# 13. Introduction to Relational Databases

Principles of Data Science with R

Dr. Uma Ravat

PSTAT 10

# Announcement

1. No homework due this Wednesday.
2. Worksheet 13, is due 30 mins after section
3. Exam grades will be released at the end of the lecture
4. Exam viewing
   - Ethan, Jason, Cyrus's section will view exam during HW clinic on Wednesday 6pm - 8pm.
   - Other sections, stop by your TA office hour this week
5. Your TA will review questions missed by most in section this week.

# Next we will see...

- Intro to Databases
- Database tools for R
  - the R packages `RSQLite`, `sqldf`, `DBI`
  - the database on disk/file `Chinook_Sqlite.sqlite`
- SQL (Structured Query Language)
- The relational data model
  - Primary keys
  - Foreign keys
  - Integrity constraints

# data frames vs Databases

Data frames in R are tables in database lingo

| R jargon | Database jargon |
|---|---|
| column | field |
| row | record |
| data frame | table |
| types of the columns | table schema |
| collection of data frames | database |

# What is a database?

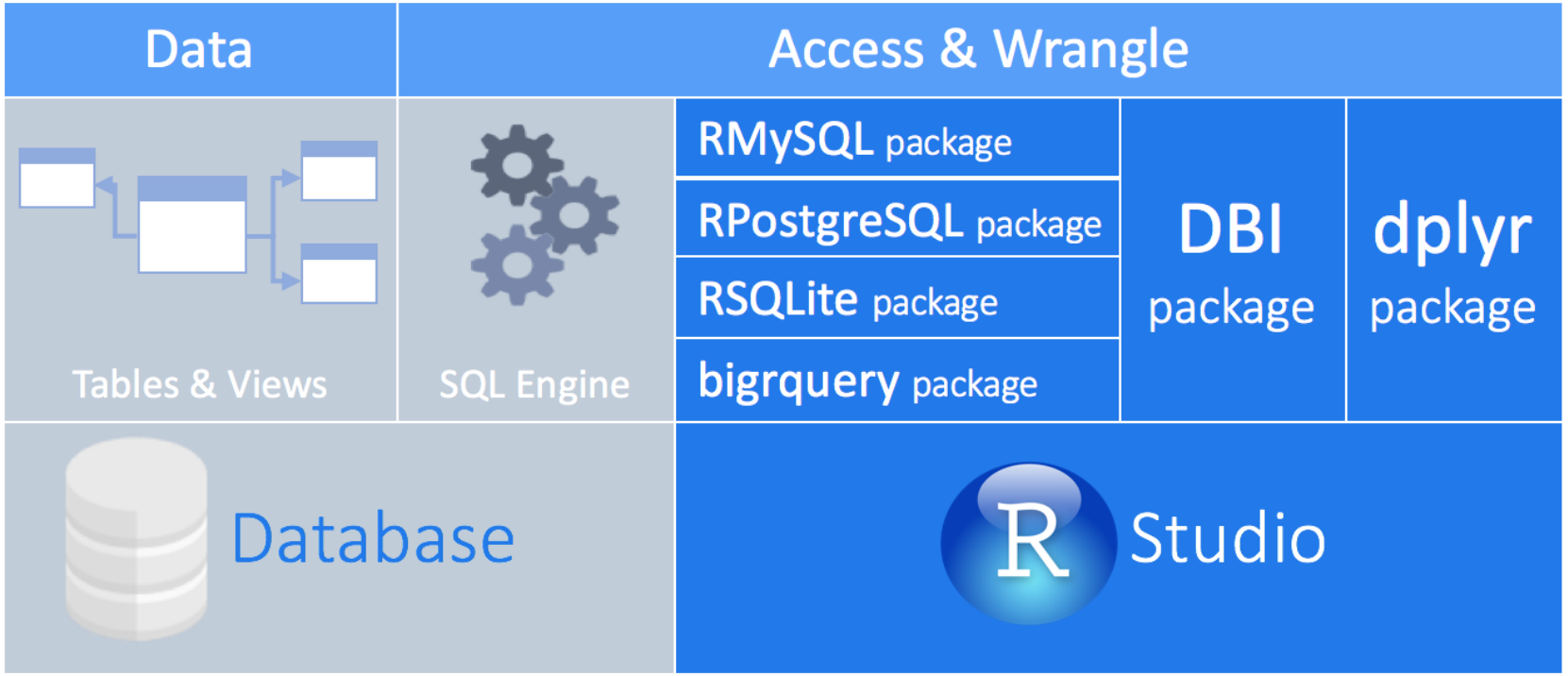
First some terms:

- A **field** is a variable/quantity of interest
- A **record** is a collection of **fields**
- A **table** is a collection of records which all have the same fields (with different values)
- A **database** is a collection of tables

# Why do we need database software?

- **Size**
  - R keeps its data frames in memory
  - Industrial databases are much bigger, need to store out of memory and bring into memory only required subsets
  - Must work with selected subsets
- **Speed**
  - Relational model published in 70's by E. F. Codd at IBM labs in San Jose, CA.
  - Smart people have worked very hard making relational databases efficient
  - 2014 Turing award winner Michael Stonebraker
- **Concurrency**
  - Many users access the same database simultaneously
  - Potential for trouble (two users want to change the same record at once)
  - Database software takes care of this issue.

# Open Source Databases

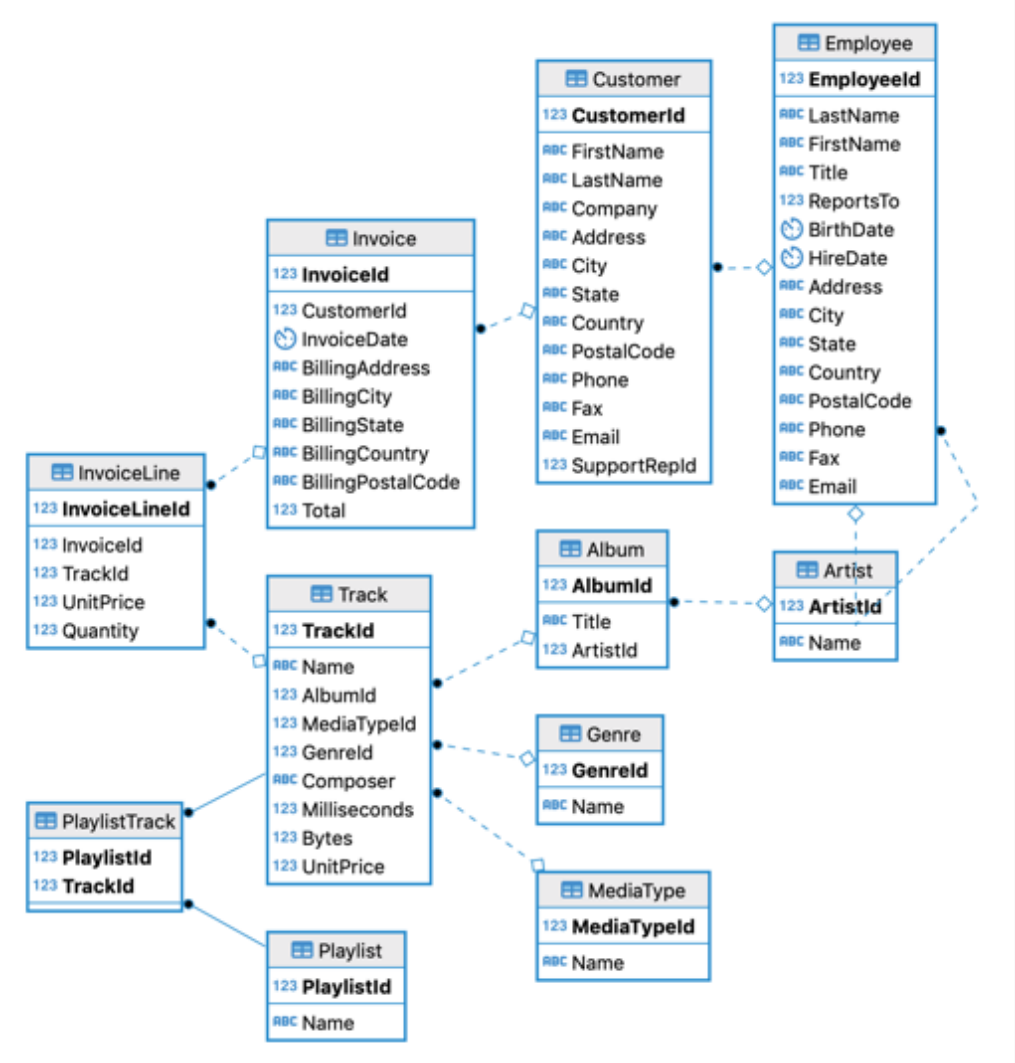


# Client-server model and SQL

- Databases live on a **server**, which manages them
- Users interact with the server through a **client** program
- Lets multiple users access the same database simultaneously
- **SQL** (**structured query language**) is the standard for database software
    - many different implementations of SQL, each with unique features. Each implementation is called a *Relational Database Management System* (RDBMS).
- Most basic actions with databases are **SQL queries**, like row/column selections, inserts, updates and deletes
- **SQLite** is a popular RDBMS designed for simple applications (mobile apps)
    - is a simpler, file-based system that we will use via `RSQLite` package

# A relational data model

A model used to represent data and the relationships between data items



# The Chinook DB

We will use the Chinook DB, a relational database.

This DB represents a digital media store, including tables for artists, albums, media tracks, invoices, and customers.

# Connecting R to SQLite

SQL is its own language, independent of R. For simplicity, we're going to learn how to run SQL queries through R

First, we need to install the packages `DBI`, `RSQLite`, then we load them into our R session with `library()`

Also, we need a database file available at `YT01/data/Chinook_Sqlite.sqlite` for this lecture

# Connecting to the database in R

```
library(DBI)
library(RSQLite)
drv = dbDriver("SQLite")
chinook_db = dbConnect(drv,
                       dbname="data/Chinook_Sqlite.sqlite")
```

The object `chinook_db` is now a persistent connection to the `Chinook_Sqlite.sqlite` database on disk.

# Listing what's available

The data in a relational database is stored in relations, aka **tables**:

```
# List tables in our database
dbListTables(chinook_db)
```

```
##  [1] "Album"         "Artist"        "Customer"      "Em
##  [5] "Genre"         "Invoice"       "InvoiceLine"   "Me
##  [9] "Playlist"      "PlaylistTrack" "Track"
```

Each table has rows of tuples, aka **records**, and columns of attributes, aka **fields**.

```
#List fields in Customer table
dbListFields(chinook_db, "Customer")
```

```
##  [1] "CustomerId"   "FirstName"    "LastName"     "Compa
##  [6] "City"         "State"        "Country"      "Posta
## [11] "Fax"          "Email"        "SupportRepId"
```

# Importing a table as a data frame

```
customer = dbReadTable(chinook_db, "Customer")
class(customer)
```

```
## [1] "data.frame"
```

```
dim(customer)
```

```
## [1] 60 13
```

Now we could go on and perform R operations on `customer`, since it's a data frame

We'll use this route primarily to check our work in SQL; in general, should try to do as much in SQL as possible, since it's more efficient and can be simpler

# SELECT

Main tool in the SQL language: `SELECT`, which allows you to perform queries on a particular table in a database. It has the form:

```
SELECT columns
  FROM table
  WHERE condition
  GROUP BY columns
  HAVING condition
  ORDER BY column [ASC | DESC]
  LIMIT offset, count;
```

`WHERE`, `GROUP BY`, `HAVING`, `ORDER BY`, `LIMIT` are all optional

# Example

Pick out five columns from the table “Customer”, and only look at the first 6 rows:

```
dbGetQuery(chinook_db,
    "select CustomerId, FirstName, LastName, City, Country
     from Customer
     limit 6")
```

```
##   CustomerId FirstName    LastName                City
## 1          1      Luís   Gonçalves São José dos Campos
## 2          2    Leonie      Köhler           Stuttgart
## 3          3  François    Tremblay            Montréal
## 4          4     Bjørn      Hansen                Oslo
## 5          5 František Wichterlová              Prague
## 6          6    Helena        Holý              Prague
```

## To replicate this simple command on the imported data frame:

```
customer[1:6,
         c("CustomerId", "FirstName", "LastName",
           "City", "Country")]
```

```
##   CustomerId FirstName    LastName                City
## 1          1      Luís   Gonçalves São José dos Campos
## 2          2    Leonie      Köhler           Stuttgart
## 3          3  François    Tremblay            Montréal
## 4          4     Bjørn      Hansen                Oslo
## 5          5 František Wichterlová              Prague
## 6          6    Helena        Holý              Prague
```

(Note: this was simply to check our understanding, and we wouldn't actually want to do this on a large database, since it'd be inefficient to first read into an R data frame, and then call R commands)

# ORDER BY

We can use the `ORDER BY` option in `SELECT` to specify an ordering for the rows

Default is ascending order; add `DESC` for descending

```
dbGetQuery(chinook_db,
           "select CustomerId, FirstName, LastName,
               City, Country
                   from Customer
                   ORDER BY FirstName DESC
                   limit 10")
```

```
##    CustomerId FirstName   LastName           City
## 1          42     Wyatt     Girard       Bordeaux
## 2          25    Victor    Stevens        Madison
## 3          19       Tim      Goyer      Cupertino
## 4          44     Terhi Hämäläinen       Helsinki
```

# Field metadata

Unlike a data frame, there is extra information in a database table that expresses relational information between tables.

```
dbGetQuery(chinook_db, "pragma table_info(Customer)")
```

```
##    cid          name         type notnull dflt_value pk
## 1    0    CustomerId      INTEGER        1         NA  1
## 2    1     FirstName NVARCHAR(40)        1         NA  0
## 3    2      LastName NVARCHAR(20)        1         NA  0
## 4    3       Company NVARCHAR(80)        0         NA  0
## 5    4       Address NVARCHAR(70)        0         NA  0
## 6    5          City NVARCHAR(40)        0         NA  0
## 7    6         State NVARCHAR(40)        0         NA  0
## 8    7       Country NVARCHAR(40)        0         NA  0
## 9    8    PostalCode NVARCHAR(10)        0         NA  0
## 10   9         Phone NVARCHAR(24)        0         NA  0
```

# Primary key

The **primary key** is a *unique identifier* of the rows in a table. Two rows cannot have the same primary key:

```
dbGetQuery(chinook_db,
    "select CustomerId, FirstName, LastName,
        City, Country
        from Customer
        limit 2")
```

```
##   CustomerId FirstName  LastName                 City Cou
## 1          1      Luís Gonçalves São José dos Campos  Br
## 2          2    Leonie    Köhler           Stuttgart Ger
```

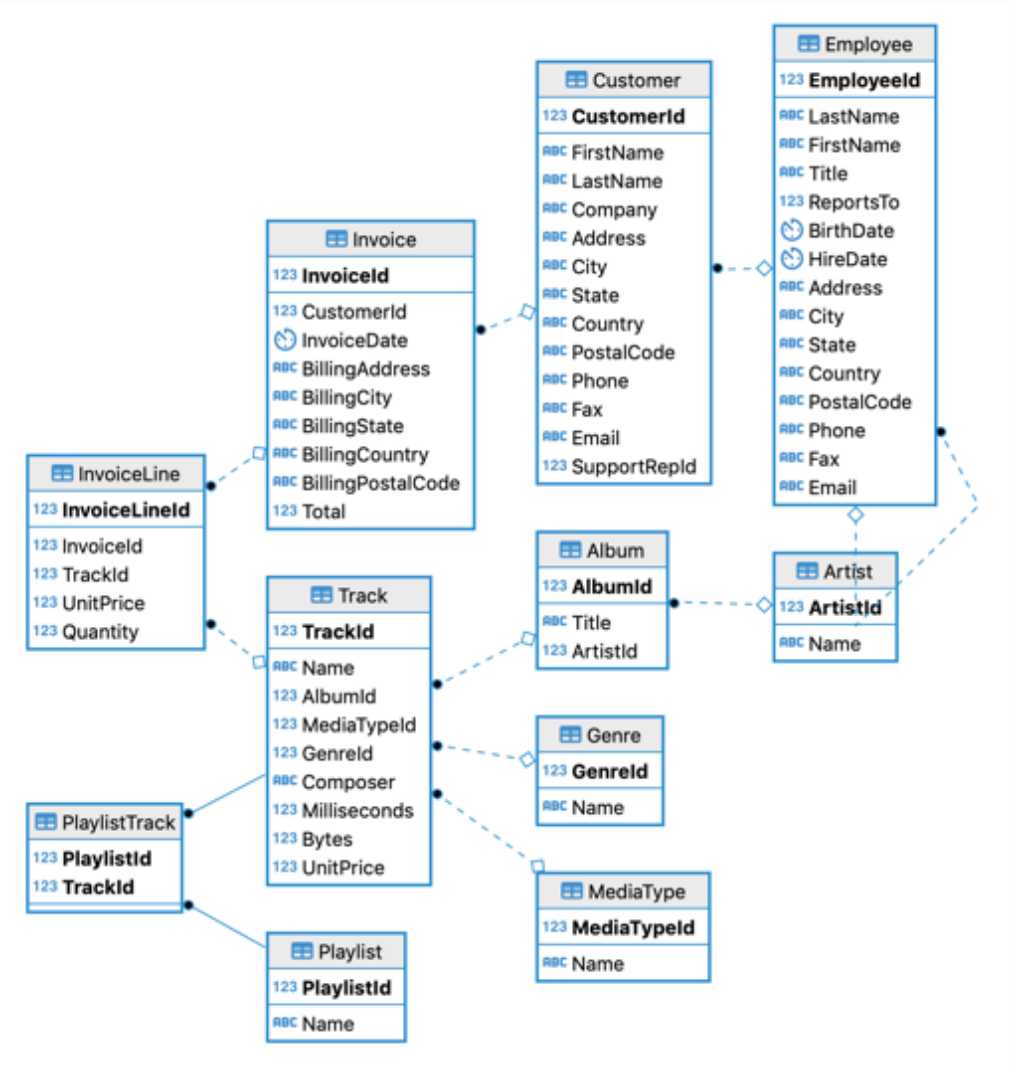
```
dbExecute(chinook_db,
    paste("insert into customer",
        "(CustomerId, FirstName, LastName, Email)",
        "values",
        "(1, 'Luis','Armstrong','LuisArmstrong@pstat.ucsb.e
```

```
## Error: UNIQUE constraint failed: Customer.CustomerId
```

CustomerId is the **primary key** and must be unique.

# Primary key

Tables are not required to have a primary key, but most do. All the tables in Chinook have a primary key.



# Foreign keys

The relationship between tables is expressed by primary keys and **foreign keys**. Remember we are working with a relational database, following a relational data model.

```
dbGetQuery(chinook_db,
           "pragma foreign_key_list(customer)")
```

```
##   id seq    table          from         to on_update on_
## 1  0   0 Employee SupportRepId EmployeeId NO ACTION NO A
```


A foreign key field *points to* the primary key of another table.

# Interpretation of foreign key

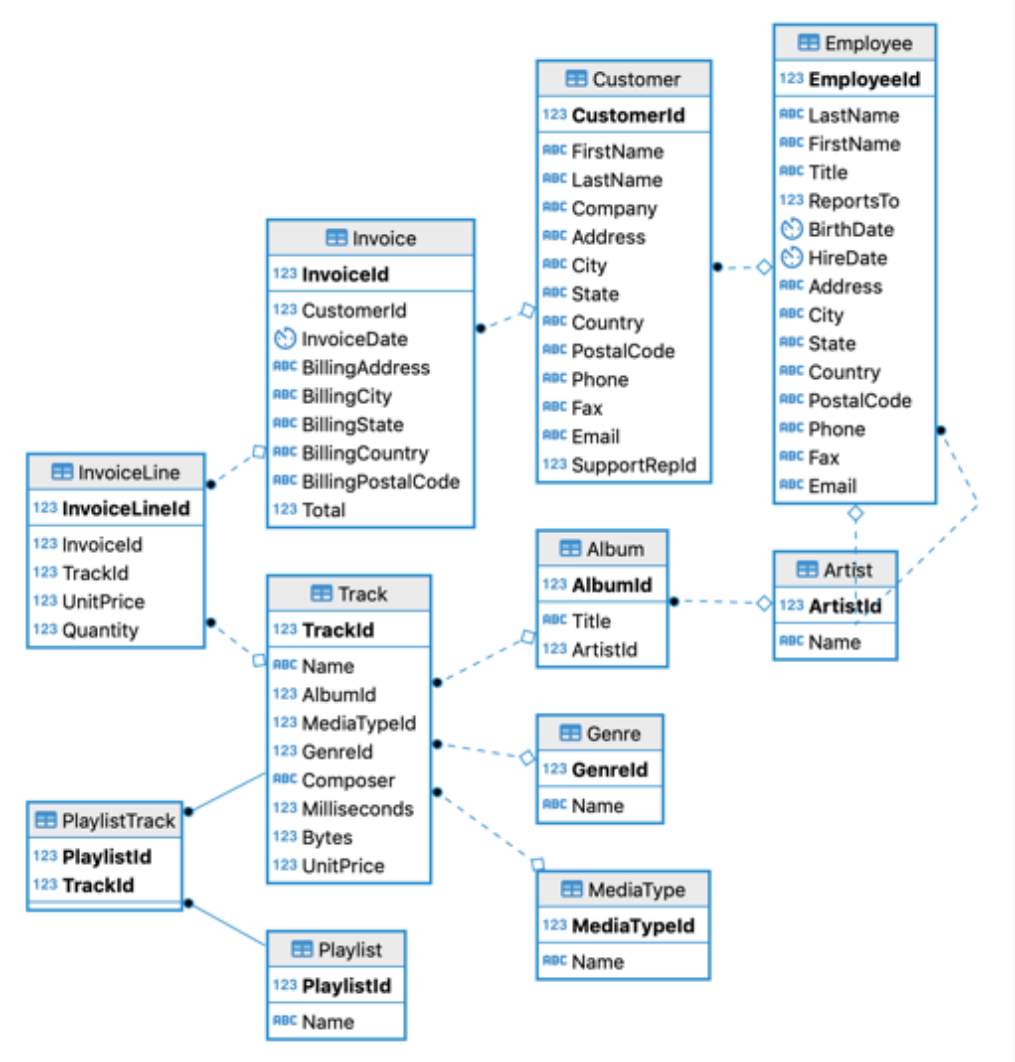

- Each customer in Customer table can be assigned a support representative
- The support rep is an employee at the store and therefore has a unique id, EmployeeId
- This unique id,EmployeeId , is the primary key of the employee table



Thus real-world relationship is encoded by the relational model

# Foreign keys

What are some other foreign keys in Chinook?



# Foreign keys

Foreign keys must either point to an existing value or be NULL.

```
dbGetQuery(chinook_db, "select max(EmployeeId) from employe
```

```
##   max(EmployeeId)
## 1               8
```

```
dbExecute(chinook_db,
    "insert into customer
    (CustomerId, FirstName, LastName, Email, SupportRepId)
    values
    (888, 'Luis', 'Armstrong',
    'luisArmstrong@pstat.ucsb.edu', 9999)")
```

```
## Error: UNIQUE constraint failed: Customer.CustomerId
```

# Integrity Constraints

We have seen two examples of *integrity constraints*: - Primary keys must be unique (and not NULL) - Foreign keys must reference existing primary keys or be NULL

These constraints enforce the *integrity* of a database; no bad data or corrupted relationships.

**Keys help maintain the integrity of the data**

# Database Schema

The **schema** of a database describes its *structure*: - Names of all the tables - Names of all fields in each table - Primary key/foreign key relationships between tables - Other metadata (data types of each field in each table, . . . )

Basically everything other than the actual data itself.

We have been looking at parts of the schema with the `pragma` keyword.

```
dbGetQuery(chinook_db, "pragma table_info(customer)")
```

```
##   cid          name         type notnull dflt_value pk
## 1   0    CustomerId      INTEGER       1         NA  1
## 2   1     FirstName NVARCHAR(40)       1         NA  0
## 3   2      LastName NVARCHAR(20)       1         NA  0
## 4   3       Company NVARCHAR(80)       0         NA  0
```

# Database tools for R

- the R packages `RSQLite`, `sqldf`, `DBI`
- the database `Chinook_Sqlite.sqlite`

```
dbExecute(chinook_db, "pragma foreign_keys = on")
```

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

```
# Required for foreign-key support
```

# SQL

We opened a connection as follows:

```
chinook_db <- dbConnect(SQLite(),
                        "Chinook_Sqlite.sqlite")
```

After the end of a session, it is good practice to explicitly close your connection.

```
dbDisconnect(chinook_db)
```

Indeed the connection is closed. Try reading some data:

```
dbGetQuery(chinook_db,
           "select CustomerId, FirstName, LastName from Cus
```

```
## Error: Invalid or closed connection
```

Next time, introduction to SQL queries.

# We saw...

- Databases are used to store massive amounts of data that cannot fit in memory.
- SQL is the language used to manipulate relational databases
- SQLite is the SQL implementation we will use, provided by the `RSQLite` package.