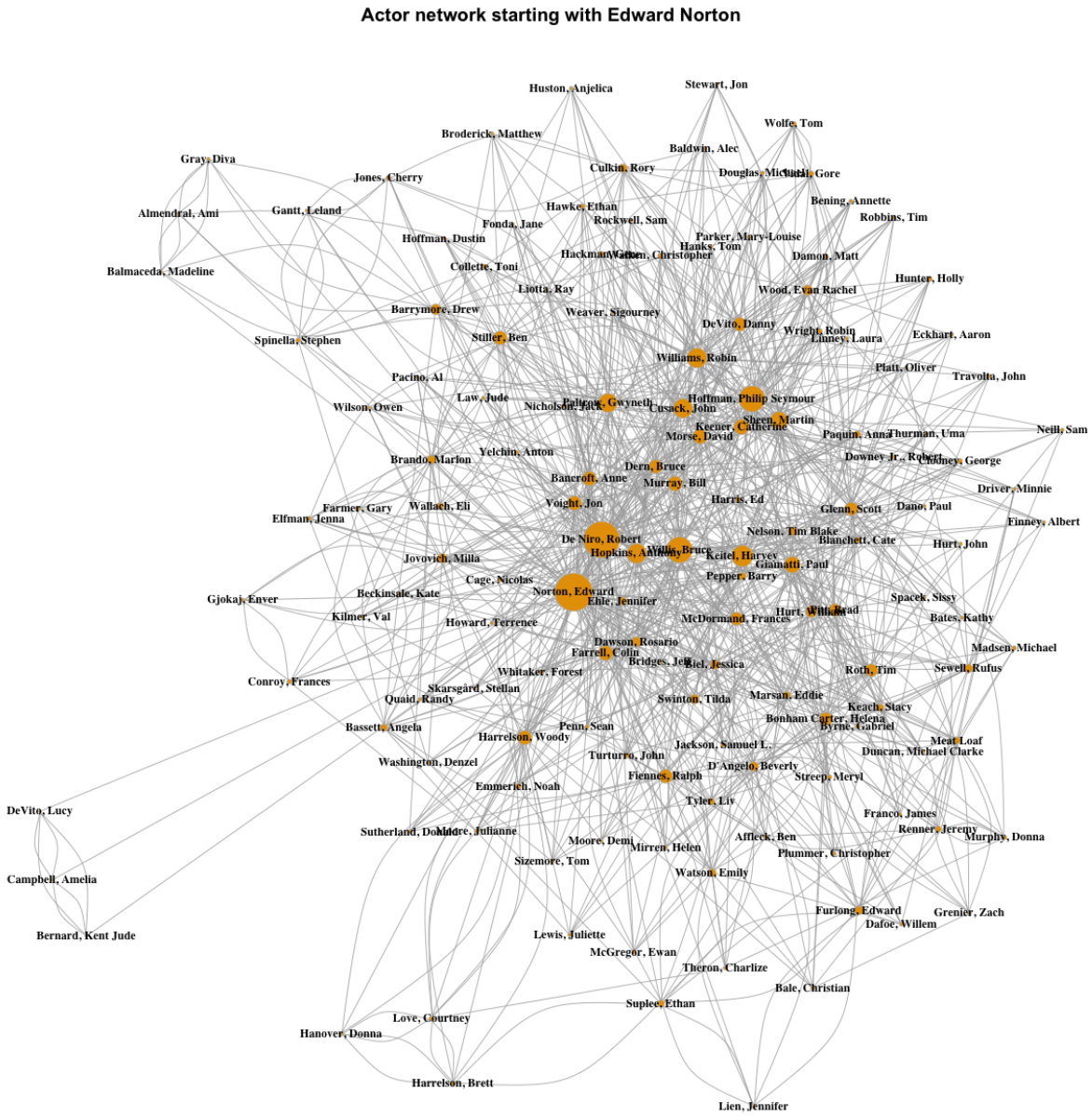
```
            t1.movie_id = t2.movie_id AND
            t1.person_id <> first_actors AND
            t1.person_id <> original_person

        UNION

        SELECT first_actors, original_person
        FROM with_edward
    )

SELECT DISTINCT
    t2.name AS actor_2,
    t3.name AS actor_1
FROM with_other t1
JOIN movie_actor_tmp t2
JOIN movie_actor_tmp t3
ON
    actor_1 = t2.person_id AND
    actor_2 = t3.person_id
"
network_df = dbGetQuery( con, query )
```

Using igraph lets turn these actor pairs into an undirected network, where nodes represent actors and edges represent that they were in a movie together. This network contains over 4,300 nodes and 7,200 edges. For display purposes, lets exclude nodes with degree less than seven, which reduces the network to 145 nodes and ~1000 edges. We also made the size of the node proportional to its degree, and hence its importance in the network.

Not surprizingly, the largest (i.e. most important) nodes in the network are Edward Norton, Roberty De Niro, Philip Seymour Hoffman, and Bruce Willis, all older movie stars that have worked with many different actors over the years. Interestingly, we also see several sub-clusters, where a few actors have all worked together and all have the same single (or very few) connection to the main network. For example Lucy Devito, Amelia Campbell, Kent Jude Bernard, and Edward Norton were all in a movie together, but those other actors haven't been in a movie with any of the other actors remaining in the netword (the movie was *Leaves of Grass* which my Google search found in the book *Six Degrees of Kevin Bacon*, go figure).

```
require(igraph)

ordering = order( network_df$actor_1, network_df$actor_2 )
network_df = network_df[ ordering, ]

actor_network = graph.data.frame( network_df, directed = FALSE )

# remove nodes with degree less than 7
to_remove = V( actor_network )[ degree( actor_network ) < 7 ]
actor_network = delete.vertices( actor_network, to_remove )

# size of node porportional to degree
V( actor_network )$size = degree( actor_network ) / 10

image = save_plot("./images/problem_13.png", 1200, 1200)
par( mar = c(0, 0, 1, 0) )
plot( actor_network,
      vertex.frame.color = 'grey',
```

```
      vertex.label.color = 'black',
      vertex.label.font = 2,
      vertex.label.cex = 0.75 )
title( main = "Actor network starting with Edward Norton" )

invisible( dev.off() )

grid.raster( readPNG(image) )
```




```
# highest degree nodes in network
degree = as.data.frame( head( sort( degree( actor_network ), decreasing = TRUE ), 10 ) )
set_colnames( degree, c( "node_degree" ) )
```

```
##                           node_degree
## Norton, Edward                     74
## De Niro, Robert                    71
## Hoffman, Philip Seymour            51
## Willis, Bruce                      51
## Keitel, Harvey                     42
## Hopkins, Anthony                   39
## Williams, Robin                    39
## Cusack, John                       38
## Paltrow, Gwyneth                   37
## Sheen, Martin                      34
```

14. Bonus Question: What are the 10 television series that have the most number of movie stars appearing in the shows?

We can find the 10 television series with the most movie stars using one long query below. We again use a WITH clause with several tables which perform the following:

- The actor / tv table contain the role_tye / kind_type for actors and episodes / tv series.
- The title_tv table performs a self join on the title table, in order to link tv episodes with their series.
- The main_table joins cast_info, name, and title_tv tables, similar to the movie_actor table used before.
- For the movie_stars table, we make the assumption that a movie star is an actor who has had top billing (billed as 1, 2 or 3) in more than 10 movies. We create a table of movie stars and join with the table containing tv series and episodes. We then get the unique rows containing the series name and actor (since we're concerned with the number of movie stars, not the number of episodes including a movie star).

Finally we group and find the series with the most movie stars is *One Life to Live*. Via Wikipedia, this is a soap opera which has been running for 43 years and contains over 11,000 episodes, so this results makes a lot of sense.

```
query = "
WITH actor AS (
        SELECT id
        FROM role_type
        WHERE role IN ('actor', 'actress')
    ),
    tv AS (
        SELECT id
        FROM kind_type
        WHERE kind IN ('episode', 'tv series')
    ),
    title_tv AS (
        SELECT
            t1.id,
            t1.title AS series,
            t2.title AS episode,
            t1.production_year
        FROM title t1
        JOIN title t2
        ON
```

```
            t1.id = t2.episode_of_id AND
            t1.kind_id IN tv
    ),
    main_table AS (
        SELECT DISTINCT
            movie_id,
            series,
            episode,
            production_year,
            person_id,
            nr_order,
            name
        FROM title_tv t
        JOIN cast_info c
        JOIN name n
        ON
            t.id = c.movie_id AND
            n.id = c.person_id AND
            c.role_id IN actor AND
            nr_order IN (1, 2, 3)
    ),
    movie_stars AS (
        SELECT DISTINCT series, t1.person_id, t1.name
        FROM main_table t1
        JOIN (
            SELECT DISTINCT name, person_id
            FROM top_billing
            GROUP BY person_id
            HAVING COUNT(1) > 10 ) t2
        ON
            t1.person_id = t2.person_id
    )

SELECT series, COUNT(1) AS movie_star_cnt
FROM movie_stars
GROUP BY series
ORDER BY movie_star_cnt DESC
LIMIT 10;
"
dbGetQuery( con, query )
```

```
##                  series movie_star_cnt
## 1      One Life to Live             25
## 2         Another World             22
## 3       Corazón salvaje              6
## 4        Estafa de amor              6
## 5                Loving              6
## 6            El enemigo              5
## 7  Secreto de confesión              5
## 8            Apasionada              4
## 9   Cumbres borrascosas              4
## 10               Malevo              4
```