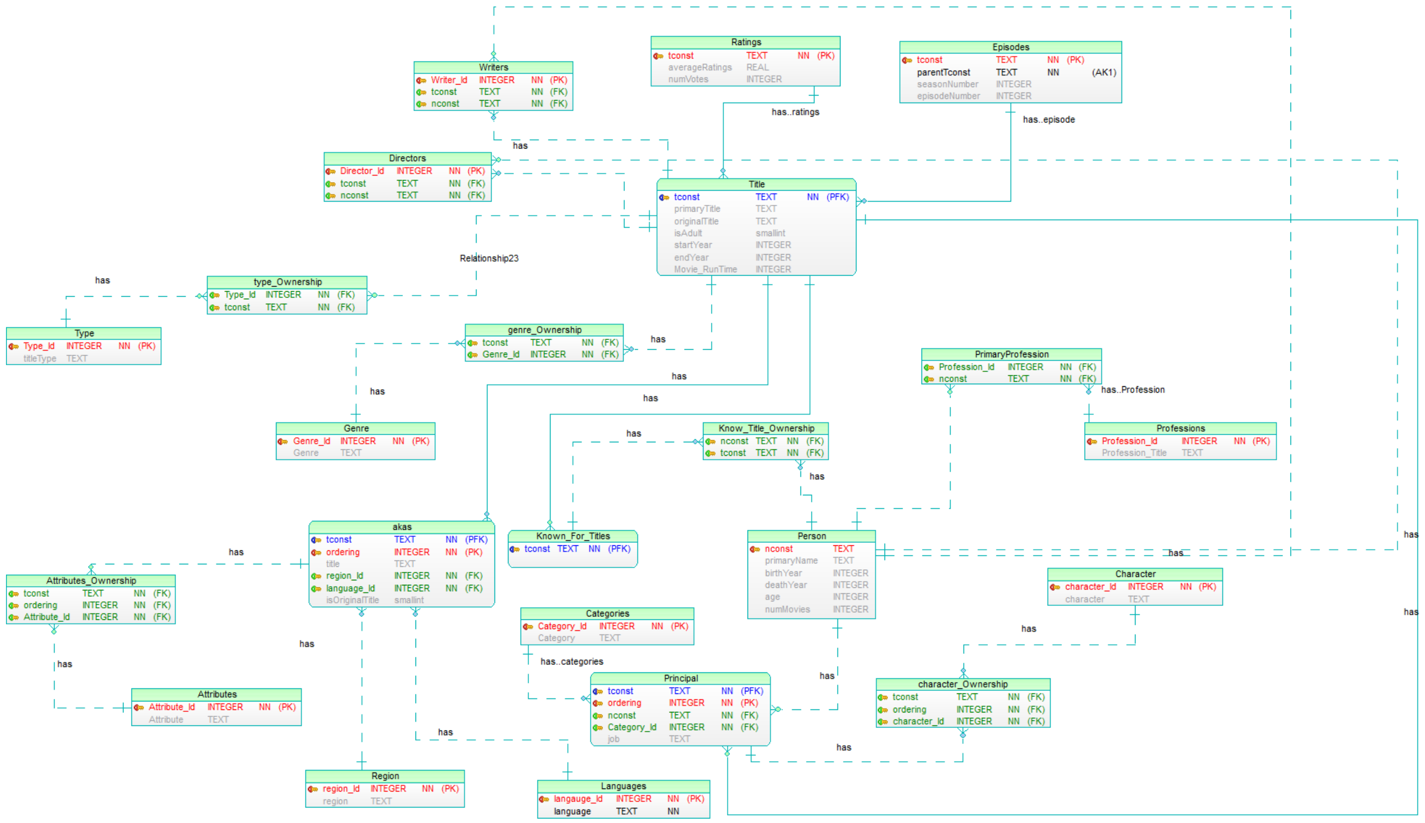
Practicum II

Kanishka Parganiha

11/23/2020

2 Importing Data

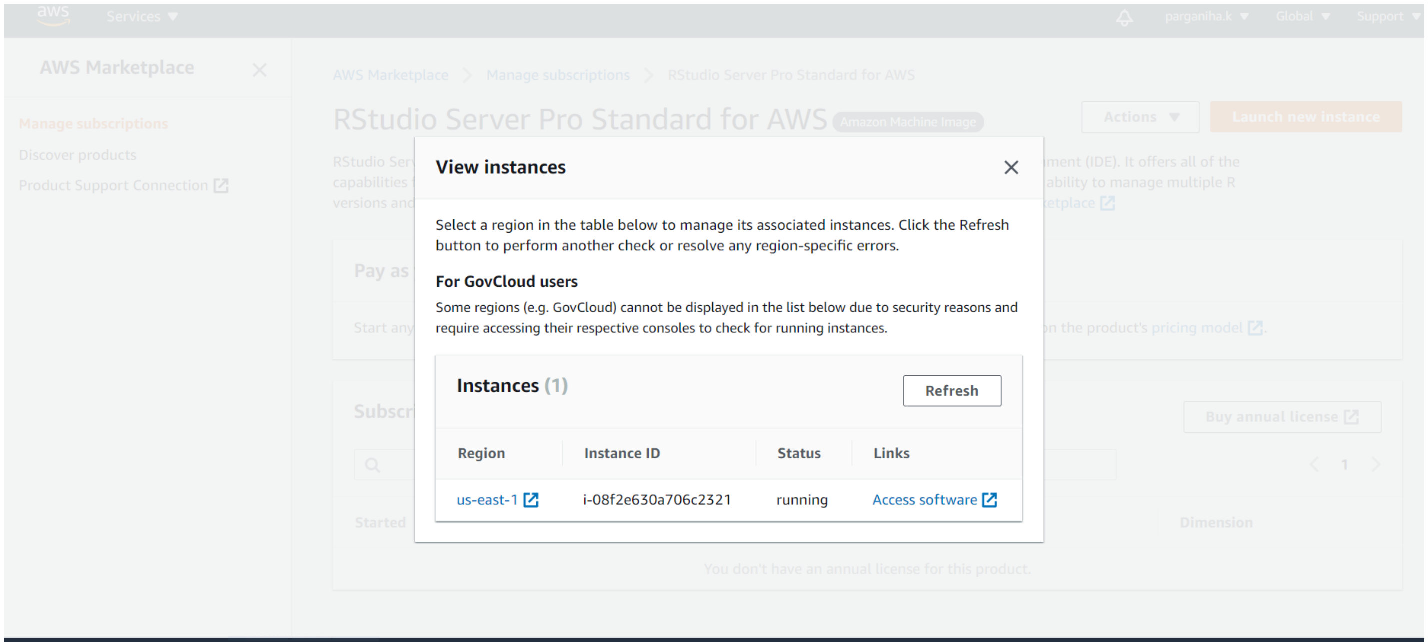
create a data model in the form of an ERD in Crow's Foot notation using a tool of your choice (e.g., LucidChart, TOAD, MySQL Workbench, etc.) and embed an image of the model in your notebook. Your ERD should be a normalized model and not a direct representation of the data in the files. Later, when you load the data you will map the data from the files to the columns in the tables.



ERD Diagram Design in MySql WorkBench

```r
#install.packages('aws.s3')
#devtools::install_github("cloudyr/aws.s3", ref="c71daa6ba3fa38df965550b7fcb251e6492c1b64")
library(aws.s3)
```

```r
Sys.setenv("AWS_ACCESS_KEY_ID" = "AKIAJLIPE6ZUYDK4YGRQ",
"AWS_SECRET_ACCESS_KEY" = "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",
"AWS_DEFAULT_REGION" = "us-east-1")
library(aws.s3)
```


aws
Services
parganiha.k
Global
Support
AWS Marketplace
Manage subscriptions
Discover products
Product Support Connection
AWS Marketplace > Manage subscriptions > RStudio Server Pro Standard for AWS
RStudio Server Pro Standard for AWS
Amazon Machine Image
Actions
Launch new instance
Buy annual license
Started
Dimension
You don't have an annual license for this product.

View instances

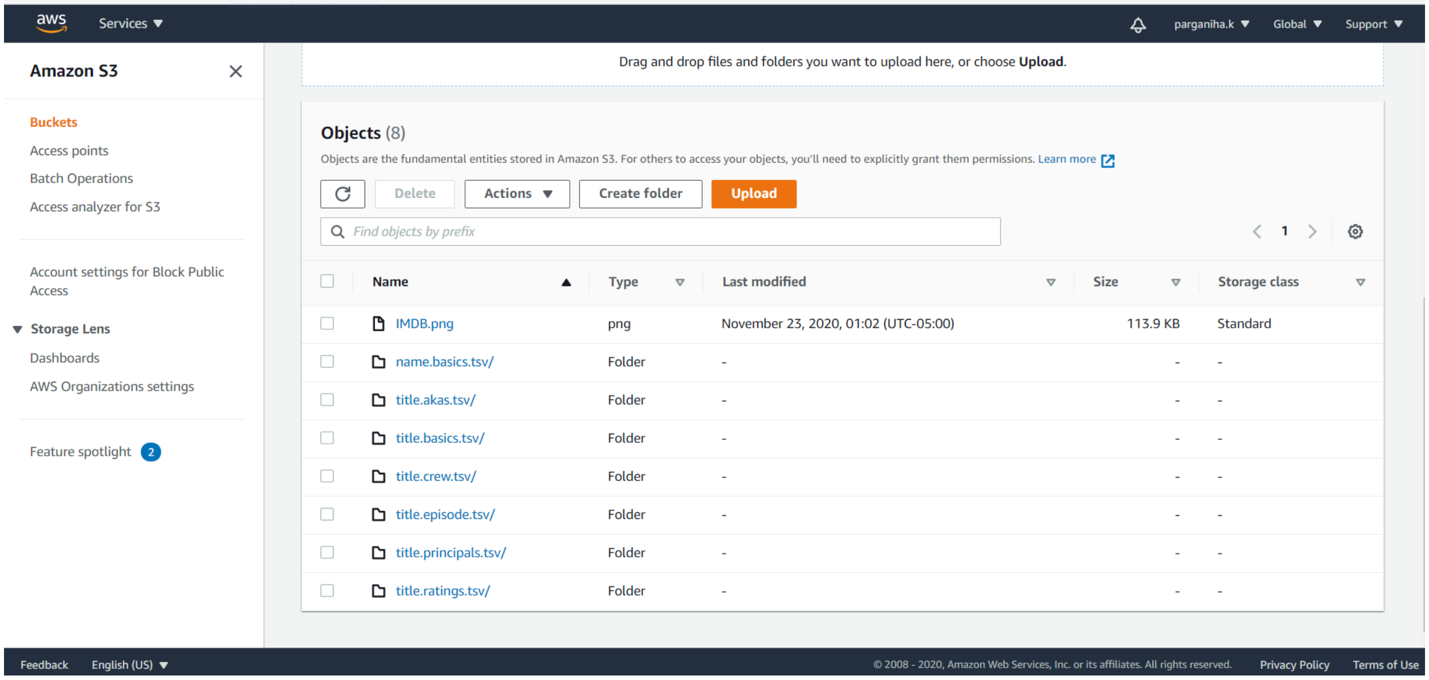
Select a region in the table below to manage its associated instances. Click the Refresh button to perform another check or resolve any region-specific errors.

For GovCloud users

Some regions (e.g. GovCloud) cannot be displayed in the list below due to security reasons and require accessing their respective consoles to check for running instances.

Instances (1) Refresh

Region	Instance ID	Status	Links
us-east-1	i-08f2e630a706c2321	running	Access software



Drag and drop files and folders you want to upload here, or choose **Upload**.

Objects (8)

Objects are the fundamental entities stored in Amazon S3. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Delete | Actions | Create folder | Upload

Find objects by prefix

Name	Type	Last modified	Size	Storage class
IMDB.png	png	November 23, 2020, 01:02 (UTC-05:00)	113.9 KB	Standard
name.basics.tsv/	Folder	-	-	-
title.akas.tsv/	Folder	-	-	-
title.basics.tsv/	Folder	-	-	-
title.crew.tsv/	Folder	-	-	-
title.episode.tsv/	Folder	-	-	-
title.principals.tsv/	Folder	-	-	-
title.ratings.tsv/	Folder	-	-	-

title.principals

```
library(tidyr)
title.principals<-read.csv(text = rawToChar(aws.s3::get_object(object ="s3://parganiha.k.imdb/ti
tle.principals.tsv/data.tsv")),sep='\t',na = c("\\N"))
```

```
## Warning in scan(file = file, what = what, sep = sep, quote = quote, dec = dec, :
## EOF within quoted string
```

```
title.principals<- title.principals %>% drop_na()
head(title.principals[,c(1,2,3,4)])
```

```
##      tconst ordering    nconst category
## 1 tt0029166        7 nm0151949   writer
## 2 tt0056810        6 nm0366454   writer
## 3 tt0066502        7 nm0504249   writer
## 4 tt0073480        8 nm0002042   writer
## 5 tt0114349        6 nm0801953   writer
## 6 tt0130357        7 nm5272248   writer
```

name.basics

```
#Library(tidyr)
name.basics<-read.csv(text = rawToChar(aws.s3::get_object(object = "s3://parganiha.k.imdb/name.b
asics.tsv/data.tsv")),sep='\t',na = c("\\N"))
name.basics<- name.basics %>% drop_na()
head(name.basics)
```

```
##      nconst     primaryName birthYear deathYear              primaryProfession
## 1 nm0000001    Fred Astaire      1899      1987 soundtrack,actor,miscellaneous
## 2 nm0000002   Lauren Bacall      1924      2014             actress,soundtrack
## 3 nm0000004    John Belushi      1949      1982        actor,soundtrack,writer
## 4 nm0000005  Ingmar Bergman      1918      2007          writer,director,actor
## 5 nm0000006  Ingrid Bergman      1915      1982    actress,soundtrack,producer
## 6 nm0000007 Humphrey Bogart      1899      1957      actor,soundtrack,producer
##                            knownForTitles
## 1 tt0072308,tt0031983,tt0050419,tt0053137
## 2 tt0071877,tt0037382,tt0038355,tt0117057
## 3 tt0077975,tt0078723,tt0072562,tt0080455
## 4 tt0050986,tt0050976,tt0060827,tt0083922
## 5 tt0038787,tt0038109,tt0036855,tt0034583
## 6 tt0040897,tt0034583,tt0043265,tt0037382
```

title.akas

```
title.akas<-read.csv(text = rawToChar(aws.s3::get_object(object = "s3://parganiha.k.imdb/title.a
kas.tsv/data.tsv")),sep='\t',na = c("\\N"))
title.akas<- title.akas %>% drop_na()
head(title.akas)
```

```
##     titleId ordering                   title region language       types
## 1 tt0022542        1   Di shtime fun Yisroel     US       yi alternative
## 2 tt0024265        4        Geleb un gelakht     US       yi alternative
## 3 tt0024751        9             Avram Ovenu     US       yi alternative
## 4 tt0026010        3 Der yidishe Kenigen Lir     US       yi alternative
## 5 tt0027911        1      Libe un Laydnshaft     US       yi alternative
## 6 tt0028902        4     Freylekhe kabtsonim     US       yi alternative
##          attributes isOriginalTitle
## 1   YIVO translation               0
## 2 modern translation               0
## 3   YIVO translation               0
## 4   YIVO translation               0
## 5 modern translation               0
## 6   YIVO translation               0
```

title.basics

```
title.basics<-read.csv(text = rawToChar(aws.s3::get_object(object=  "s3://parganiha.k.imdb/titl
e.basics.tsv/data.tsv")),sep='\t',na = c("\\N"))
title.basics<- title.basics %>% drop_na()
head(title.basics)
```

```
##      tconst titleType             primaryTitle            originalTitle isAdult
## 1 tt0035803  tvSeries The German Weekly Review Die Deutsche Wochenschau       0
## 2 tt0039120  tvSeries                Americana                Americana       0
## 3 tt0039121  tvSeries           Birthday Party           Birthday Party       0
## 4 tt0039123  tvSeries            Kraft Theatre Kraft Television Theatre       0
## 5 tt0039125  tvSeries        Public Prosecutor        Public Prosecutor       0
## 6 tt0040021  tvSeries           Actor's Studio           Actor's Studio       0
##   startYear endYear runtimeMinutes              genres
## 1      1940    1945             12    Documentary,News
## 2      1947    1949             30    Family,Game-Show
## 3      1947    1949             30              Family
## 4      1947    1958             60               Drama
## 5      1947    1951             20 Crime,Drama,Mystery
## 6      1948    1950             30               Drama
```

title.crew

```
title.crew<-read.csv(text = rawToChar(aws.s3::get_object(object = "s3://parganiha.k.imdb/title.c
rew.tsv/data.tsv")),sep='\t',na = c("\\N"))
title.crew<- title.crew %>% drop_na()
head(title.crew)
```

```
##      tconst directors   writers
## 1 tt0000009 nm0085156 nm0085156
## 2 tt0000036 nm0005690 nm0410331
## 3 tt0000076 nm0005690 nm0410331
## 4 tt0000091 nm0617588 nm0617588
## 5 tt0000108 nm0005690 nm0410331
## 6 tt0000109 nm0005690 nm0410331
```

title.episode

```
##      tconst parentTconst seasonNumber episodeNumber
## 1 tt0041951    tt0041038            1             9
## 2 tt0042816    tt0989125            1            17
## 3 tt0043426    tt0040051            3            42
## 4 tt0043631    tt0989125            2            16
## 5 tt0043693    tt0989125            2             8
## 6 tt0043710    tt0989125            3             3
```

title.ratings

```
title.ratings<-read.csv(text = rawToChar(aws.s3::get_object(object="s3://parganiha.k.imdb/title.
ratings.tsv/data.tsv")),sep='\t',na = c("\\N"))
title.ratings<- title.ratings %>% drop_na()

head(title.ratings)
```

```
##      tconst averageRating numVotes
## 1 tt0000001           5.6     1656
## 2 tt0000002           6.1      201
## 3 tt0000003           6.5     1368
## 4 tt0000004           6.2      122
## 5 tt0000005           6.2     2151
## 6 tt0000006           5.3      115
```

Setup SQLIte

```
library(RSQLite)
con <- dbConnect(RSQLite::SQLite(), "IMDb.db")

dbListTables(con)
```

```
## character(0)
```

```
# Disconnect from the database
# dbDisconnect(con)
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
```

3. Create and then run CREATE TABLE statements to build the schema. These statements must run from within your notebook and not from a separate script. Ensure proper referential integrity.’ Setting up Database

Avoiding Foriegn key check

```
dbExecute(con, "PRAGMA foreign_keys = OFF;")
```

```
## [1] 0
```

```
dbExecute(con, "CREATE TABLE IF NOT EXISTS Title
(
  tconst TEXT NOT NULL,
  primaryTitle TEXT,
  originalTitle TEXT,
  isAdult smallint,
  startYear INTEGER,
  endYear INTEGER,
  Movie_RunTime INTEGER,
  CONSTRAINT PK_Title PRIMARY KEY (tconst),
  CONSTRAINT has_ratings FOREIGN KEY (tconst) REFERENCES Ratings (tconst),
  CONSTRAINT has_episode FOREIGN KEY (tconst) REFERENCES Episodes (tconst)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS Ratings
(
  tconst TEXT NOT NULL,
  averageRatings REAL,
  numVotes INTEGER,
  CONSTRAINT PK_Ratings PRIMARY KEY (tconst)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS Episodes
(
  tconst TEXT NOT NULL,
  parentTconst TEXT,
  seasonNumber INTEGER,
  episodeNumber INTEGER,
  CONSTRAINT PK_Episodes PRIMARY KEY (tconst)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE type_Ownership
(
  Type_Id INTEGER NOT NULL,
  tconst TEXT NOT NULL,
  CONSTRAINT has FOREIGN KEY (Type_Id) REFERENCES Type (Type_Id),
  CONSTRAINT Relationship23 FOREIGN KEY (tconst) REFERENCES Title (tconst)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE Type
(
  Type_Id INTEGER NOT NULL,
  titleType TEXT,
  CONSTRAINT PK_Type PRIMARY KEY (Type_Id)
);

")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS Genre
(
  Genre_Id INTEGER NOT NULL,
  Genre TEXT,
  CONSTRAINT PK_Genre PRIMARY KEY (Genre_Id)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS genre_Ownership
(
  tconst TEXT NOT NULL,
  Genre_Id INTEGER NOT NULL,
  CONSTRAINT has FOREIGN KEY (tconst) REFERENCES Title (tconst),
  CONSTRAINT has FOREIGN KEY (Genre_Id) REFERENCES Genre (Genre_Id)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS Person
(
  nconst TEXT NOT NULL,
  primaryName TEXT,
  birthYear INTEGER,
  deathYear INTEGER,
  age INTEGER,
  numMovies INTEGER,
  CONSTRAINT PK_Person PRIMARY KEY (nconst)
);" )
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS Known_For_Titles
(
  tconst TEXT NOT NULL,
  CONSTRAINT PK_Known_For_Titles PRIMARY KEY (tconst),
  CONSTRAINT has FOREIGN KEY (tconst) REFERENCES Title (tconst)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS Know_Title_Ownership
(
  nconst TEXT NOT NULL,
  tconst TEXT NOT NULL,
  CONSTRAINT has FOREIGN KEY (nconst) REFERENCES Person (nconst),
  CONSTRAINT has FOREIGN KEY (tconst) REFERENCES Known_For_Titles (tconst)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS Professions
(
  Profession_Id INTEGER NOT NULL,
  Profession_Title TEXT,
  CONSTRAINT PK_Professions PRIMARY KEY (Profession_Id)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS PrimaryProfession
(
  Profession_Id INTEGER NOT NULL,
  nconst TEXT NOT NULL,
  CONSTRAINT has_Profession FOREIGN KEY (Profession_Id) REFERENCES Professions (Profession_Id),
  CONSTRAINT has FOREIGN KEY (nconst) REFERENCES Person (nconst)
);" )
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS Categories
(
  Category_Id INTEGER NOT NULL,
  Category TEXT,
  CONSTRAINT PK_Categories PRIMARY KEY (Category_Id)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS Principal
(
  tconst TEXT NOT NULL,
  ordering INTEGER NOT NULL,
  nconst TEXT NOT NULL,
  Category_Id INTEGER NOT NULL,
  job TEXT,
  CONSTRAINT PK_Principal PRIMARY KEY (tconst,ordering),
  CONSTRAINT has_categories FOREIGN KEY (Category_Id) REFERENCES Categories (Category_Id),
  CONSTRAINT has FOREIGN KEY (tconst) REFERENCES Title (tconst),
  CONSTRAINT has FOREIGN KEY (nconst) REFERENCES Person (nconst)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS Character
(
  character_Id INTEGER NOT NULL,
  character TEXT,
  CONSTRAINT PK_Character PRIMARY KEY (character_Id)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS character_Ownership
(
  tconst TEXT NOT NULL,
  ordering INTEGER NOT NULL,
  character_Id INTEGER NOT NULL,
  CONSTRAINT has FOREIGN KEY (tconst, ordering) REFERENCES Principal (tconst, ordering),
  CONSTRAINT has FOREIGN KEY (character_Id) REFERENCES Character (character_Id)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE Directors
(
  Director_Id INTEGER NOT NULL,
  tconst TEXT NOT NULL,
  nconst TEXT NOT NULL,
  CONSTRAINT PK_director_Ownership PRIMARY KEY (Director_Id),
  CONSTRAINT has FOREIGN KEY (tconst) REFERENCES Title (tconst),
  CONSTRAINT has FOREIGN KEY (nconst) REFERENCES Person (nconst)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS Writers
(
  Writer_Id INTEGER NOT NULL,
  tconst TEXT NOT NULL,
  nconst TEXT NOT NULL,
  CONSTRAINT PK_Writers PRIMARY KEY (Writer_Id),
  CONSTRAINT has FOREIGN KEY (tconst) REFERENCES Title (tconst),
  CONSTRAINT has FOREIGN KEY (nconst) REFERENCES Person (nconst)
);
")
```

```
## [1] 0
```

```
dbExecute(con,
"CREATE TABLE IF NOT EXISTS Region
(
  region_Id INTEGER NOT NULL,
  region TEXT,
  CONSTRAINT PK_Region PRIMARY KEY (region_Id)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"
CREATE TABLE IF NOT EXISTS Languages
(
  langauge_Id INTEGER NOT NULL,
  language TEXT NOT NULL,
  CONSTRAINT PK_Languages PRIMARY KEY (langauge_Id)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"
CREATE TABLE IF NOT EXISTS akas
(
  tconst TEXT NOT NULL,
  ordering INTEGER NOT NULL,
  title TEXT,
  region_Id INTEGER NOT NULL,
  language_Id INTEGER NOT NULL,
  isOriginalTitle smallint,
  CONSTRAINT PK_akas PRIMARY KEY (tconst,ordering),
  CONSTRAINT has FOREIGN KEY (tconst) REFERENCES Title (tconst),
  CONSTRAINT has FOREIGN KEY (region_Id) REFERENCES Region (region_Id),
  CONSTRAINT has FOREIGN KEY (language_Id) REFERENCES Languages (langauge_Id)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"
CREATE TABLE IF NOT EXISTS Types
(
  type_Id INTEGER NOT NULL,
  type TEXT,
  CONSTRAINT PK_Types PRIMARY KEY (type_Id)
);")
```

```
## [1] 0
```

```
dbExecute(con,
"
CREATE TABLE IF NOT EXISTS Attributes
(
  Attribute_Id INTEGER NOT NULL,
  Attribute TEXT,
  CONSTRAINT PK_Attributes PRIMARY KEY (Attribute_Id)
);")
```

```
## [1] 0
```

```
dbExecute(con,
          "
CREATE TABLE Attributes_Ownership
(
  tconst TEXT NOT NULL,
  ordering INTEGER NOT NULL,
  Attribute_Id INTEGER NOT NULL,
  CONSTRAINT has FOREIGN KEY (tconst, ordering) REFERENCES akas (tconst, ordering),
  CONSTRAINT has FOREIGN KEY (Attribute_Id) REFERENCES Attributes (Attribute_Id)
);")
```

```
## [1] 0
```

```
dbExecute(con, "PRAGMA foreign_keys = OFF;")
```

```
## [1] 0
```

```
dbListTables(con)
```

```
##  [1] "Attributes"           "Attributes_Ownership" "Categories"
##  [4] "Character"            "Directors"            "Episodes"
##  [7] "Genre"                "Know_Title_Ownership" "Known_For_Titles"
## [10] "Languages"            "Person"               "PrimaryProfession"
## [13] "Principal"            "Professions"          "Ratings"
## [16] "Region"               "Title"                "Type"
## [19] "Types"                "Writers"              "akas"
## [22] "character_Ownership"  "genre_Ownership"      "type_Ownership"
```

Data Preparation and Manipulation for inserting into Database

Title

```
#head(title.basics)
df.title.basics<-title.basics[,-c(2,9)]


library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────────────────────── tidyverse
1.3.0 ──
```

```
## ✓ ggplot2 3.3.2     ✓ purrr   0.3.4
## ✓ tibble  3.0.3     ✓ stringr 1.4.0
## ✓ readr   1.3.1     ✓ forcats 0.5.0
```

```
## ── Conflicts ──────────────────────────────────────────────────────────── tidyverse_confli
cts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
df.title.basics<- df.title.basics %>% drop_na()
#map(df.title.basics, ~sum(is.na(.)))

colnames(df.title.basics)<- c("tconst","primaryTitle","originalTitle","isAdult","startYear","end
Year" ,"Movie_RunTime")

dbWriteTable(con,"Title",df.title.basics, append=TRUE)

dbGetQuery(con,"select * from Title limit 10")
```

```
##       tconst                primaryTitle              originalTitle isAdult
## 1  tt0035803   The German Weekly Review   Die Deutsche Wochenschau       0
## 2  tt0039120                  Americana                  Americana       0
## 3  tt0039121             Birthday Party             Birthday Party       0
## 4  tt0039123              Kraft Theatre   Kraft Television Theatre       0
## 5  tt0039125          Public Prosecutor          Public Prosecutor       0
## 6  tt0040021             Actor's Studio             Actor's Studio       0
## 7  tt0040022 The Adventures of Oky Doky The Adventures of Oky Doky       0
## 8  tt0040023          The Alan Dale Show          The Alan Dale Show       0
## 9  tt0040024                America Song               America Song       0
## 10 tt0040026       America's Town Meeting     America's Town Meeting       0
##    startYear endYear Movie_RunTime
## 1       1940    1945            12
## 2       1947    1949            30
## 3       1947    1949            30
## 4       1947    1958            60
## 5       1947    1951            20
## 6       1948    1950            30
## 7       1948    1949            30
## 8       1948    1951            30
## 9       1948    1949            15
## 10      1948    1952            60
```

Episodes

```
df.title.episode<-title.episode

df.title.episode<- df.title.episode %>% drop_na()

dbWriteTable(con,"Episodes",df.title.episode, append=TRUE)

dbGetQuery(con,"select * FROM Episodes limit 10")
```

```
##       tconst parentTconst seasonNumber episodeNumber
## 1  tt0041951    tt0041038            1             9
## 2  tt0042816    tt0989125            1            17
## 3  tt0043426    tt0040051            3            42
## 4  tt0043631    tt0989125            2            16
## 5  tt0043693    tt0989125            2             8
## 6  tt0043710    tt0989125            3             3
## 7  tt0044093    tt0959862            1             6
## 8  tt0044668    tt0044243            2            16
## 9  tt0044901    tt0989125            3            46
## 10 tt0045519    tt0989125            4            11
```

Ratings

```
df.title.ratings<-title.ratings

df.title.ratings<- df.title.ratings %>% drop_na()

colnames(df.title.ratings)<-c("tconst","averageRatings","numVotes")

dbWriteTable(con,"Ratings",df.title.ratings, append=TRUE)

dbGetQuery(con,"select * FROM Ratings limit 10")
```

```
##       tconst averageRatings numVotes
## 1  tt0000001            5.6     1656
## 2  tt0000002            6.1      201
## 3  tt0000003            6.5     1368
## 4  tt0000004            6.2      122
## 5  tt0000005            6.2     2151
## 6  tt0000006            5.3      115
## 7  tt0000007            5.4      661
## 8  tt0000008            5.4     1820
## 9  tt0000009            5.9      155
## 10 tt0000010            6.9     6074
```

Writers

```
df.title.crew<-title.crew

#map(df.title.crew, ~sum(is.na(.)))

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

colnames(df.writers)<-c("tconst","nconst")

df.writers<- df.writers %>% drop_na()

df.writers<-tibble::rowid_to_column(df.writers, "Writer_Id")

dbWriteTable(con,"Writers",df.writers, append=TRUE)
```

Directors

```r
df.title.crew<-title.crew

#map(df.title.crew, ~sum(is.na(.)))

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

colnames(df.directors)<-c("tconst","nconst")

df.directors<- df.directors %>% drop_na()

df.directors<-tibble::rowid_to_column(df.directors, "Director_Id")

dbWriteTable(con,"Directors",df.directors, append=TRUE)
```

Genre

```r
df.genre<-title.basics[,c(1,9)]

#map(df.title.crew, ~sum(is.na(.)))

df.genre<-df.genre[c("tconst", "genres")] %>%
    mutate(genres = strsplit(as.character(genres), ",")) %>%
    unnest(genres)


df.genre<- df.genre %>% drop_na()

Genre<-unique(df.genre$genres)


genres<-tibble::rowid_to_column(data.frame(Genre), "Genre_Id")




dbWriteTable(con,"Genre",genres, append=TRUE)

x<-c()

for (i in df.genre$genres) {
x<-c(x,genres[which(genres$Genre==i),][["Genre_Id"]])
}

df.genre$Genre_Id<-x


genre_Ownership<-df.genre[,-2]

dbWriteTable(con,"genre_Ownership",genre_Ownership, append=TRUE)
```

Type

```
df.type<-title.basics[,c(1,2)]

#map(df.title.crew, ~sum(is.na(.)))


library(tidyverse)

df.type<- df.type %>% drop_na()

titleType<-unique(df.type$titleType)


titleType<-tibble::rowid_to_column(data.frame(titleType), "Type_Id")




#dbWriteTable(con,"Genre",genres, append=TRUE)

x<-c()

for (i in df.type$titleType) {
x<-c(x,titleType[which(titleType$titleType==i),][["Type_Id"]])
}

df.type$Type_Id<-x


dbWriteTable(con,"Type",titleType, append=TRUE)

dbWriteTable(con,"type_Ownership",df.type[,c(1,3)], append=TRUE)
```

Professions

```
df.Profession<-name.basics[,c(1,5)]



df.Profession<-df.Profession[c("nconst", "primaryProfession")] %>%
    mutate(primaryProfession = strsplit(as.character(primaryProfession), ",")) %>%
    unnest(primaryProfession)


df.Profession<- df.Profession %>% drop_na()

primaryProfession<-unique(df.Profession$primaryProfession)


df.primaryProfession<-tibble::rowid_to_column(data.frame(primaryProfession), "Profession_Id")



colnames(df.primaryProfession)<-c("Profession_Id","Profession_Title")

dbWriteTable(con,"Professions",df.primaryProfession, append=TRUE)

x<-c()

for (i in df.Profession$primaryProfession) {
x<-c(x,df.primaryProfession[which(df.primaryProfession$Profession_Title==i),][["Profession_Id"
]])
}

df.Profession$Profession_Id<-x


genre_Ownership<-df.Profession[,-2]

dbWriteTable(con,"PrimaryProfession",df.Profession[,-2], append=TRUE)
```

Know for Titles

```
df.knowntitles<-name.basics[,c(1,6)]



df.knowntitles<-df.knowntitles[c("nconst", "knownForTitles")] %>%
    mutate(knownForTitles = strsplit(as.character(knownForTitles), ",")) %>%
    unnest(knownForTitles)


df.knowntitles<- df.knowntitles %>% drop_na()

Known_For_Titles<-unique(df.knowntitles$knownForTitles)




colnames(df.knowntitles)<-c("nconst","tconst")

dbWriteTable(con,"Know_Title_Ownership",df.knowntitles, append=TRUE)

Known_For_Titles<-as.data.frame(Known_For_Titles)

colnames(Known_For_Titles)<-"tconst"


dbWriteTable(con,"Know_For_Titles",Known_For_Titles, append=TRUE)
```

Person

```
#head(title.basics)
df.name.basics<-name.basics[,c(1:4)]



library(tidyverse)

df.name.basics<- df.name.basics %>% drop_na()

Current_Year<-as.integer(format(Sys.Date(), "%Y"))

df.name.basics$age<-0

df.name.basics$numMovies<-0

df.name.basics[which(df.name.basics$deathYear!=0),]["age"]=df.name.basics[which(df.name.basics$d
eathYear!=0),]["deathYear"]-df.name.basics[which(df.name.basics$deathYear!=0),]["birthYear"]

df.name.basics[which(df.name.basics$deathYear==0),]["age"]=Current_Year-df.name.basics[which(df.
name.basics$deathYear==0),]["birthYear"]



dbWriteTable(con,"Person",df.name.basics, append=TRUE)




dbGetQuery(con,"select * from Person limit 10")
```

```
##       nconst     primaryName birthYear deathYear age numMovies
## 1  nm0000001    Fred Astaire      1899      1987  88         0
## 2  nm0000002   Lauren Bacall      1924      2014  90         0
## 3  nm0000004    John Belushi      1949      1982  33         0
## 4  nm0000005  Ingmar Bergman      1918      2007  89         0
## 5  nm0000006  Ingrid Bergman      1915      1982  67         0
## 6  nm0000007 Humphrey Bogart      1899      1957  58         0
## 7  nm0000008   Marlon Brando      1924      2004  80         0
## 8  nm0000009  Richard Burton      1925      1984  59         0
## 9  nm0000010    James Cagney      1899      1986  87         0
## 10 nm0000011     Gary Cooper      1901      1961  60         0
```

AlsoKnowAsTitle

```
#head(title.basics)
df.title.akas<-title.akas[,-c(6:7)]

df.title.akas<- df.title.akas %>% drop_na()



region<-unique(df.title.akas$region)


df.region_Id<-tibble::rowid_to_column(data.frame(region), "region_Id")



#dbWriteTable(con,"Professions",df.primaryProfession, append=TRUE)

x<-c()

for (i in df.title.akas$region) {
x<-c(x,df.region_Id[which(df.region_Id$region==i),][["region_Id"]])
}

df.title.akas$region_Id<-x




langauge<-unique(df.title.akas$language)


df.language_Id<-tibble::rowid_to_column(data.frame(langauge), "language_Id")


y<-c()

for (i in df.title.akas$language) {
y<-c(y,df.language_Id[which(df.language_Id$langauge==i),][["language_Id"]])
}

df.title.akas$language_Id<-y

df.title.akas$region<-df.title.akas$region_Id

df.title.akas$language<-df.title.akas$language_Id

df.title.akas<-df.title.akas[,-c(7:8)]

colnames(df.title.akas)<-c("tconst","ordering","title","region_Id","language_Id","isOriginalTitl
e" )


dbWriteTable(con,"akas",df.title.akas, append=TRUE)
```

Attributes

```r
#head(title.basics)
df.attributes<-title.akas[,c(1,2,7)]

df.attributes<- df.attributes %>% drop_na()

Attribute<-unique(df.attributes$attributes)


df.Attribute_Id<-tibble::rowid_to_column(data.frame(Attribute), "Attribute_Id")


#dbWriteTable(con,"Professions",df.primaryProfession, append=TRUE)

x<-c()

for (i in df.attributes$attributes) {
x<-c(x,df.Attribute_Id[which(df.Attribute_Id$Attribute==i),][["Attribute_Id"]])
}


df.attributes$Attribue_Id<-x

colnames(df.attributes)<-c("tconst","ordering","Attribute","Attribue_Id")

dbWriteTable(con,"Attribute_Ownership",df.attributes[,-3], append=TRUE)

dbWriteTable(con,"Attribute",df.attributes[,c(4,3)], append=TRUE)
```

Principals

```r
#head(title.basics)
df.principal<-title.principals[,-6]

df.principal<- df.principal %>% drop_na()

category<-unique(df.principal$category)


df.Category_Id<-tibble::rowid_to_column(data.frame(category), "Category_Id")


#dbWriteTable(con,"Professions",df.primaryProfession, append=TRUE)

x<-c()

for (i in df.principal$category) {
x<-c(x,df.Category_Id[which(df.Category_Id$category==i),][["Category_Id"]])
}


df.principal$Category_Id<-x


df.principal$category<-NULL

df.principal<-df.principal[,c(1,2,3,5,4)]


dbWriteTable(con,"Principal",df.principal, append=TRUE)


dbWriteTable(con,"Categories",df.Category_Id, append=TRUE)
```

```
## Warning: Column names will be matched ignoring character case
```

Characters

```
library(stringr)
#head(title.basics)
df.characters<-title.principals[,c(1,2,6)]

df.characters<- df.characters %>% drop_na()

df.characters$characters<-gsub("\\[|\\]", "", df.characters$characters)


df.characters<-df.characters %>%
    mutate(characters = strsplit(as.character(characters), ",")) %>%
    unnest(characters)

df.characters$characters <- str_remove_all(df.characters$characters, "[^0-9A-Za-z///' ]")

characters<-unique(df.characters$characters)


df.character_Id<-tibble::rowid_to_column(data.frame(characters), "Character_Id")




x<-c()

for (i in df.characters$characters) {
x<-c(x,df.character_Id[which(df.character_Id$characters==i),][["Character_Id"]])
}


df.characters$Character_Id<-x


#dbWriteTable(con,"character_Ownership",df.characters[,c(1,2,4)], append=TRUE)

colnames(df.character_Id)<-c("character_Id","character")

#dbWriteTable(con,"Character",df.character_Id, append=TRUE)
```

5. After loading the data, execute UPDATE statements for the two newly created columns in (2C). You may interpret what appearing in movies means and what you classify as movies – just make it clear in your notebook.

```
gc()
```

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   9004865  481.0   56333772 3008.6  70417214 3760.7
## Vcells 136407482 1040.8  391634604 2988.0 796699305 6078.4
```

```
dbExecute(con,"BEGIN TRANSACTION;")
```

```
## [1] 0
```

```
dbExecute(con,"UPDATE Person SET age=(select strftime('%Y','now'))-birthYear where deathYear in
 (select deathYear from Person where deathYear is  0)")
```

```
## [1] 0
```

```
dbExecute(con,"UPDATE Person SET age=deathYear-birthYear where deathYear in (select deathYear fr
om Person where deathYear is not 0)")
```

```
## [1] 161262
```

```
dbExecute(con,"UPDATE Person SET numMovies=(select count(*) from Person Pr inner join PrimaryPro
fession PP on Pr.nconst=PP.nconst inner join Professions Pf on PP.Profession_Id=Pf.Profession_Id
inner join Know_Title_Ownership K on K.nconst=Pr.nconst  where Pf.Profession_Title = 'actor' or
 Pf.Profession_Title = 'actress' group by K.nconst)")
```

```
## [1] 161262
```

```
dbExecute(con,"COMMIT;")
```

```
## [1] 0
```

6. Add triggers to the appropriate tables so that the newly created columns in (2C) are automatically updated when new data in inserted.

```
dbExecute(con,"drop trigger if exists Age")
```

```
## [1] 0
```

```
dbExecute(con,"drop trigger if exists numMovies")
```

```
## [1] 0
```

```
dbExecute(con,"CREATE TRIGGER Age after insert  on  Person
 BEGIN
update  Person  set age = (deathYear-birthYear) where nconst=new.nconst and deathYear!=0 ;
update  Person  set age = (select strftime('%Y','now')-birthYear) where nconst=new.nconst and de
athYear=0;
END;")
```

```
## [1] 0
```

```
dbExecute(con,"CREATE TRIGGER numMovies after insert on Know_Title_Ownership
 BEGIN
update  Person  set numMovies = (select numMovies from Person where nconst=new.nconst) + (select
count(distinct(new.tconst)) from Know_Title_Ownership group by nconst) where nconst=new.nconst;
END;")
```

```
## [1] 0
```

```
## Update Statement
dbExecute(con,"BEGIN TRANSACTION;")
```

```
## [1] 0
```

```
dbExecute(con,"INSERT INTO Person  VALUES('nm9999991','Kanishka Parganiha',1954,2013,0,0);")
```

```
## [1] 2
```

```
dbExecute(con,"INSERT INTO Person  VALUES('nm9999992','Him Sampat',1984,0,0,0);")
```

```
## [1] 2
```

```
dbExecute(con,"INSERT INTO Know_Title_Ownership  VALUES('nm9999992','tt0299658');")
```

```
## [1] 2
```

```
dbExecute(con,"INSERT INTO Know_Title_Ownership  VALUES('nm9999991','tt9999999');")
```

```
## [1] 2
```

```
dbExecute(con,"INSERT INTO Know_Title_Ownership  VALUES('nm9999992','tt0299699');")
```

```
## [1] 2
```

```
dbExecute(con,"INSERT INTO Know_Title_Ownership  VALUES('nm9999991','tt4999999');")
```

```
## [1] 2
```

```
dbExecute(con,"INSERT INTO Know_Title_Ownership  VALUES('nm9999991','tt5999999');")
```

```
## [1] 2
```

```
dbExecute(con,"COMMIT;")
```

```
## [1] 0
```

```
dbGetQuery(con,"select * from Person where nconst > 'nm9999990'  ")
```

```
##      nconst        primaryName birthYear deathYear age numMovies
## 1 nm9999991 Kanishka Parganiha      1954      2013  59         3
## 2 nm9999992         Him Sampat      1984         0  36         2
```

```
dbGetQuery(con,"select * from Know_Title_Ownership where nconst > 'nm9999990'  ")
```

```
##      nconst    tconst
## 1 nm9999992 tt0299658
## 2 nm9999991 tt9999999
## 3 nm9999992 tt0299699
## 4 nm9999991 tt4999999
## 5 nm9999991 tt5999999
```

7. Create a view that lists the name of each actor or actress, their age, whether they are dead or not, and how many movies they are known for based on what is stored for each actor or actress. If you work with a data sample it does not matter if the movie is actually in the database.

```
gc()
```

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   9004556  480.9   45067018 2406.9  70417214 3760.7
## Vcells 136407165 1040.8  391634604 2988.0 796699305 6078.4
```

```
dbExecute(con,"drop view if exists Actor;")
```

```
## [1] 0
```

```
dbExecute(con,"CREATE VIEW Actor as select Pr.nconst,Pr.primaryName,Pf.Profession_Title,Pr.age,P
r.numMovies from Person Pr inner join PrimaryProfession PP on Pr.nconst=PP.nconst inner join Pro
fessions Pf on PP.Profession_Id=Pf.Profession_Id inner join Know_Title_Ownership K on K.nconst=P
r.nconst  where Pf.Profession_Title = 'actor' or Pf.Profession_Title = 'actress' group by K.ncon
st")
```

```
## [1] 0
```

```
dbGetQuery(con,'select * from Actor limit 5')
```

```
##      nconst    primaryName Profession_Title age numMovies
## 1 nm0000001   Fred Astaire            actor  88         4
## 2 nm0000002  Lauren Bacall          actress  90         4
## 3 nm0000004   John Belushi            actor  33         4
## 4 nm0000005 Ingmar Bergman            actor  89         4
## 5 nm0000006 Ingrid Bergman          actress  67         4
```

8. Write a query that finds the number of seasons for each TV series. Using the results of the query create a histogram (frequency plot) with proper axis labels and title.

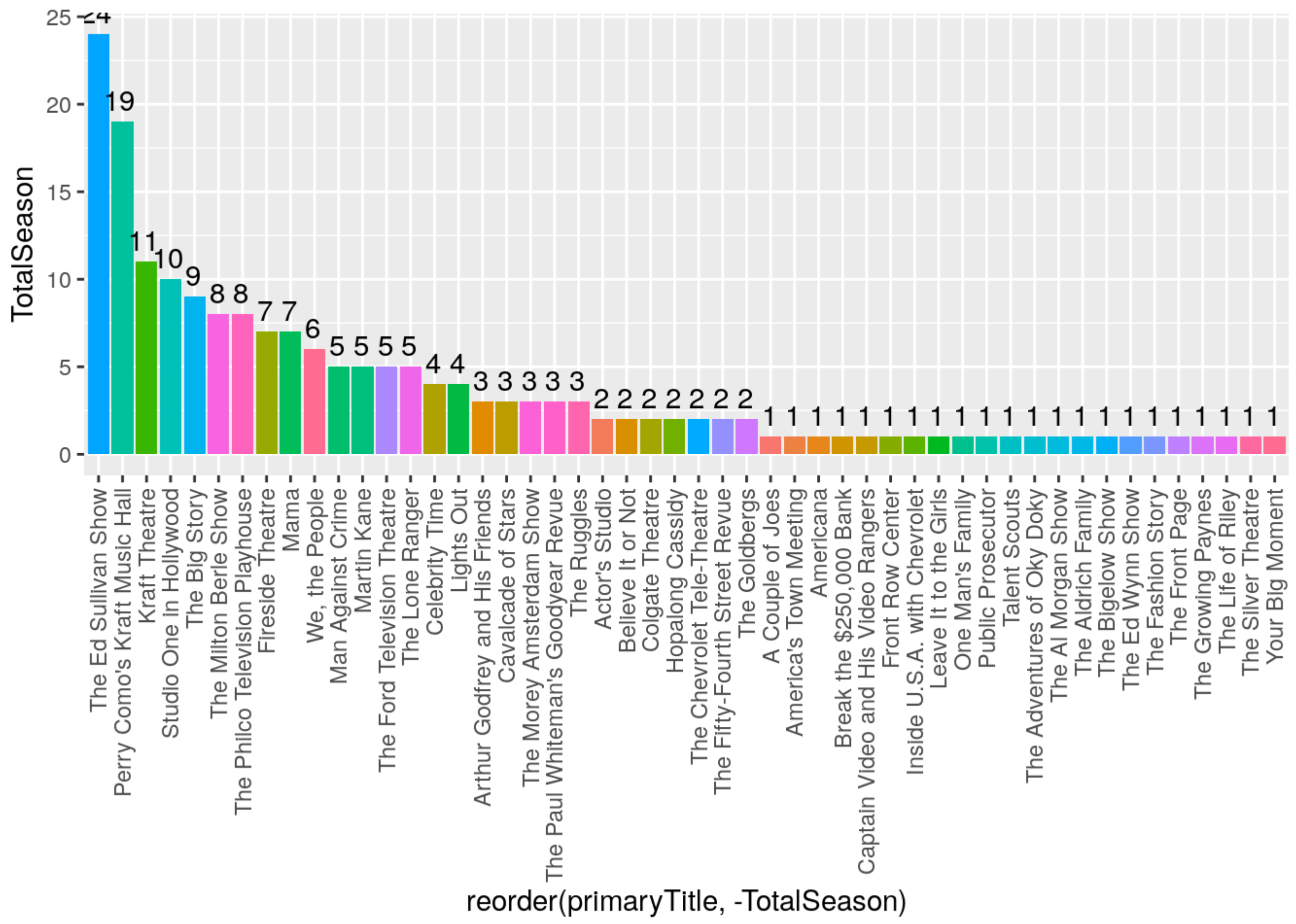
```
gc()
```

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   9004534  480.9   36053615 1925.5  70417214 3760.7
## Vcells 136407841 1040.8  391634604 2988.0 796699305 6078.4
```

```
SeasonCountNew<-dbGetQuery(con,"select T.primaryTitle,count(distinct(E.seasonNumber)) as TotalSe
ason  from Episodes E inner join Title T on E.parentTconst=T.tconst group by E.parentTconst")

head(SeasonCountNew)
```

```
##                 primaryTitle TotalSeason
## 1                  Americana           1
## 2              Kraft Theatre          11
## 3          Public Prosecutor           1
## 4             Actor's Studio           2
## 5 The Adventures of Oky Doky           1
## 6     America's Town Meeting           1
```

```
library(ggplot2)
ggplot(SeasonCountNew[1:50,], aes(x=reorder(primaryTitle,-TotalSeason), y=TotalSeason,fill = pri
maryTitle))+ geom_bar(stat = "identity")+theme(axis.text.x = element_text(angle = 90, hjust =1,v
just = 0.5))+geom_text(aes(label = TotalSeason),angle = 0, hjust=0.6,vjust=-0.5)+theme(legend.po
sition = "none")
```



9. Build a function in your code or a stored procedure in the database (approach is your choice) called addActor() that adds a new actor to the database: this requires updating several tables, so the insertions must occur within a transaction in order to function properly in a concurrent environment. Test your function by inserting a new actor – you may make up a name and associated information. Show evidence in your notebook that the actor was properly inserted.

```
gc()
```

```
##              used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells    9095614  485.8   28842892 1540.4  70417214 3760.7
## Vcells  136612338 1042.3  391634604 2988.0 796699305 6078.4
```

```
newnconst <- "nm9999993"
newprimaryName <- "Nandan Chaudhari"
newbirthYear <- as.integer(1993)
newdeathYear <- as.integer(0)
Id<-c("'tt0443489'","'tt044348d'" ,"'tt0443486'")

options(useFancyQuotes = FALSE)


 addActor<-function(newnconst,newprimaryName,newbirthYear,newdeathYear,Id)
   {

   dbExecute(con,"BEGIN TRANSACTION;")
   dbExecute(con,paste("INSERT INTO Person VALUES(",sQuote(newnconst),",",sQuote(newprimaryNam
e),",",newbirthYear,",",
                            newdeathYear,",","0",",","0",");",sep=''))
   for(i in Id){dbExecute(con,paste("INSERT INTO  Know_Title_Ownership VALUES(",sQuote(newncons
t),",",i,");"))}
   dbExecute(con,paste("INSERT INTO PrimaryProfession VALUES(",sQuote(newnconst),",",' 1',");"))
   dbExecute(con,"COMMIT;")

   }



#dbExecute(con,"rollback;")

addActor(newnconst,newprimaryName,newbirthYear,newdeathYear,Id)
```

```
## [1] 0
```

```
dbGetQuery(con,"select * from Person where nconst > 'nm9999990'  ")
```

```
##      nconst        primaryName birthYear deathYear age numMovies
## 1 nm9999991 Kanishka Parganiha      1954      2013  59         3
## 2 nm9999992         Him Sampat      1984         0  36         2
## 3 nm9999993   Nandan Chaudhari      1993         0  27         3
```

```
dbGetQuery(con,"select * from Know_Title_Ownership where nconst > 'nm9999990'  ")
```

```
##      nconst    tconst
## 1 nm9999992 tt0299658
## 2 nm9999991 tt9999999
## 3 nm9999992 tt0299699
## 4 nm9999991 tt4999999
## 5 nm9999991 tt5999999
## 6 nm9999993 tt0443489
## 7 nm9999993 tt044348d
## 8 nm9999993 tt0443486
```

10. Build a function in your code or a stored procedure in the database (approach is your choice) called deleteActor() that removes an actor from the database: this requires updating several tables, so the deletions must occur within a transaction in order to function properly in a concurrent environment. Test your function by deleting a new actor inserted in (9) – show evidence that the removal was successful.

```
gc()
```

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   9095864  485.8   28842892 1540.4  70417214 3760.7
## Vcells 136614271 1042.3  391634604 2988.0 796699305 6078.4
```

```
 deleteActor<-function(newnconst)
   {

   dbExecute(con,"BEGIN TRANSACTION;")
   dbExecute(con,paste("DELETE FROM Person where nconst=",sQuote(newnconst),sep=''))
   dbExecute(con,paste("DELETE FROM Know_Title_Ownership where nconst=",sQuote(newnconst),sep=''
))
   dbExecute(con,paste("DELETE FROM PrimaryProfession where nconst=",sQuote(newnconst),sep=''))
   dbExecute(con,"COMMIT;")

   }

deleteActor('nm9999993')
```

```
## [1] 0
```

```
deleteActor('nm9999991')
```

```
## [1] 0
```

```
deleteActor('nm9999992')
```

```
## [1] 0
```

```
dbGetQuery(con,"select * from Person where nconst > 'nm9999990'  ")
```

```
## [1] nconst      primaryName birthYear   deathYear   age         numMovies
## <0 rows> (or 0-length row.names)
```

```
dbGetQuery(con,"select * from Know_Title_Ownership where nconst > 'nm9999990'  ")
```

```
## [1] nconst tconst
## <0 rows> (or 0-length row.names)
```

11.Write a query to retrieve the names and ages of all actors who appeared in more than two movies (but not TV Movies) which an above average rating. Show the results of the query in your notebook. Do not hard code the average rating. If you did not load the title.principals.tsv file then you can use the knownFor field in the names.basic.tsv file – or the provided samples.

```
gc()
```

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   9095964  485.8   28842892 1540.4  70417214 3760.7
## Vcells 136615130 1042.3  391634604 2988.0 796699305 6078.4
```

```
dbGetQuery(con,"select distinct Pr.primaryName,Pr.age,Pr.numMovies from Person Pr
                 inner join PrimaryProfession PP on Pr.nconst=PP.nconst
                 inner join Professions Pf on PP.Profession_Id=Pf.Profession_Id
                 inner join Know_Title_Ownership K on K.nconst=Pr.nconst
                    where Pf.Profession_Title = 'actor' or Pf.Profession_Title = 'actress' and P
r.numMovies>2
                        and  K.tconst in (select tconst from Ratings  where averageRatings > (se
lect avg(averageRatings) from Ratings))
                            and tconst in (select O.tconst from Type T inner join type_Ownership
O on T.Type_Id=O.Type_Id where T.titleType='movie' )limit 5")
```

```
##       primaryName age numMovies
## 1    Fred Astaire  88         4
## 2    John Belushi  33         4
## 3  Ingmar Bergman  89         4
## 4 Humphrey Bogart  58         4
## 5   Marlon Brando  80         4
```

12. Write a query that finds an actor by name (pick a name). Measure the execution time of the query. Then create an index that would improve the performance of the query and then run and measure it again. Show the difference in a bar chart and comment on why that's the case.

```
gc()
```

```
##             used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells   9096101  485.8   28842892 1540.4  70417214 3760.7
## Vcells 136616595 1042.4  391634604 2988.0 796699305 6078.4
```

```
dbExecute(con,"DROP index if exists NEW_INDEX")
```

```
## [1] 0
```

```
start_time_1 <- Sys.time()

dbGetQuery(con,"select * from Person where primaryName='Albert Minns' ")
```

```
##      nconst  primaryName birthYear deathYear age numMovies
## 1 nm9993432 Albert Minns      1920      1985  65         4
```

```
end_time_1 <- Sys.time()

p<-round(difftime(end_time_1, start_time_1, units = "sec"),2)



##Creating Index
dbExecute(con,"CREATE INDEX NEW_INDEX ON Person(LENGTH(nconst)); ")
```

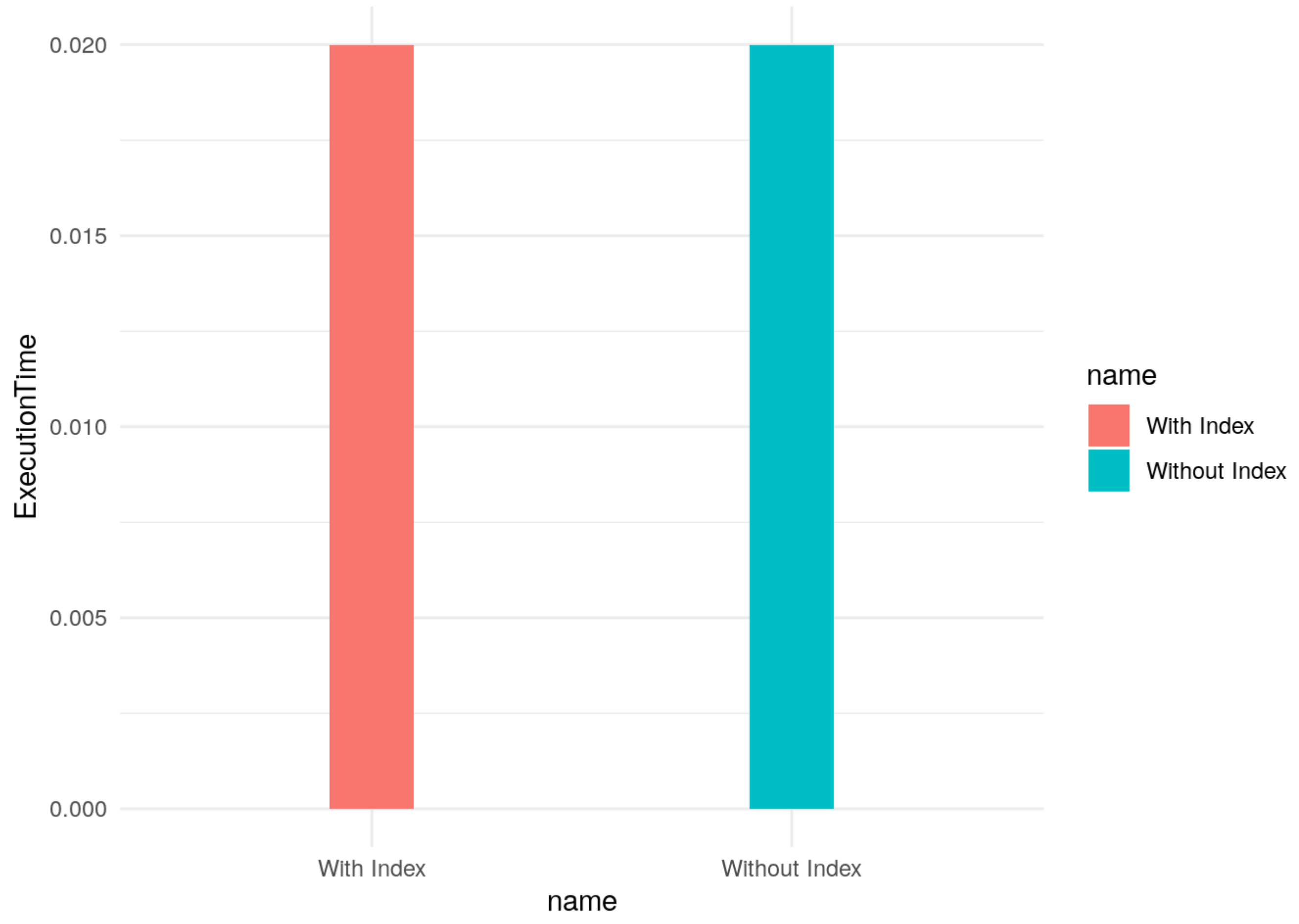
```
## [1] 0
```

```
start_time_2 <- Sys.time()

dbGetQuery(con,"select * from Person where primaryName='Albert Minns' ")
```

```
##      nconst  primaryName birthYear deathYear age numMovies
## 1 nm9993432 Albert Minns      1920      1985  65         4
```

```
end_time_2 <- Sys.time()

q<-round(difftime(end_time_2, start_time_2, units = "sec"),2)



##Display
library(ggplot2)
data <- data.frame(
    name=c("Without Index","With Index") ,
    ExecutionTime=c(p,q)
)


# Barplot

ggplot(data, aes(x=name, y=ExecutionTime,fill=name)) +
    geom_bar(stat = "identity", width=0.2) +theme_minimal()
```

```
## Don't know how to automatically pick scale for object of type difftime. Defaulting to continu
ous.
```



```
#To delete a row
#dbExecute(con,"DROP index NEW_INDEX") ;
```

When we create an index for a column in SQLite, then it maintains an ordered list of the data within the index's columns as well as their records' primary key values. SQLite uses these indexes to perform a binary search on the title values to row value = Charli Thweatt.

13. Add several indexes to a table of your choice (one containing lots of rows) and then insert additional rows into the table. Provide measurements of insert performance with no additional index, one, two, three, etc. Plot the performance change in a line graph and comment on the difference.

```
gc()
```

```
##              used   (Mb) gc trigger   (Mb)  max used   (Mb)
## Ncells    9110516  486.6   28842892 1540.4  70417214 3760.7
## Vcells  136652820 1042.6  391634604 2988.0 796699305 6078.4
```

```
dbExecute(con,"CREATE INDEX NEWINDEX ON Episodes(LENGTH(tconst)); ")
```

```
## [1] 0
```

```
start_time_1 <- Sys.time()


dbExecute(con,"INSERT INTO Episodes  VALUES('tt1111111','tt0048844',1,1); ")
```

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

```
end_time_1 <- Sys.time()

p1<-round(difftime(end_time_1, start_time_1, units = "sec"),2)

start_time_1 <- Sys.time()

dbExecute(con,"INSERT INTO Episodes  VALUES('tt1111112','tt0048844',1,2); ")
```

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

```
dbExecute(con,"INSERT INTO Episodes  VALUES('tt1111113','tt0048844',1,5); ")
```

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

```
end_time_1 <- Sys.time()

p2<-round(difftime(end_time_1, start_time_1, units = "sec"),2)

start_time_1 <- Sys.time()

dbExecute(con,"INSERT INTO Episodes  VALUES('tt1111114','tt0048844',1,2); ")
```

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

```
dbExecute(con,"INSERT INTO Episodes  VALUES('tt1111115','tt0048844',1,5); ")
```

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

```
dbExecute(con,"INSERT INTO Episodes  VALUES('tt1111116','tt0048844',1,5); ")
```

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

```
end_time_1 <- Sys.time()

p3<-round(difftime(end_time_1, start_time_1, units = "sec"),2)




dbExecute(con,"drop INDEX NEWINDEX; ")
```

```
## [1] 0
```

```
dbExecute(con,"delete from Episodes  where tconst='tt1111111'; ")
```

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

```
dbExecute(con,"delete from Episodes  where tconst='tt1111112'; ")
```

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

```
dbExecute(con,"delete from Episodes  where tconst='tt1111113'; ")
```

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

```
dbExecute(con,"delete from Episodes  where tconst='tt1111114';  ")
```

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

```
dbExecute(con,"delete from Episodes  where tconst='tt1111115';  ")
```

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

```
dbExecute(con,"delete from Episodes  where tconst='tt1111116';  ")
```

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

```
start_time_1 <- Sys.time()


dbExecute(con,"INSERT INTO Episodes  VALUES('tt1111111','tt0048844',1,1); ")
```

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

```
end_time_1 <- Sys.time()

q1<-round(difftime(end_time_1, start_time_1, units = "sec"),2)

start_time_1 <- Sys.time()

dbExecute(con,"INSERT INTO Episodes  VALUES('tt1111112','tt0048844',1,2); ")
```

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

```
dbExecute(con,"INSERT INTO Episodes  VALUES('tt1111113','tt0048844',1,5); ")
```

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

```
end_time_1 <- Sys.time()

q2<-round(difftime(end_time_1, start_time_1, units = "sec"),2)

start_time_1 <- Sys.time()

dbExecute(con,"INSERT INTO Episodes  VALUES('tt1111114','tt0048844',1,2); ")
```

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

```
dbExecute(con,"INSERT INTO Episodes  VALUES('tt1111115','tt0048844',1,5); ")
```

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

```
dbExecute(con,"INSERT INTO Episodes  VALUES('tt1111116','tt0048844',1,5); ")
```

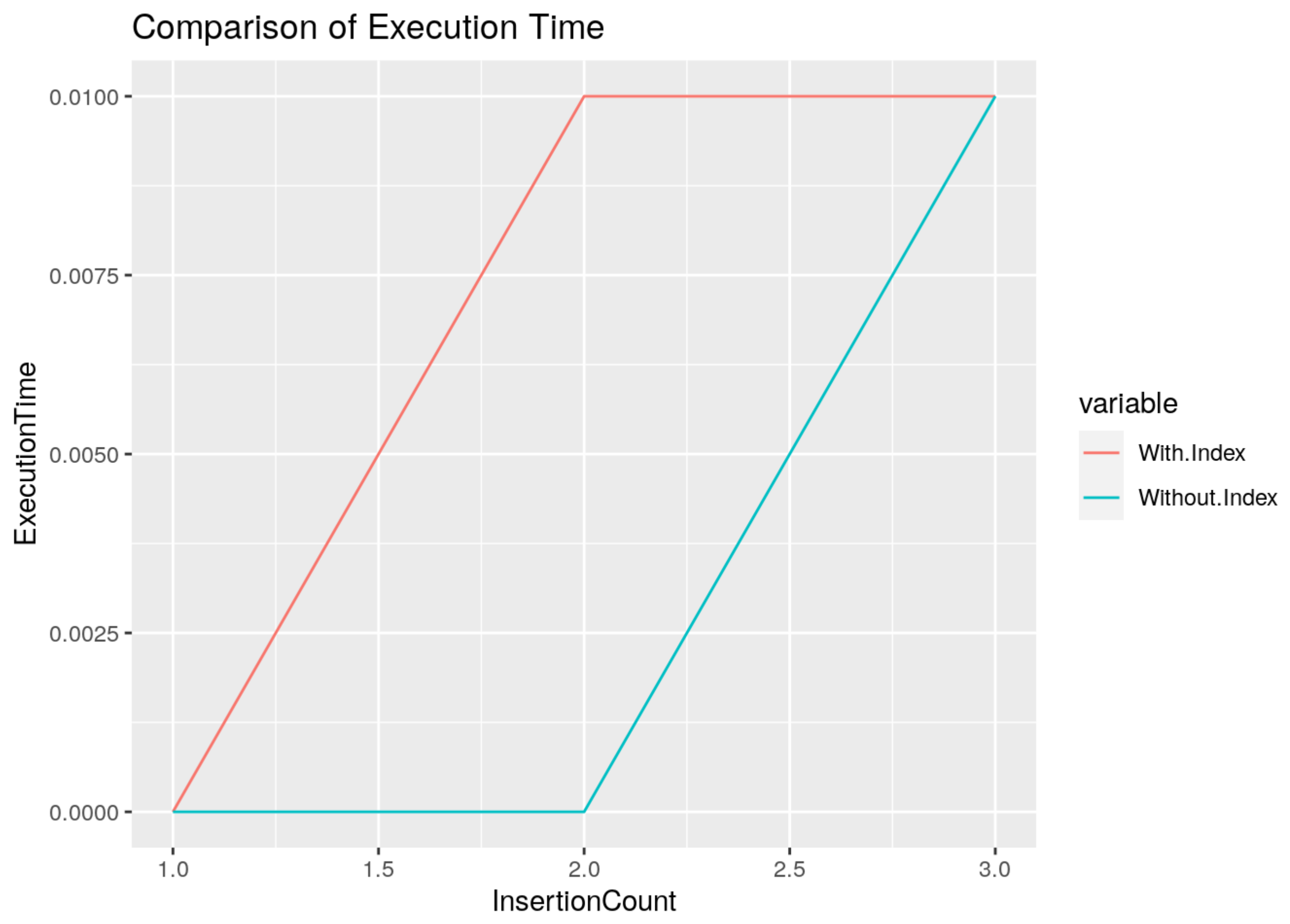
```
## [1] 1
```

```
end_time_1 <- Sys.time()

q3<-round(difftime(end_time_1, start_time_1, units = "sec"),2)









##Display
data <- data.frame(InsertionCount=c(1,2,3),
    With.Index=as.numeric(c(p1,p2,p3)) ,
    Without.Index=as.numeric(c(q1,q2,q3))
)

# LinePLot
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following object is masked from 'package:tidyr':
##
##     smiths
```

```
test_data<-melt(data, id="InsertionCount")
colnames(test_data)<-c("InsertionCount","variable","ExecutionTime")
ggplot(data=test_data,aes(x=InsertionCount, y=ExecutionTime, colour=variable)) + geom_line()+ggt
itle("Comparison of Execution Time")
```



```
dbExecute(con,"delete from Episodes  where tconst='tt1111111'; ")
```

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

```
dbExecute(con,"delete from Episodes  where tconst='tt1111112'; ")
```

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

```
dbExecute(con,"delete from Episodes  where tconst='tt1111113'; ")
```

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

```
dbExecute(con,"delete from Episodes  where tconst='tt1111114';  ")
```

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

```
dbExecute(con,"delete from Episodes  where tconst='tt1111115';  ")
```

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

```
dbExecute(con,"delete from Episodes  where tconst='tt1111116';  ")
```

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

```
#dbGetQuery(con,"select * from Episodes where parentTconst='tt0048844' ")
```

When we insert records in your indexed tables, each of the insertion operations will take slightly longer when there are indexes on the table than when there are no indexes. This is due to the presence of indexes on the table as, during insertion operation, the database must make sure the new entry is also found via these indexes. For this reason, it has to add the new entry to each and every index on that table. The number of indexes is therefore a multiplier for the cost of an insert statement.