CS5200 Fall 2020: Practicum 2

Chandra Davis, Evan Douglass

Overview

We've decided to work with SQLite for this practicum. As such, to work with these files you will need SQLite installed on your machine. The data we are using is provided by IMDB at https://datasets.imdbws.com/

The code below will download all the compressed tsv files required by this notebook into a folder called `data/`. You can leave them in their compressed format, as that is how we will read them in to this notebook. The downloads can take some time, so if you already have the files in a `data/` folder, we will avoid downloading them again. If you would like to replace the files, for example if new ones were published, you can change the `updateData` variable in the next code block to `TRUE`.

Setup

Before working with the data we need to create a place to store it. This section will set up a SQLite database and any constants needed later.

```
library(RSQLite)
library(readr)
library(tidyr)
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(data.table)
```

```
##
## Attaching package: 'data.table'

## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```r
library(hash)
```

```
## hash-2.2.6.1 provided by Decision Patterns

##
## Attaching package: 'hash'

## The following object is masked from 'package:data.table':
##
##     copy
```

```r
# Setup SQLite database
DB_NAME <- "imdb-data.db"
conn <- dbConnect(RSQLite::SQLite(), DB_NAME)

# Set up data folder and variables needed for download of datasets
dir.create("data", showWarnings=FALSE)

URL <- "https://datasets.imdbws.com/"
updateData <- FALSE  # Change to TRUE if you want to update these files
directory <- "./data/"
nameBasics <- "name.basics.tsv.gz"
titleBasics <- "title.basics.tsv.gz"
titleAkas <- "title.akas.tsv.gz"
titleCrew <- "title.crew.tsv.gz"
titleEpisode <- "title.episode.tsv.gz"
titlePrincipals <- "title.principals.tsv.gz"
titleRatings <- "title.ratings.tsv.gz"
```

The next section downloads all the data files needed, if they don't exist already. This may take a while if you do not already have the files.

```r
downloadData <- function(file) {
  #' Downloads a gziped file from imdb
  destination <- paste(directory,file,sep="")
  if(!file.exists(destination) || updateData){
    website <- paste(URL,file,sep="")
    download.file(website,destination)
  }
  destination # final statement in function returned
}

# Download data
# NOTE: only run this chunk once per execution of notebook, otherwise
# filepaths will get messed up.
titleAkas <- downloadData(titleAkas)
titleCrew <- downloadData(titleCrew)
titleEpisode <- downloadData(titleEpisode)
nameBasics <- downloadData(nameBasics)
titlePrincipals <- downloadData(titlePrincipals)
titleRatings <- downloadData(titleRatings)
titleBasics <- downloadData(titleBasics)
```

We will also need the following functions to read in data from files and convert them to dataframes.

```r
createDataframe <- function(file, rows=-1) {
  #' Creates a dataframe out of a gzipped tsv file. Can specify a row limit
  #' with rows, but defaults to the whole file.
  fread(file, na.strings = "\\N", encoding = "UTF-8", data.table = FALSE,
        showProgress = FALSE, nrows = rows)  # fread is a faster alternative to read.tsv
}

findMaxRowExactKey <- function(file, stopKeyValue) {
  #' Finds the row value in the gzipped tsv immediately before the given key
  #' value. stopKeyValue is assumed to be the first entry in a line of the tsv.
  #' It should also be the key above the last one you want to include, as there
  #' are some files with multiple of the same key in a row.
  fconn <- gzfile(file, 'r')
  rowNum <- -1
  while (TRUE) {
    # Read one line
    line <- readLines(fconn, n=1)
    if (length(line) == 0) {
      break
    }

    # Stop if the key is found, or on EOF
    foundLine <- startsWith(line, stopKeyValue)
    if (foundLine) {
      break
    }
    rowNum <- rowNum + 1
  }
  close(fconn)
  rowNum
}

findMaxRowLessThanKey <- function(file, stopKeyValue) {
  #' Similar to findMaxRowExactKey except it can be used in files where the
  #' exact key may not be present. Will return the row value directly below
  #' the first row with a key greater than or equal to the given key. This
  #' function is less efficient than the exact key version because it requires
  #' parsing the number part of a key.
  fconn <- gzfile(file, 'r')
  rowNum <- 0
  stopKeyNum <- parse_number(stopKeyValue)
  while (TRUE) {
    # Read one line
    line <- readLines(fconn, n=1)

    # Ignore header line
    if (startsWith(line, "tconst")) {
      next
    }

    # Split line and parse number part of key
    parts <- strsplit(line, "\t")[[1]]
    keyNum <- parse_number(parts[1])

    # Stop if the key is found, or on EOF
    if (length(line) == 0 || keyNum >= stopKeyNum) {
      break
    }
    rowNum <- rowNum + 1
  }
  close(fconn)
  rowNum
}
```

Database Schema

The following chunk creates all the necessary tables in our database based on the schema presented earlier.

```
# First we need to remove any existing data,
# for example, if this has been run before.
drop_table <- function(table_name) {
  paste("DROP TABLE IF EXISTS ", table_name, ";", sep="")
}

# Saving rows affected to a var prevents output
rows_aff <- dbExecute(conn, "PRAGMA foreign_keys = OFF;") # Avoid FK checks
curr_tables <- dbListTables(conn)
for (table in curr_tables) {
  dbExecute(conn, drop_table(table))
}
rows_aff <- dbExecute(conn, "PRAGMA foreign_keys = ON;")

# Now we can create the tables
build_table <- function(table_def) {
  CREATE <- "CREATE TABLE IF NOT EXISTS"
  paste(CREATE, table_def)
}

# Build table definition list
tables <- c(
  build_table(
    "Title_Type (
      format_id INTEGER PRIMARY KEY,
      format TEXT NOT NULL
    );"
  ),
  build_table(
    "Media (
      tconst TEXT PRIMARY KEY,
      format_id INTEGER NOT NULL,
      primaryTitle TEXT,
      originalTitle TEXT,
      isAdult INTEGER, -- 0=false, else true
      startYear INTEGER,
      endYear INTEGER,
      runtimeMins INTEGER,
      FOREIGN KEY (format_id) REFERENCES Title_Type(format_id)
    );"
  ),
  build_table(
    "Ratings (
      tconst TEXT PRIMARY KEY,
      averageRating REAL
      numVotes INTEGER,
      FOREIGN KEY (tconst) REFERENCES Media(tconst)
    );"
  ),
  build_table(
    "Episode (
      tconst TEXT PRIMARY KEY,
      parentTconst TEXT NOT NULL,
      seasonNumer INTEGER,
      episodeNumber INTEGER,
      FOREIGN KEY (tconst) REFERENCES Media(tconst),
      FOREIGN KEY (parentTconst) REFERENCES Media(tconst)
    );"
  ),
  build_table(
    "Genres (
      genre_id INTEGER PRIMARY KEY,
      genre TEXT NOT NULL
    );"
  ),
  build_table(
    "Media_Genres (
      mg_id INTEGER PRIMARY KEY,
      tconst TEXT NOT NULL,
      genre_id INTEGER NOT NULL,
      FOREIGN KEY (tconst) REFERENCES Media(tconst),
      FOREIGN KEY (genre_id) REFERENCES Genres(genre_id),
      CONSTRAINT unique_mg_tc_gid UNIQUE (tconst, genre_id)
    );"
  ),
  build_table(
    "People (
      nconst TEXT PRIMARY KEY,
      primaryName TEXT,
      birthYear INTEGER,
      deathYear INTEGER,
      age INTEGER,
      numMovies INTEGER
    );"
  ),
  build_table(
    "Known_For_Titles (
      kt_id INTEGER PRIMARY KEY,
      nconst TEXT NOT NULL,
      tconst TEXT NOT NULL,
      FOREIGN KEY (tconst) REFERENCES Media(tconst),
      FOREIGN KEY (nconst) REFERENCES People(nconst),
      CONSTRAINT unique_kft_nc_tc UNIQUE (nconst, tconst)
    );"
  ),
  build_table(
    "Professions (
      prof_id INTEGER PRIMARY KEY,
      prof_title TEXT NOT NULL
    );"
  ),
  build_table(
    "Primary_Profession (
      pp_id INTEGER PRIMARY KEY,
      nconst TEXT NOT NULL,
      prof_id INTEGER NOT NULL,
      FOREIGN KEY (nconst) REFERENCES People(nconst),
      FOREIGN KEY (prof_id) REFERENCES Professions(prof_id),
      CONSTRAINT unique_pp_nc_pid UNIQUE (nconst, prof_id)
    );"
  ),
  build_table(
    "Crew (
      tconst TEXT NOT NULL,
      ordering INTEGER NOT NULL,
      nconst TEXT NOT NULL,
      category TEXT,
      job TEXT,
      characters TEXT,
      PRIMARY KEY (tconst, ordering),
      FOREIGN KEY (tconst) REFERENCES Media(tconst),
      FOREIGN KEY (nconst) REFERENCES People(nconst)
    );"
  ),
  build_table(
    "Media_Directors (
      md_id INTEGER PRIMARY KEY,
      tconst TEXT NOT NULL,
      nconst TEXT NOT NULL,
      FOREIGN KEY (tconst) REFERENCES Media(tconst),
      FOREIGN KEY (nconst) REFERENCES People(nconst),
      CONSTRAINT unique_md_nc_tc UNIQUE (nconst, tconst)
    );"
  ),
  build_table(
    "Media_Writers (
      mw_id INTEGER PRIMARY KEY,
      tconst TEXT NOT NULL,
      nconst TEXT NOT NULL,
      FOREIGN KEY (tconst) REFERENCES Media(tconst),
      FOREIGN KEY (nconst) REFERENCES People(nconst),
      CONSTRAINT unique_mw_nc_tc UNIQUE (nconst, tconst)
    );"
  ),
  build_table(
    "Also_Known_As (
      tconst TEXT NOT NULL,
      ordering INTEGER NOT NULL,
      title TEXT,
      region TEXT,
      language TEXT,
      isOriginalTitle INTEGER, -- 0=false, else true
      PRIMARY KEY (tconst, ordering),
      FOREIGN KEY (tconst) REFERENCES Media(tconst)
    );"
  ),
  build_table(
    "Title_Types (
      type_id INTEGER PRIMARY KEY,
      type TEXT NOT NULL
    );"
  ),
  build_table(
    "Aka_Types (
      akt_id INTEGER PRIMARY KEY,
      tconst TEXT NOT NULL,
      ordering INTEGER NOT NULL,
      type_id INTEGER NOT NULL,
      -- Media table PK tconst has ref. integrity through AKA Table
      FOREIGN KEY (tconst, ordering) REFERENCES Also_Known_As(tconst, ordering),
      FOREIGN KEY (type_id) REFERENCES Title_Types(type_id),
      CONSTRAINT unique_akat_tc_ord_tid UNIQUE (tconst, ordering, type_id)
    );"
  ),
  build_table(
    "Attributes (
      att_id INTEGER PRIMARY KEY,
      att_name TEXT NOT NULL
    );"
  ),
  build_table(
    "Aka_Attributes (
      aka_id INTEGER PRIMARY KEY,
      tconst TEXT NOT NULL,
      ordering INTEGER NOT NULL,
      att_id INTEGER NOT NULL,
      -- Media table PK tconst has ref. integrity through AKA Table
      FOREIGN KEY (tconst, ordering) REFERENCES Also_Known_As(tconst, ordering),
      FOREIGN KEY (att_id) REFERENCES Attributes(att_id),
      CONSTRAINT unique_akaatt_tc_ord_aid UNIQUE (tconst, ordering, att_id)
    );"
  )
)

# Actually create the tables
for (table_stmt in tables) {
  dbExecute(conn, table_stmt)
}
```

Reading Data Locally

Next we need to load data into the database. Because of the size of the data set, we will take a small sample of it for the purposes of this practicum.

Titles

The number of titles we choose to include will determine the amount of data read in from other files. Therefore we will get this data first.

```
# We've decided arbitrarily to load 2000(?) titles/media
stopTconst <- "tt0002001"
rows <- findMaxRowExactKey(titleBasics, stopTconst)
df.tb <- createDataframe(titleBasics, rows)
tail(df.tb)
```

```
##         tconst titleType                             primaryTitle
## 1972 tt0001995     short                             Adam and Eve
## 1973 tt0001996     short        The Adventure of the Italian Model
## 1974 tt0001997     short The Adventure of the Retired Army Colonel
## 1975 tt0001998     movie    The Adventures of Lieutenant Petrosino
## 1976 tt0001999     short                           Agaton och Fina
## 1977 tt0002000     short                             The Agitator
##                                originalTitle isAdult startYear endYear
## 1972                            Adam and Eve       0      1912      NA
## 1973        The Adventure of the Italian Model       0      1912      NA
## 1974 The Adventure of the Retired Army Colonel       0      1912      NA
## 1975    The Adventures of Lieutenant Petrosino       0      1912      NA
## 1976                         Agaton och Fina       0      1912      NA
## 1977                            The Agitator       0      1912      NA
##      runtimeMinutes        genres
## 1972             11   Drama,Short
## 1973             NA   Drama,Short
## 1974             NA   Drama,Short
## 1975             NA         Drama
## 1976             NA  Comedy,Short
## 1977             10 Short,Western
```

AKAs

```
rows <- findMaxRowExactKey(titleAkas, stopTconst)
df.akas <- createDataframe(titleAkas, rows)
tail(df.akas)
```

```
##        titleId ordering                                      title region
## 4647 tt0001998        2 The Life and Death of Lieutenant Petrosino   <NA>
## 4648 tt0001998        3     The Adventures of Lieutenant Petrosino     US
## 4649 tt0001999        1                             Agaton och Fina     SE
## 4650 tt0002000        1                        The Cowboy Socialist   <NA>
## 4651 tt0002000        2                                The Agitator   <NA>
## 4652 tt0002000        3                                The Agitator     US
##      language    types attributes isOriginalTitle
## 4647     <NA>     <NA>       <NA>               0
## 4648     <NA>     <NA>       <NA>               0
## 4649     <NA>     <NA>       <NA>               0
## 4650     <NA>     <NA>       <NA>               0
## 4651     <NA> original       <NA>               1
## 4652     <NA>     <NA>       <NA>               0
```

Episodes

There are no episodes below tt0002000. Is this OK? How should we deal with this?

Ratings

```
rows <- findMaxRowLessThanKey(titleRatings, stopTconst)
df.ratings <- createDataframe(titleRatings, rows)
tail(df.ratings)
```

```
##        tconst averageRating numVotes
## 949 tt0001973           6.1      235
## 950 tt0001978           5.5       13
## 951 tt0001987           6.1       11
## 952 tt0001989           4.3        6
## 953 tt0001993           3.7       11
## 954 tt0001998           4.2       16
```

Principles

Principles will help us determine which people to include

```
rows <- findMaxRowExactKey(titlePrincipals, stopTconst)
df.princ <- createDataframe(titlePrincipals, rows)
tail(df.princ)
```

```
##         tconst ordering    nconst category  job                  characters
## 9147 tt0001999        7 nm0959818    actor <NA>                        <NA>
## 9148 tt0002000        1 nm0449857    actor <NA>       ["The Brave Cowboy"]
## 9149 tt0002000        2 nm0124189  actress <NA> ["The Rancher's Daughter"]
## 9150 tt0002000        3 nm0888126  actress <NA>                        <NA>
## 9151 tt0002000        4 nm0245385 director <NA>                        <NA>
## 9152 tt0002000        5 nm0256221   writer <NA>                        <NA>
```

Crew

Crew data also contains data about people who worked on titles/media.

```
rows <- findMaxRowExactKey(titleCrew, stopTconst)
df.crew <- createDataframe(titleCrew, rows)
tail(df.crew)
```

```
##         tconst directors   writers
## 1972 tt0001995      <NA>      <NA>
## 1973 tt0001996 nm0111753 nm0111762
## 1974 tt0001997 nm0111753      <NA>
## 1975 tt0001998 nm0325670      <NA>
## 1976 tt0001999 nm0540217      <NA>
## 1977 tt0002000 nm0245385 nm0256221
```

The Crew data has some values in directors and writers that are separated by commas. Note the 7th row in the output below. These will have to be delt with in order to figure out which people we need to include in the database.

```
head(df.crew, 10)
```

```
##       tconst           directors   writers
## 1  tt0000001           nm0005690      <NA>
## 2  tt0000002           nm0721526      <NA>
## 3  tt0000003           nm0721526      <NA>
## 4  tt0000004           nm0721526      <NA>
## 5  tt0000005           nm0005690      <NA>
## 6  tt0000006           nm0005690      <NA>
## 7  tt0000007 nm0374658,nm0005690      <NA>
## 8  tt0000008           nm0005690      <NA>
## 9  tt0000009           nm0085156 nm0085156
## 10 tt0000010           nm0525910      <NA>
```

We use tidyverse packages to split out the writers and directors into separate dataframes, with each name id on it's own line.

```
# Separate directors
df.directors <- df.crew[c("tconst", "directors")] %>%
  mutate(directors = strsplit(as.character(directors), ",")) %>%
  unnest(directors)

# Remove titles with no director listed
df.directors <- drop_na(df.directors)

head(df.directors, 10)
```

```
## # A tibble: 10 x 2
##    tconst    directors
##    <chr>     <chr>
##  1 tt0000001 nm0005690
##  2 tt0000002 nm0721526
##  3 tt0000003 nm0721526
##  4 tt0000004 nm0721526
##  5 tt0000005 nm0005690
##  6 tt0000006 nm0005690
##  7 tt0000007 nm0374658
##  8 tt0000007 nm0005690
##  9 tt0000008 nm0005690
## 10 tt0000009 nm0085156
```

```r
# Separate writers
df.writers <- df.crew[c("tconst", "writers")] %>%
  mutate(writers = strsplit(as.character(writers), ",")) %>%
  unnest(writers)

# Remove rows with no writers listed
df.writers <- drop_na(df.writers)

head(df.writers, 10)
```

```
## # A tibble: 10 x 2
##    tconst    writers
##    <chr>     <chr>
##  1 tt0000009 nm0085156
##  2 tt0000036 nm0410331
##  3 tt0000076 nm0410331
##  4 tt0000091 nm0617588
##  5 tt0000108 nm0410331
##  6 tt0000109 nm0410331
##  7 tt0000110 nm0410331
##  8 tt0000111 nm0410331
##  9 tt0000112 nm0410331
## 10 tt0000113 nm0410331
```

Names/People

Getting data from the name basics data is more complicated. Unlike the other files, there is no natural ordering on the media IDs. However, we do have name IDs for every crew member on the media that we are including. By creating a set of these name IDs we can use it to filter out people we don't want.

Unfortunately we will have to read in the whole file, so this will take some time if running it yourself. We could filter the original file and write out a new one with the filtered rows, but this requires parsing the whole file anyways, so it makes more sense to simply read in the whole file to a dataframe and then filter that.

```r
# This may take a while...
df.names <- createDataframe(nameBasics)
```

```
## Warning in fread(file, na.strings = "\\N", encoding = "UTF-8", data.table =
## FALSE, : Found and resolved improper quoting out-of-sample. First healed line
## 4956828: <<nm3470678 "Luna" Ashlee Searles \N \N \N>>. If the fields are not
## quoted (e.g. field separator does not appear within any field), try quote="" to
## avoid this warning.
```

```r
tail(df.names)
```

```
##               nconst        primaryName birthYear deathYear
## 10471367 nm9993713      Sambit Mishra        NA        NA
## 10471368 nm9993714  Romeo del Rosario        NA        NA
## 10471369 nm9993716      Essias Loberg        NA        NA
## 10471370 nm9993717 Harikrishnan Rajan        NA        NA
## 10471371 nm9993718         Aayush Nair        NA        NA
## 10471372 nm9993719          Andre Hill        NA        NA
##                         primaryProfession                 knownForTitles
## 10471367                           writer tt10709066,tt10449366,tt8325250
## 10471368 animation_department,art_department                   tt2455546
## 10471369                                                            <NA>
## 10471370                  cinematographer                       tt8736744
## 10471371                  cinematographer                            <NA>
## 10471372                                                            <NA>
```

```r
# Gather all unique name IDs into single series
allNames <- unique(
  union_all(
    as.character(df.princ$nconst),
    df.directors$directors,
    df.writers$writers
  )
)

# Add them to a hash
nameSet = hash(allNames, 1:length(allNames))  # values don't matter for the set

# Filter for used names
# This may take several minutes. May be longer than the reading...
df.names <- filter(df.names, has.key(nconst, nameSet))
tail(df.names)
```

```
##         nconst               primaryName birthYear deathYear primaryProfession
## 1763 nm9551905             Victor Darlay        NA        NA            writer
## 1764 nm9653419           Suzanne Lumière        NA        NA
## 1765 nm9735579              Rose Lumière        NA        NA
## 1766 nm9735580            Marcel Koehler        NA        NA
## 1767 nm9735581 Jeanne-Joséphine Lumière        NA        NA
## 1768 nm9880573           Isabel Ferreira        NA        NA           actress
##                                knownForTitles
## 1763                                     <NA>
## 1764                                     <NA>
## 1765 tt0221930,tt0222259,tt0791136,tt0000012
## 1766                     tt8245694,tt0000012
## 1767                                     <NA>
## 1768                                     <NA>
```

Loading Data to Database