Assignment - SQL and R

Jawaid Hakim

2022-09-08

Contents

1 Introduction

SQL databases are widely used repositories for mission critical data. This solution illustrates connection to SQL databases from R. Both local and remote databases are explored. A remote AWS RDS MySQL database is available out of the box. However, a local MySQL database can also be easily set up and configured for use with this solution.

2 Database schema

Schema for this assignment is normalized into 3 tables: FRIENDS, MOVIES, and RATINGS. Normalization has many benefits including efficient storage and reduced operational overhead. Each entity can evolve independently. For instance, adding additional details to FRIENDS, like the date of birth or address, can be done without impacting other entities. Additionally, database maintenance overhead is also reduced - e.g., changing the first name of a FRIEND can be done by updating just a single row.

2.1 Primary Keys

FRIENDS and MOVIES tables each have a primary key named ID. This key is auto-generated by the database and the sequence is initialized to 1 - i.e. first record in table has ID = 1.

2.2 Foreign Keys

Foreign key relationships are created to enforce referential integrity. In this schema, foreign key constraints have been set in the RATINGS table. Click ER diagram to see foreign key relationships.

2.3 Missing Ratings

It is possible, even highly likely, that not all friends would have viewed all movies. This database schema accommodates this scenario by design - only rated movies need be loaded into RATINGS table.

However, to allow 'NA' ratings to be loaded, the RATINGS.RATING columns is defined as an ENUM [0, 1, 2, 3, 4, 5], where 0 (default) is reserved for unrated movies.

3 Database

This solution works with either a pre-configured remote AWS (RDS) MySQL database or a local MySQL database.

3.1 Remote AWS (RDS) MySQL Database

A pre-configured AWS MySQL database has been set up and pre-populated with data to make it easy to run this R solution. Creating the AWS MySQL database was straightforward using the wizard. One caveat was that the default security group rules do not permit access to RDS from external IP addresses. In other words, connection to RDS over the open internet is not allowed.

To get around this limitation, I created a custom security group in AWS and configured it's **Inbound rules** as follows: **Protocol**: TCP, **Type**: MYSQL/AURORA, **Port range**: 3306, **Source**:0.0.0.0/0. Finally, I assigned this security group to the RDS database. RDS can now be accessed over the internet from R Studio and MySQL Workbench.

Note: connection to any database assumes the server is in a running state. Connection attempts will fail in case the preconfigured RDS database is shutdown.

3.1.1 Test Remote Database Connection

To test connectivity to the RDS MySQL database do the following:

1. Start MySQL Workbench
2. Click + button next to **My SQLConnections**. The **Setup New Connection** windows will launch
3. **Connection name:** Enter any name of your choosing
4. **Hostname:** Enter the **Endpoint** of the RDS MySQL database. Endpoint for the pre-configured RDS database is **cuny-ds.c5iiratvieki.us-east-1.rds.amazonaws.com**
5. **Username**: Enter **guest** for the pre-configured RDS database
6. **Password:** Enter **guestpass** for the pre-configured RDS database
7. Click **Test Connection** to validate connection then click **OK** to finish
8. Log into RDS MySQL using the new connection. The **Assignment - SQL and R** schema will be visible. Enjoy!

3.2 Local Database Setup

Make sure you have MySQL and MySQL Workbench installed. Admin database privileges will be required for creating database

3.2.1 Create Local Database

1. Download schema creation script to local storage
2. In MySQL Workbench click **File - Open SQL Script**, select downloaded file to load into editor, and execute script. On successful execution, schema will be created in MySQL database

3.2.2 Populate Local Database

1. Load data into FRIENDS table
   - Download FRIENDS data to local storage
   - In MySQL Workbench right-mouse click **Assignment - SQL and R -> Tables -> FRIENDS** table and select **Table Data Import Wizard**. Use wizard to load downloaded FRIENDS data into the FRIENDS table
2. Load data into MOVIES table
   - Download MOVIES data to local storage
   - In MySQL Workbench right-mouse click **Assignment - SQL and R -> Tables -> MOVIES** table and select **Table Data Import Wizard**. Use wizard to load downloaded MOVIES data into the MOVIES table
3. Load data into RATINGS table
   - Download RATINGS data to local storage
   - In MySQL Workbench right-mouse click **Assignment - SQL and R -> Tables -> RATINGS** table and select **Table Data Import Wizard**. Use wizard to load downloaded RATINGS data into the RATINGS table

```
library(DBI)
library(RMySQL)
library(RMariaDB)
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(knitr)
```

4 Database Connection

For security reasons, database connection details may be stored in a **CNF** file and not exposed in R scripts. This R script uses this technique and the CNF file **must** be downloaded from github and installed as described below.

4.1 Install CNF File

1. Download CNF file to local storage in a folder that is accessible by R runtime e.g., current working directory. From the R console, run **getwd()** command to see the current working directory. To change the working directory, use the **setwd()** command. Make sure the downloaded CNF file is in the working directory.

Modify **assignment_sql_and_r.cnf** as appropriate for your MySQL installation. For example, CNF has two configurations: **local_movie_ratings** for a local MySQL database and **remote_movie_ratings** for the pre-configured AWS MySQL. Note: the local configuration assumes there is a **guest** database user.

Set variable with name of CNF file. As mentioned above, path to the CNF is the current working directory:

```
cnf.settingsfile <- 'assignment_sql_and_r.cnf'
```

4.2 Remote Database Connection

Create functions to connect to either the remote or local database. Remote database connections use the using the **RMySQL** driver and local connections use the **MariaDB** driver:

```
my.dbConnectRemote <- function() {
    cnf.group <- 'remote_movie_ratings'
    db <- dbConnect(RMySQL::MySQL(), default.file=cnf.settingsfile, group=cnf.group)
    db
}

my.dbConnectLocal <- function() {
cnf.group <- 'local_movie_ratings'
    db <- dbConnect(RMariaDB::MariaDB(), default.file=cnf.settingsfile, group=cnf.group)
    db
```

```
}
my.dbConnectRemoteOrLocal <- function(rem_or_local = "remote") {
    if (startsWith(rem_or_local, 'r')) {
        db <- my.dbConnectRemote()
    } else {
        db <- my.dbConnectLocal()
    }
    db
}
```

5 Select Remote or Local Database

Now we select the database to which to connect, remote or local Once the connection is established to the appropriate database, rest of the R script can remain unchanged:

```
db <- my.dbConnectRemoteOrLocal('remote') # AWS MySQL database
#db <- my.dbConnectLocal('local') # Local MySQL database
```

For sanity check, list the tables. As expected, FRIENDS, MOIVIES, and RATINGS tables are listed.

```
dbListTables(db)
```

```
## [1] "FRIENDS" "MOVIES"  "RATINGS"
```

6 Load FRIENDS and MOVIES

Load the FRIENDS table:

```
qry <- 'SELECT * FROM FRIENDS ORDER BY FIRST_NAME, LAST_NAME'
rs <- dbSendQuery(db, qry)
friends <- dbFetch(rs, n=-1)
dbClearResult(rs)
```

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

```
head(friends)
```

```
##   ID FIRST_NAME LAST_NAME
## 1  2       Alex  Zakharov
## 2  1    Claudia    Schaab
## 3  4     Howard      Pein
## 4  3       Igor Vaysiberg
## 5  5     Monish     Darda
```

Load the MOVIES table:

```
qry <- 'SELECT * FROM MOVIES ORDER BY TITLE'
rs <- dbSendQuery(db, qry)
movies <- dbFetch(rs, n=-1)
dbClearResult(rs)
```

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

```
head(movies)
```

```
##   ID             TITLE
## 1 14     A QUIET PLACE
## 2 13    A STAR IS BORN
## 3 10 BOHEMIAN RHAPSODY
## 4  9              CODA
## 5 11 CRAZY RICH ASIANS
## 6  5     DON'T LOOK UP
```

7 Load Ratings from Database

Finally, load the RATINGS. Since the database schema is normalized, join FRIENDS, MOVIES, and RATINGS tables to load aggregate ratings data.

```
qry <- 'SELECT f.FIRST_NAME, f.LAST_NAME, m.TITLE, r.RATING
            FROM FRIENDS f, MOVIES m, RATINGS r
            WHERE r.FRIEND_ID = f.ID AND r.MOVIE_ID = m.ID
            ORDER BY f.FIRST_NAME, f.LAST_NAME, m.TITLE'
rs <- dbSendQuery(db, qry)
ratings <- dbFetch(rs, n=-1)
dbClearResult(rs)
```

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

```
head(ratings)
```

```
##   FIRST_NAME LAST_NAME             TITLE RATING
## 1       Alex  Zakharov     A QUIET PLACE      3
## 2       Alex  Zakharov    A STAR IS BORN      4
## 3       Alex  Zakharov BOHEMIAN RHAPSODY      4
## 4       Alex  Zakharov              CODA      3
## 5       Alex  Zakharov CRAZY RICH ASIANS      4
## 6       Alex  Zakharov     DON'T LOOK UP      3
```

Enumerations in MySQL are stored as characters. This is not the most convenient representation because we may want to do numeric analysis on ratings, e.g. average movie rating. Convert RATING from character to integer. Notice that RATING column has Min. : 0 (missing rating).

```
ratings <- ratings %>% mutate(RATING = as.integer(RATING))
summary(ratings)
```

```
##   FIRST_NAME         LAST_NAME            TITLE               RATING
##  Length:70          Length:70          Length:70          Min.   :0.000
##  Class :character   Class :character   Class :character   1st Qu.:3.000
##  Mode  :character   Mode  :character   Mode  :character   Median :4.000
##                                                           Mean   :3.643
##                                                           3rd Qu.:5.000
##                                                           Max.   :5.000
```

A rating of 0 is assigned to unrated movies. Filter out data for unrated movies. As expected, notice that RATING column now has Min. : 1 (missing rating)

```
ratings <- ratings %>% filter(RATING != 0)
summary(ratings)
```

```
##   FIRST_NAME         LAST_NAME            TITLE               RATING
##  Length:66          Length:66          Length:66          Min.   :1.000
##  Class :character   Class :character   Class :character   1st Qu.:3.000
##  Mode  :character   Mode  :character   Mode  :character   Median :4.000
##                                                           Mean   :3.864
##                                                           3rd Qu.:5.000
##                                                           Max.   :5.000
```

Be a good citizen and close the database connection:

```
dbDisconnect(db)
```

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