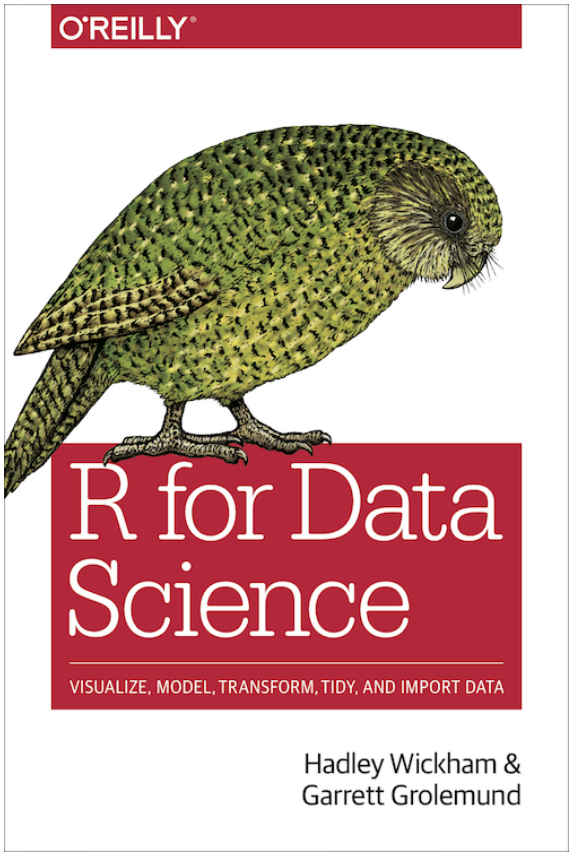
# Socio-Informatics 348

## Import
## Databases

Dr Lisa Martin

Department of Information Science
Stellenbosch University

# Today's Reading




*R for Data Science, Chapter 21*

# Database Basics

- Definition: databases are collections of tables, stored on disk; can be much larger than in-memory data frames.
- Key differences from data frames:
  - Size & storage (on disk vs memory)
  - Indexes (for fast lookups)
  - DBMS type: row-oriented vs column-oriented storage.
- Types of DBMS:
  - Client-server (e.g. PostgreSQL, MySQL)
  - Cloud DBMS (e.g. BigQuery, RedShift)
  - In-process DBMS (e.g. SQLite, duckdb) – run entirely on your computer.

# Connecting to a Database

- DBI (database interface) provides generic functions to connect, write, query, etc.
- Use DBMS-specific backend packages
- e.g. RPostgres for PostgreSQL, RMariaDB for MariaDB, etc.

```
con <- DBI::dbConnect(
  RMariaDB::MariaDB(),
  username = "foo"
)
con <- DBI::dbConnect(
  RPostgres::Postgres(),
  hostname = "databases.mycompany.com",
  port = 1234
)
```

# Connecting to a Database

- For the purposes of the lecture, we will use duckdb, an in-process DBMS.
- Install and load the `duckdb` package.
- Create a temp database in your working directory.

```
library(DBI)
library(dbplyr)
library(tidyverse)
```

```
con <- DBI::dbConnect(duckdb::duckdb())
```

```
con <- DBI::dbConnect(duckdb::duckdb(), dbdir = "duckdb")
```

# Loading & Writing Data

- Use `dbWriteTable()` to put data frames into database tables.
- Requires: a DBI connection object, table name, data frame.

```
dbWriteTable(con, "mpg", ggplot2::mpg)
dbWriteTable(con, "diamonds", ggplot2::diamonds)
```

# DBI Basics

- Inspect tables: `dbListTables()`, `dbReadTable()` to list and retrieve contents from tables (Note: `as_tibble()`).

```
dbListTables(con)
#> [1] "diamonds" "mpg"

con |>
  dbReadTable("diamonds") |>
  as_tibble()
#> # A tibble: 53,940 × 10
#>   carat cut       color clarity depth table price     x     y     z
#>   <dbl> <fct>     <fct> <fct>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
#> 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
#> 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
#> 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
#> 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
#> 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
#> 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
#> # i 53,934 more rows
```

# DBI Basics

- Use a query to retrieve specific data: `dbGetQuery()`.
- Requires: a DBI connection object, SQL query string.

```
sql <- "
  SELECT carat, cut, clarity, color, price
  FROM diamonds
  WHERE price > 15000
"
as_tibble(dbGetQuery(con, sql))
#> # A tibble: 1,655 × 5
#>   carat cut       clarity color price
#>   <dbl> <fct>     <fct>   <fct> <int>
#> 1  1.54 Premium   VS2     E     15002
#> 2  1.19 Ideal     VVS1    F     15005
#> 3  2.1  Premium   SI1     I     15007
#> 4  1.69 Ideal     SI1     D     15011
#> 5  1.5  Very Good VVS2    G     15013
#> 6  1.73 Very Good VS1     G     15014
#> # ℹ 1,649 more rows
```

# dbplyr Basics

- Concept: write dplyr code; dbplyr translates it into SQL, executes via DBI
- Use `tbl()` to refer to database tables without loading them fully into R.

```
diamonds_db <- tbl(con, "diamonds")
diamonds_db
#> # Source:   table<diamonds> [?? x 10]
#> # Database: DuckDB 1.4.0 [unknown@Linux 6.11.0-1018-azure:R 4.5.1/:memory:]
#>   carat cut       color clarity depth table price     x     y     z
#>   <dbl> <fct>     <fct> <fct>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
#> 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
#> 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
#> 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
#> 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
#> 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
#> 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
#> # i more rows
```

# Note:

- `tbl()` – Creates a lazy reference to a database table. Data stays in the database until you explicitly pull it with `collect()`. Ideal for large tables and tidyverse pipelines.
- `dbReadTable()` – Immediately reads an entire table into R as a data.frame. Best for small tables you want fully in memory.
- `dbGetQuery()` – Executes a custom SQL query and returns the result immediately in R. Useful for filtering, joins, or any query logic before pulling data.

# dbplyr Basics

**Lazy?**

- This means that when you use `tbl()` to reference a table, no data is actually loaded into R until you explicitly call `collect()`.
- This object represents a database query because it prints the DBMS name at the top.

```
big_diamonds_db <- diamonds_db |>
  filter(price > 15000) |>
  select(carat:clarity, price)

big_diamonds_db
#> # Source:   SQL [?? x 5]
#> # Database: DuckDB 1.4.0 [unknown@Linux 6.11.0-1018-azure:R 4.5.1/:memory:]
#>   carat cut       color clarity price
#>   <dbl> <fct>     <fct> <fct>   <int>
#> 1  1.54 Premium   E     VS2     15002
#> 2  1.19 Ideal     F     VVS1    15005
#> 3  2.1  Premium   I     SI1     15007
#> 4  1.69 Ideal     D     SI1     15011
#> 5  1.5  Very Good G     VVS2    15013
```

# dbplyr Basics

**Lazy?**

- You can see the SQL code generated by the dplyr function `show_query()`.
- This is a great way to learn SQL!

```
big_diamonds_db |>
  show_query()
#> <SQL>
#> SELECT carat, cut, color, clarity, price
#> FROM diamonds
#> WHERE (price > 15000.0)
```

# dbplyr Basics

**Lazy?**

To get all the data back into R, you call `collect()`. Behind the scenes, this generates the SQL, calls `dbGetQuery()` to get the data, then turns the result into a tibble.

```
big_diamonds <- big_diamonds_db |>
  collect()
big_diamonds
#> # A tibble: 1,655 × 5
#>   carat cut       color clarity price
#>   <dbl> <fct>     <fct> <fct>   <int>
#> 1  1.54 Premium   E     VS2     15002
#> 2  1.19 Ideal     F     VVS1    15005
#> 3  2.1  Premium   I     SI1     15007
#> 4  1.69 Ideal     D     SI1     15011
#> 5  1.5  Very Good G     VVS2    15013
#> 6  1.73 Very Good G     VS1     15014
#> # ℹ 1,649 more rows
```

# SQL Basics

- Use `dbplyr` to get some datasets that we already know.

```
dbplyr::copy_nycflights13(con)
#> Creating table: airlines
#> Creating table: airports
#> Creating table: flights
#> Creating table: planes
#> Creating table: weather
flights <- tbl(con, "flights")
planes <- tbl(con, "planes")
```

- Five important classes that make up a query: `SELECT`, `FROM`, `WHERE`, `GROUP BY`, `ORDER BY`.

# SQL Basics

## SELECT

```
planes |>
  select(tailnum, type, manufacturer, model, year) |>
  show_query()
#> <SQL>
#> SELECT tailnum, "type", manufacturer, model, "year"
#> FROM planes

planes |>
  select(tailnum, type, manufacturer, model, year) |>
  rename(year_built = year) |>
  show_query()
#> <SQL>
#> SELECT tailnum, "type", manufacturer, model, "year" AS year_built
#> FROM planes
```

# SQL Basics

## SELECT

```
planes |>
  select(tailnum, type, manufacturer, model, year) |>
  relocate(manufacturer, model, .before = type) |>
  show_query()
#> <SQL>
#> SELECT tailnum, manufacturer, model, "type", "year"
#> FROM planes
```

```
flights |>
  mutate(
    speed = distance / (air_time / 60)
  ) |>
  show_query()
#> <SQL>
#> SELECT flights.*, distance / (air_time / 60.0) AS speed
#> FROM flights
```

# SQL Basics

## GROUP BY

```
diamonds_db |>
  group_by(cut) |>
  summarize(
    n = n(),
    avg_price = mean(price, na.rm = TRUE)
  ) |>
  show_query()
#> <SQL>
#> SELECT cut, COUNT(*) AS n, AVG(price) AS avg_price
#> FROM diamonds
#> GROUP BY cut
```

# SQL Basics

WHERE ... OR... AND

```
flights |>
  filter(dest == "IAH" | dest == "HOU") |>
  show_query()
#> <SQL>
#> SELECT flights.*
#> FROM flights
#> WHERE (dest = 'IAH' OR dest = 'HOU')

flights |>
  filter(arr_delay > 0 & arr_delay < 20) |>
  show_query()
#> <SQL>
#> SELECT flights.*
#> FROM flights
#> WHERE (arr_delay > 0.0 AND arr_delay < 20.0)
```

# SQL Basics

## WHERE... IN

```
flights |>
  filter(dest %in% c("IAH", "HOU")) |>
  show_query()
#> <SQL>
#> SELECT flights.*
#> FROM flights
#> WHERE (dest IN ('IAH', 'HOU'))
```

# SQL Basics

## ORDER BY

```
flights |>
  arrange(year, month, day, desc(dep_delay)) |>
  show_query()
#> <SQL>
#> SELECT flights.*
#> FROM flights
#> ORDER BY "year", "month", "day", dep_delay DESC
```

# SQL Sub-Queries

- Subqueries are queries nested inside other queries.
- Useful for breaking complex queries into simpler parts.

```
flights |>
  mutate(
    year1 = year + 1,
    year2 = year1 + 1
  ) |>
  show_query()
#> <SQL>
#> SELECT q01.*, year1 + 1.0 AS year2
#> FROM (
#>   SELECT flights.*, "year" + 1.0 AS year1
#>   FROM flights
#> ) q01
```

# Joins

```
SELECT flights.*, "type", manufacturer, model, engines, seats, speed
FROM flights
INNER JOIN planes ON (flights.tailnum = planes.tailnum)

SELECT flights.*, "type", manufacturer, model, engines, seats, speed
FROM flights
RIGHT JOIN planes ON (flights.tailnum = planes.tailnum)

SELECT flights.*, "type", manufacturer, model, engines, seats, speed
FROM flights
FULL JOIN planes ON (flights.tailnum = planes.tailnum)
```

# More on dbplyr

Visit the dbplyr website:
https://dbplyr.tidyverse.org/reference/