Accessing Databases with R

Sean Hellingman ©

Data Visualization and Manipulation through Scripting (ADSC1010)

shellingman@tru.ca

Fall 2024

THOMPSON RIVERS UNIVERSITY

Topics

2. Introduction
3. Relational Databases
4. Accessing a Database from R
5. Working With the Data
6. Joining Data
7. Process
8. Exercises and References

Introduction

- There are R packages that allows users to connect to and query a database directly.
- This allows you to use R syntax and data structures to work with databases.
- We can use the *dbplyr* package to help with this.

Databases

- Simple data sets are easily saved and accessed as .csv files.
- As the complexity of data grows, you may need multiple frames to organise your data.
- Difficult to store data of different structures in a single sheet.
- The data files may simply be too large to store on a local computer.
- A database (database management system) is used to organise, save, and access information.

Relational Databases

- Relational databases are the most commonly used.
- Data is organised into tables where each row represents a **record** and each column represents a **field** (individual data property of that item).
- Tables are structured like data frames in R.
- Databases may have thousands of tables representing different facets of the data.
- **Relational database management system (RDMS)**

Primary Key

- Relational databases identifies each record in the database table using a **primary key**.
- In each table, one field (column) is designated as the primary key that is unique to each row.
- Primary keys are unique *identifiers* for each observation in the data.
- As data within a database can change, we cannot use row numbers as a primary key.
- Only one primary key is permitted per table.

Foreign Key

- Each *record* may be associated with another.
- Example: Assume we have table with information about musical artists and another about individual songs.
    - We can connect the two tables based on the names of the musical artists.
- **Foreign keys** allow you to join tables together like you would using the `join()` function in R.
- Provides the *relational* functionality of relational databases.

Comments

- There are many different Structured Query Language (SQL) developers.
- SQLite is the simplest database system, generally not used in industrial settings.
- Others free developers: PostgreSQL & MySQL.

Motivation

- You can access your data directly from the database, query the data you want, save that data, and then import it into R or some other statistical software.
- Or, we can use R to directly query a database directly.
- Then we can use familiar R syntax to work with databases.
- More specifically, we will use the functionality of the *dplyr* package to manipulate the data.

R Packages

- Need to use *dplyr*.
- Will need to install and load the *dbplyr* package.
- Will need to install and load the *DBI* package.
  - *The DBI package helps connecting R to database management systems.*
- Install and load the *RSQLite* package.
  - *If you want to access SQLite databases*
- Install and load the *RPostgreSQL* package.
  - *If you want to access Postgres databases*
- Remeber: `if(!require(package_name)) install.packages("package_name") library(package_name)`

Connecting to Databases

- Databases are managed and accessed through an RDMS, which is separate from R.

- We need to “connect” to the database through R.

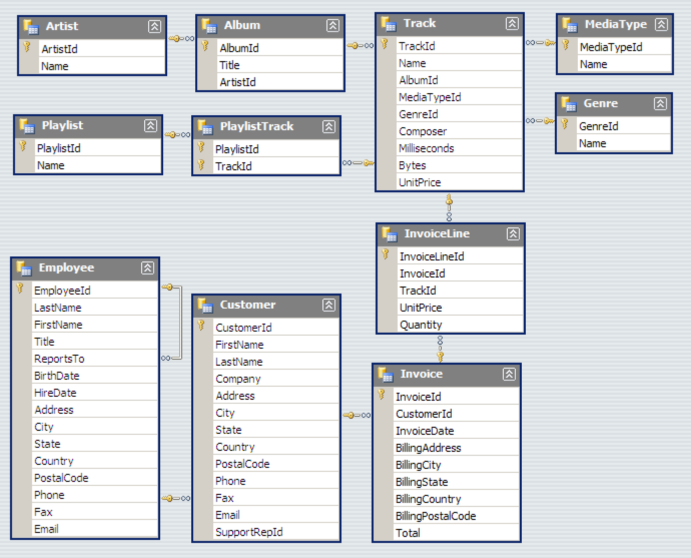
- Use: `db_connection <- dbConnect(SQLite(), dbname = "path/to/database.sqlite")`
  - First argument is the relevant database connection package (RSQLite).
  - If database is in working directory, you can just access the database directly.
  - User & password: `user="Username", password="Password123"`

- To disconnect from your database use: `dbDisconnect(db_connection)`

Example 1

- Download the *Chinook_Sqlite.sqlite* database from moodle and save it in an appropriate location.
- Use the `dbConnect()` function to access the *Chinook_Sqlite.sqlite* database from R.

Chinook Database



Artist
ArtistId
Name

Album
AlbumId
Title
ArtistId

Track
TrackId
Name
AlbumId
MediaTypeId
GenreId
Composer
Milliseconds
Bytes
UnitPrice

MediaType
MediaTypeId
Name

Genre
GenreId
Name

Playlist
PlaylistId
Name

PlaylistTrack
PlaylistId
TrackId

InvoiceLine
InvoiceLineId
InvoiceId
TrackId
UnitPrice
Quantity

Employee
EmployeeId
LastName
FirstName
Title
ReportsTo
BirthDate
HireDate
Address
City
State
Country
PostalCode
Phone
Fax
Email

Customer
CustomerId
FirstName
LastName
Company
Address
City
State
Country
PostalCode
Phone
Fax
Email
SupportRepId

Invoice
InvoiceId
CustomerId
InvoiceDate
BillingAddress
BillingCity
BillingState
BillingCountry
BillingPostalCode
Total

Table Names

- Once you have accessed the database, you can use the `dbListTables()` function to get a vector of the table names.

- Remember: data come from specific tables within the database.

- Need to create a variable in R that references specific tables.
    - Can use the `tbl()` function to accomplish this.

- If you examine the variable name, it looks *mostly* like a normal data frame.
- But this variable actually comes from a remote source.

Example 2

- Use the `dbListTables()` functions to get a list of tables in the *Chinook* database.
    - You may refer to slide 13 to help you visualise the structure of the database.
- Use the `tbl()` function to create a variable for the *Track* table in R.
- What does the variable look like?

Using *dplyr*

- Once we have created a reference to the table in R, we can apply the *dplyr* functions!
- We can also construct a query using *dplyr* and generate the corresponding SQL query.
  - Use the `show_query()` function.
- Save the query as an object in R and then use the `show_query()` function.

Example 3

- Return the track *Name* and *TrackId* for all U2 Tracks (*Composer)* found in the Track table.
- Generate the corresponding SQL query.
- Use the `data.frame()` to create a data frame of U2 tracks.

`left_join()`

- Looks for matching columns between two data frames (tables).
- Returns a new data frame that is the first (*left*) argument with extra columns from the second (*right*) added on.
- The resulting table is a *merged* table of the two arguments.
- Matching occurs using the `by` argument which takes a vector of column names (strings).
    - Use: `by = join_by("key.1" == "key.2")` if foreign keys have different names.
- Left rows without a match will have `NA` in the right columns.

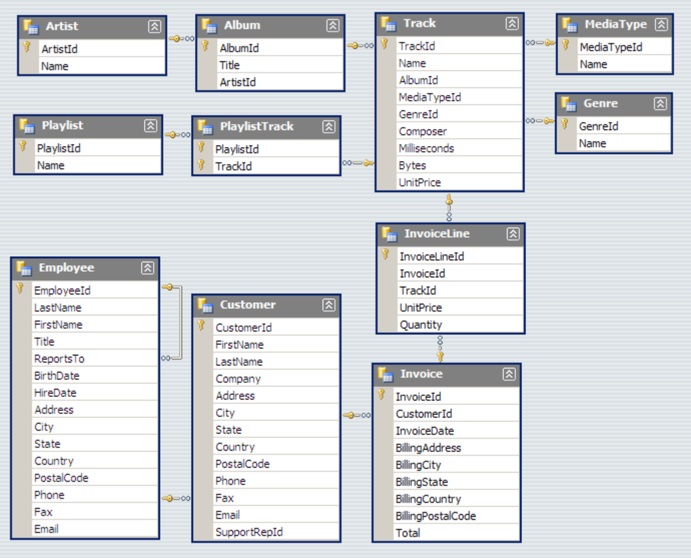
`inner_join()`

- Only rows present in *both* data frames (tables) are returned.
- Returns a new data frame that contains only observations that had matches in both data frames.
- Observations without matches will not be included (no `NA` values).
- The order of the arguments does not matter.

`full_join()`

- All the rows present in *both* data frames (tables) are returned.
- A row for every single observation is returned.
- Observations without matches will have `NA` values in the columns from the other data frame.
- Can lead to very messy data.
- The order of the arguments does not matter.

Chinook Database Again



Artist
- ArtistId
- Name

Album
- AlbumId
- Title
- ArtistId

Track
- TrackId
- Name
- AlbumId
- MediaTypeId
- GenreId
- Composer
- Milliseconds
- Bytes
- UnitPrice

MediaType
- MediaTypeId
- Name

Genre
- GenreId
- Name

Playlist
- PlaylistId
- Name

PlaylistTrack
- PlaylistId
- TrackId

InvoiceLine
- InvoiceLineId
- InvoiceId
- TrackId
- UnitPrice
- Quantity

Employee
- EmployeeId
- LastName
- FirstName
- Title
- ReportsTo
- BirthDate
- HireDate
- Address
- City
- State
- Country
- PostalCode
- Phone
- Fax
- Email

Customer
- CustomerId
- FirstName
- LastName
- Company
- Address
- City
- State
- Country
- PostalCode
- Phone
- Fax
- Email
- SupportRepId

Invoice
- InvoiceId
- CustomerId
- InvoiceDate
- BillingAddress
- BillingCity
- BillingState
- BillingCountry
- BillingPostalCode
- Total

Example 4

- Use the `tbl()` function to create a variable for the *Album* table in R.
- Include the album information to each of the tracks in the Track table (join).
- Use the `data.frame()` to create a data frame of your resulting table.

Process

1. Create a connection to an RDMS (SQLite):
   - `db_connection <- dbConnect(SQLite(), dbname = "path/to/database.sqlite")`
2. Access a specific table within the database:
   - `some_table <- tbl(db_connection, "TABLE_NAME")`
3. Construct a query of the table using *dplyr* syntax:
   - `db_query <- some_table %>% filter(some_column == some_value)`
4. Execute your query to return the data (bring it into R):
   - `results <- collect(db_query)`
     OR
   - `results <- data.frame(db_query)`
5. Disconnect from the database when you are finished:
   - `dbDisconnect(db_connection)`

Exercise 1

- Use the `dbConnect()` function to access the *Chinook_Sqlite.sqlite* database from R.
- Query all customers whose *Country* is Canada.
- Join the Canadian based customers with their respective invoices (Invoice).
    - You could have multiple rows for the same customer, if they made more than one purchase.
- Join the InvoiceLine for all of the Canadian based customers.
- *You can choose which order you would like to join the tables.*
- Import your data into R and disconnect from the database.

Exercise 2

- Using the Chinook database:
  - What is the title of the album with AlbumId 45?
  - Find the name and length (in seconds) of all tracks that have length between 55 and 75 seconds.
  - Provide a query showing a unique/distinct list of billing countries from the Invoice table.
  - Find the CustomerID of the customer(s) who made the most purchases.

References & Resources

1. Michael Freeman, Joel Ross, *Programming Skills for Data Science: Start Writing Code to Wrangle, Analyze, and Visualize Data with R*, 2019, ISBN-13: 978-0-13-513310-1

- https://dbi.r-dbi.org/
- https://github.com/lerocha/chinook-database
- https://dplyr.tidyverse.org/