Databases and SQL

Statistical Computing & Programming

Shawn Santo

Supplementary materials

Full video lecture available in Zoom Cloud Recordings

Additional resources

- `dbplyr` vignette

Databases

A **database** is a collection of data typically stored in a computer system. It is controlled by a **database management system (DBMS)**. There may be applications associated with them, such as an API.

Types of DBMS: MySQL, Microsoft Access, Microsoft SQL Server, FileMaker Pro, Oracle Database, and dBASE.

Types of databases: Relational, object-oriented, distributed, NoSQL, graph, and more.

DBMS benefits

- Lower storage and retrieval costs

- Easy data access

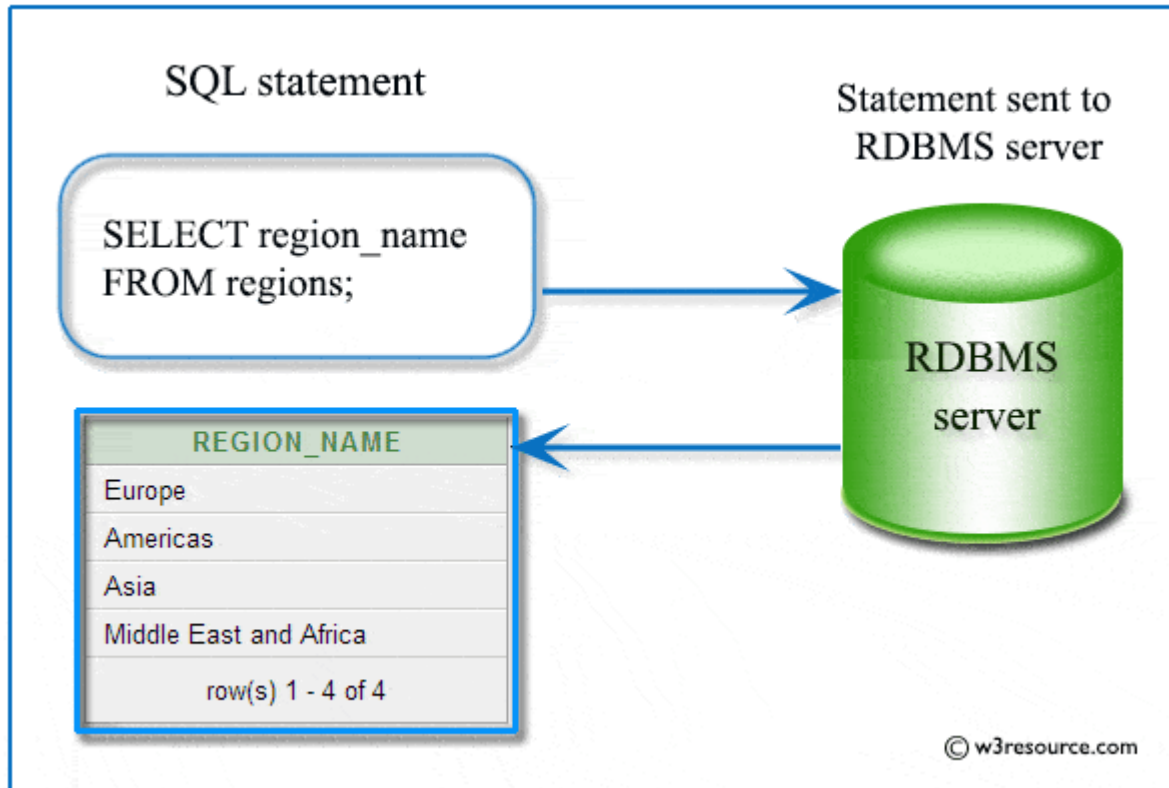- Backup and recovery

- Data consistency

Relational database management system

- A system that governs a relational database, where data is identified and accessed in relation to other data in the database.

- Relational databases generally organize data into tables comprised of **fields** and **records**.

- Many relational database management systems (RDBMS) use SQL to access data. More on SQL in the next slide.

SQL

- SQL stands for Structured Query Language.

- It is an American National Standards Institute standard computer language for accessing and manipulating RDBMS.

- There are different versions of SQL, but to be compliant with the American National Standards Institute the version must support the key query verbs (functions).

Big picture

SQL statement

SELECT region_name
FROM regions;

REGION_NAME
Europe
Americas
Asia
Middle East and Africa
row(s) 1 - 4 of 4

Statement sent to
RDBMS server

RDBMS
server

© w3resource.com

Source: https://www.w3resource.com/sql/tutorials.php

Translation to SQL

Package `dbplyr`

Package `dbplyr` allows you to query a database by automatically generating SQL queries. We'll use it as a starting point to see the connection between `dplyr` verbs (functions) and `SQL` verbs before we transition using SQL.

To get started, load the packages.

```
library(dplyr)
library(dbplyr)
library(DBI)
```

We'll use data from `nycflights13::airports` to create a table in a temporary in-memory database.

Creating an in-memory database

We'll create an in-memory SQLite database and copy the airports tibble as a table into the database.

```
con <- dbConnect(RSQLite::SQLite(), dbname = ":memory:")
```

```
copy_to(con, df = nycflights13::airports, name = "airports")
dbListTables(con)
```

```
#> [1] "airports"    "sqlite_stat1" "sqlite_stat4"
```

Retrieve a single table from our in-memory database.

```
airports_db <- tbl(con, "airports")
```

```
airports_db
```

```
#> # Source:    table<airports> [?? x 8]
#> # Database: sqlite 3.33.0 [:memory:]
#>    faa    name                        lat    lon    alt    tz dst    tzone
#>    <chr> <chr>                       <dbl>  <dbl> <dbl> <dbl> <chr>  <chr>
#>  1 04G    Lansdowne Airport          41.1  -80.6  1044    -5 A      America/New_Yo…
#>  2 06A    Moton Field Municipal A…   32.5  -85.7   264    -6 A      America/Chicago
#>  3 06C    Schaumburg Regional        42.0  -88.1   801    -6 A      America/Chicago
#>  4 06N    Randall Airport            41.4  -74.4   523    -5 A      America/New_Yo…
#>  5 09J    Jekyll Island Airport      31.1  -81.4    11    -5 A      America/New_Yo…
#>  6 0A9    Elizabethton Municipal …   36.4  -82.2  1593    -5 A      America/New_Yo…
#>  7 0G6    Williams County Airport    41.5  -84.5   730    -5 A      America/New_Yo…
#>  8 0G7    Finger Lakes Regional A…   42.9  -76.8   492    -5 A      America/New_Yo…
#>  9 0P2    Shoestring Aviation Air…   39.8  -76.6  1000    -5 U      America/New_Yo…
#> 10 0S9    Jefferson County Intl      48.1 -123.    108    -8 A      America/Los_An…
#> # … with more rows
```

What is different when compared to a tibble object?

Example

NYC flights to airports by time zone.

```
airport_timezone <- airports_db %>%
  group_by(tzone) %>%
  summarise(count = n())
```

```
airport_timezone
```

```
#> # Source:    lazy query [?? x 2]
#> # Database: sqlite 3.33.0 [:memory:]
#>    tzone              count
#>    <chr>              <int>
#>  1 <NA>                   3
#>  2 America/Anchorage    239
#>  3 America/Chicago      342
#>  4 America/Denver       119
#>  5 America/Los_Angeles  176
#>  6 America/New_York     519
#>  7 America/Phoenix       38
#>  8 America/Vancouver      2
#>  9 Asia/Chongqing         2
#> 10 Pacific/Honolulu      18
```

Translation to SQL

```
airport_timezone %>%
  show_query()
```

```
#> <SQL>
#> SELECT `tzone`, COUNT(*) AS `count`
#> FROM `airports`
#> GROUP BY `tzone`
```

```
airports_db %>%
  group_by(tzone) %>%
  summarise(count = n())
```

```
#> # Source:    lazy query [?? x 2]
#> # Database: sqlite 3.33.0 [:memory:]
#>    tzone               count
#>    <chr>               <int>
#>  1 <NA>                    3
#>  2 America/Anchorage     239
#>  3 America/Chicago       342
#>  4 America/Denver        119
#>  5 America/Los_Angeles   176
#>  6 America/New_York      519
#>  7 America/Phoenix        38
#>  8 America/Vancouver       2
#>  9 Asia/Chongqing          2
#> 10 Pacific/Honolulu       18
```

What are the `dplyr` translations to SQL?

Exercise

What are the corresponding SQL verbs based on the `dplyr` structure below?

```
airports_db %>%
  filter(lat >= 33.7666, lat <= 36.588,
         lon >= -84.3201, lon <= -75.4129) %>%
  arrange(desc(alt)) %>%
  select(name, alt)
```

Limitations

```
airports_db %>%
  filter(lat >=  33.7666, lat <=  36.588,
         lon >= -84.3201, lon <= -75.4129) %>%
  arrange(desc(alt)) %>%
  select(name, alt) %>%
  slice(1:3)

Error in UseMethod("slice_") :
  no applicable method for 'slice_' applied to an object of class
  "c('tbl_SQLiteConnection', 'tbl_dbi', 'tbl_sql', 'tbl_lazy', 'tbl')"
```

```
airports_db %>%
  filter(lat >= 33.7666, lat <= 36.588,
         lon >= -84.3201, lon <= -75.4129) %>%
  select(name, alt) %>%
  filter(stringr::str_detect(name, pattern = "Raleigh"))

Error in stri_detect_regex(string, pattern, negate = negate, opts_regex =
  object 'name' not found
```

Lazy remote queries

```
airport_car <- airports_db %>%
  filter(lat >=  33.7666, lat <=  36.588,
         lon >= -84.3201, lon <= -75.4129) %>%
  arrange(desc(alt)) %>%
  select(name, alt) %>%
  collect()
```

- Data is never pulled into R unless you explicitly ask for it with `collect()`.

- Work is delayed until the moment it is required. Until I ask for `airport_car`, nothing is communicated to the database.

Close connection

```
DBI::dbDisconnect(con)
```

SQL and R

Create a database

Set up a relational database management system and include some baseball data from package `Lahman`.

```
library(DBI)
library(Lahman)
```

```
con <- dbConnect(RSQLite::SQLite(), ":memory:")
dbWriteTable(con, name = "batting", value = Batting)
dbWriteTable(con, name = "pitching", value = Pitching)
dbWriteTable(con, name = "teams", value = Teams)
```

Seeing tables and fields

```
dbListTables(con)
```

```
#> [1] "batting"  "pitching" "teams"
```

```
dbListFields(con, name = "teams") %>% head()
```

```
#> [1] "yearID"   "lgID"     "teamID"   "franchID" "divID"     "Rank"
```

```
dbListFields(con, name = "pitching")
```

```
#>  [1] "playerID" "yearID"   "stint"    "teamID"   "lgID"     "W"
#>  [7] "L"        "G"        "GS"       "CG"       "SHO"      "SV"
#> [13] "IPouts"   "H"        "ER"       "HR"       "BB"       "SO"
#> [19] "BAOpp"    "ERA"      "IBB"      "WP"       "HBP"      "BK"
#> [25] "BFP"      "GF"       "R"        "SH"       "SF"       "GIDP"
```

Common SQL query structure

Main verbs to query data tables:

```
SELECT columns or computations
  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. Primary computations: MIN, MAX, COUNT, SUM, AVG.

We can perform these queries with dbGetQuery() and paste().

Verb connections

SQL	dplyr
SELECT	`select()`
FROM	Pipe in data frame
WHERE	`filter()` pre-aggregation/calculation
GROUP_BY	`group_by()`
HAVING	`filter()` post-aggregation/calculation
ORDER BY	`arrange()` with possibly a `desc()`
LIMIT	`slice(1:n)`

Examples

Pull some attendance numbers

```
dbGetQuery(con, paste("SELECT yearID, franchID, attendance",
                      "FROM teams",
                      "LIMIT 5"))
```

```
#>   yearID franchID attendance
#> 1   1871      BNA         NA
#> 2   1871      CNA         NA
#> 3   1871      CFC         NA
#> 4   1871      KEK         NA
#> 5   1871      NNA         NA
```

```
dbGetQuery(con, paste("SELECT yearID, franchID, attendance",
                      "FROM teams",
                      "WHERE yearID >= 2000",
                      "LIMIT 5"))
```

```
#>   yearID franchID attendance
#> 1   2000      ANA    2066982
#> 2   2000      ARI    2942251
#> 3   2000      ATL    3234304
#> 4   2000      BAL    3297031
#> 5   2000      BOS    2585895
```

What happens if we change the order or query structure?

```
dbGetQuery(con, paste("FROM teams",
                      "SELECT yearID, franchID, attendance",
                      "WHERE yearID >= 2000",
                      "LIMIT 5"))
```

```
#> Error: near "FROM": syntax error
```

Get the average yearly attendance for each franchise since 2010 and show the top 10.

```
dbGetQuery(con, paste("SELECT franchID, AVG(attendance)",
                      "FROM teams",
                      "WHERE yearID >= 2010",
                      "ORDER BY AVG(attendance) DESC",
                      "LIMIT 10"))
```

```
#>   franchID AVG(attendance)
#> 1      ARI         2422734
```

What went wrong?

Get the average yearly attendance for each franchise since 2010 and show the top 10.

```
dbGetQuery(con, paste("SELECT franchID, AVG(attendance)",
                      "FROM teams",
                      "WHERE yearID >= 2010",
                      "GROUP BY franchID",
                      "ORDER BY AVG(attendance) DESC",
                      "LIMIT 10"))
```

```
#>      franchID AVG(attendance)
#> 1         LAD         3641336
#> 2         STL         3386500
#> 3         NYY         3383453
#> 4         SFG         3240634
#> 5         ANA         3068207
#> 6         CHC         2988555
#> 7         BOS         2950688
#> 8         COL         2796172
#> 9         MIL         2726686
#> 10        PHI         2686706
```

Note that we do not need `yearID` and `attendance` in our `SELECT` line. When do you think the `SELECT` clause is evaluated?

SQL order of execution

Order	Verb
1	FROM
2	WHERE
3	GROUP BY
4	HAVING
5	SELECT
6	ORDER BY
7	LIMIT

How is this different from `dplyr`?

Which players had at least 300 strikeouts (SO) in a season between 1960 and 1990?

```
dbGetQuery(con, paste("SELECT playerID, yearID, MAX(SO) as maxK",
                      "FROM pitching",
                      "WHERE yearID >= 1960 AND yearID <= 1990",
                      "GROUP BY playerID, yearID",
                      "HAVING maxK > 300",
                      "ORDER BY maxK DESC"))
```

```
#>     playerID yearID maxK
#> 1    ryanno01   1973  383
#> 2   koufasa01   1965  382
#> 3    ryanno01   1974  367
#> 4    ryanno01   1977  341
#> 5    ryanno01   1972  329
#> 6    ryanno01   1976  327
#> 7   mcdowsa01   1965  325
#> 8   koufasa01   1966  317
#> 9   richajr01   1979  313
#> 10  carltst01   1972  310
#> 11  lolicmi01   1971  308
#> 12  koufasa01   1963  306
#> 13  scottmi03   1986  306
#> 14  mcdowsa01   1970  304
#> 15  richajr01   1978  303
#> 16   bluevi01   1971  301
#> 17   ryanno01   1989  301
```

Can we restructure the query?

Which players had at least 300 strikeouts (SO) in a season between 1960 and 1990?

```
dbGetQuery(con, paste("SELECT playerID, yearID, MAX(SO) as maxK",
                      "FROM pitching",
                      "HAVING maxK > 300",
                      "GROUP BY playerID, yearID",
                      "WHERE yearID >= 1960 AND yearID <= 1990",
                      "ORDER BY maxK DESC"))
```

```
#> Error: near "GROUP": syntax error
```

```
dbGetQuery(con, paste("SELECT yearID, franchID, attendance",
                      "FROM teams",
                      "HAVING yearID >= 2000",
                      "LIMIT 5"))
```

```
#> Error: a GROUP BY clause is required before HAVING
```

SQL arithmetic and comparison operators

SQL supports the standard +, −, *, /, and % (modulo) arithmetic operators and the following comparison operators.

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

SQL logical operators

Operator	Description
ALL	TRUE if all of the subquery values meet the condition
AND	TRUE if all the conditions separated by AND is TRUE
ANY	TRUE if any of the subquery values meet the condition
BETWEEN	TRUE if the operand is within the range of comparisons
EXISTS	TRUE if the subquery returns one or more records
IN	TRUE if the operand is equal to one of a list of expressions
LIKE	TRUE if the operand matches a pattern
NOT	Displays a record if the condition(s) is NOT TRUE
OR	TRUE if any of the conditions separated by OR is TRUE
SOME	TRUE if any of the subquery values meet the condition

Exercises

1. Add `Salaries` from package `Lahman` as a table to your in-memory database.

2. Compute the team salaries for each team in 2016 and display the 5 teams with the highest payroll. Which team had the lowest payroll in that year?

3. Who were the top 10 teams according to win percentage since 1990? *Hint*: https://www.w3schools.com/sql/func_sqlserver_cast.asp

4. How would you combine the batting and salaries tables to match up the players and years? Take a look at `?dplyr::join`. Try to combine the R data frame objects `Batting` and `Salaries`.

References

1. Introduction to dbplyr. (2021). https://cran.r-project.org/web/packages/dbplyr/vignettes/dbplyr.html

2. SQL Tutorial - w3resource. (2021). https://www.w3resource.com/sql/tutorials.php.