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.

```
# Setup SQLite database
library(RSQLite)
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"
```

```
downloadData <- function(file) {
  destination <- paste(directory,file,sep="")
  if(!file.exists(destination) || updateData){
    website <- paste(URL,file,sep="")
    download.file(website,destination)
  }
  return (destination)
}
```

```
createDataframe <- function(file) {
  return (read.table(file, fill=TRUE, header=TRUE,
              sep='\t', quote = "\"", encoding="UTF-8"))
}

getData <- function(filepath) {
  return (createDataframe(filepath))
}
```

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.

```
# 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)

# TEMP NOTE: If we use the same name for each dataframe and use only when needed
# it may help the garbage collector free up some RAM once the data is in the
# database.
# Create dataframes
# df.akas <- getData(titleAkas)
# df.crew <- getData(titleCrew)
# df.ep <- getData(titleEpisode)
# df.nb <- getData(nameBasics)
# df.prin <- getData(titlePrincipals)
# df.rate <- getData(titleRatings)
# df.tb <- getData(titleBasics)
```

The following chunk creates the database schema using CREATE TABLE statements. This schema was designed for the relational model based on the above datasets.

```
# 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;")
```

```r
# 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)
}

# Remove later - for testing
dbListTables(conn)
```

```
##  [1] "Aka_Attributes"     "Aka_Types"          "Also_Known_As"
##  [4] "Attributes"         "Crew"               "Episode"
##  [7] "Genres"             "Known_For_Titles"   "Media"
## [10] "Media_Directors"    "Media_Genres"       "Media_Writers"
## [13] "People"             "Primary_Profession" "Professions"
## [16] "Ratings"            "Title_Type"         "Title_Types"
```