Lecture 19: Oct 19, 2018

Advanced SQL

- *SQL Joins*
- *dbplyr*
- *SQL Injection*
- *Resources*

James Balamuta
STAT 385 @ UIUC

Announcements

- **hw07** is due **Friday, Nov 2nd, 2018** at **6:00 PM**
- **Office Hour Changes**
  - **John Lee's** are now from **4 - 5 PM** on **WF**
  - **Hassan Kamil's** are now from **2:30 - 3:30 PM** on **TR**
- **Quiz 08** covers Week 7 contents @ CBTF.
  - Window: Oct 16th - 18th
  - Sign up: https://cbtf.engr.illinois.edu/sched
- Want to review your homework or quiz grades?
  Schedule an appointment.

Last Time

- **Connecting to a Database**
  - Interactively obtaining and updating data
- **Structured Query Language**
  - Declarative domain-specific language that handles data querying, manipulation, access, and definitions.

Lecture Objectives

- **Manipulating** SQL queries using DBI's interface
- **Write** SQL Join queries.
- **Translate** dplyr code to SQL queries.
- **Protect** SQL queries from SQL Injections.

DBI

Previously

Table to Data Frame

… database logic vs *R's* data structures …

Students

Table
(data.frame)

Field
(Column)

Record
(Row)

id	firstname	lastname	age	instate
1	Billy	Joe	23	FALSE
2	Theodore	Squirrel	25	TRUE
3	Keeya	Nod	21	TRUE
Integer	Character	Character	Integer	Logical

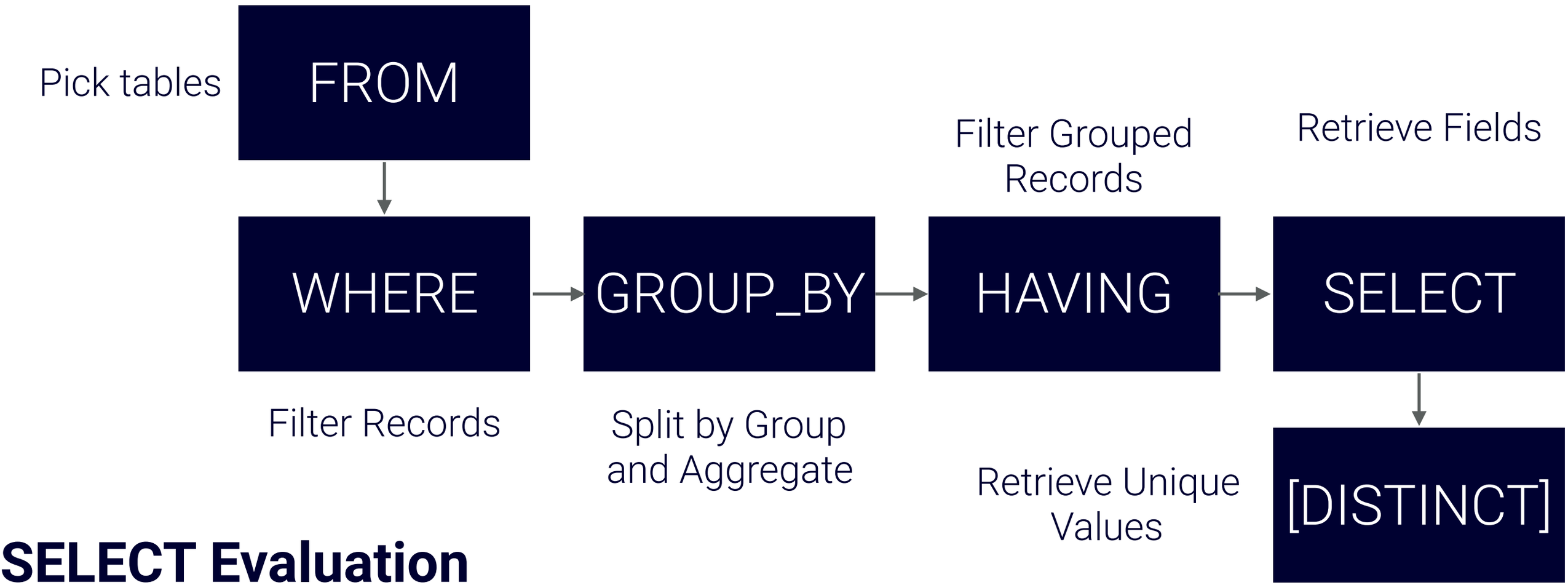
Table Scheme
(Data Types)

Previously

```
-- Select
-- Retrieval of data from a table.

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

-- Statements inside of [ ] are optional.
```



SELECT Evaluation

Translators Guide

… moving from *R* to *SQL* and back again !!!

R	*SQL*
Row (Observation)	Record
Column (Variable)	Field
data.frame	Table
Data Types of Variables	Table Schema
Subset	SELECT *columns* FROM *table* WHERE *condition*
Order	ORDER BY *columns* <ASC | DSC>
Aggregation by Group (Split-Apply-Combine)	GROUP BY
Merging	**JOIN**

Previously

SQLite

... embedded database ...

```
# Working with a Database Locally

# In memory connection...
db = DBI::dbConnect(
        RSQLite::SQLite(), ":memory:"
)

# File-backed connection...
db = DBI::dbConnect(
        RSQLite::SQLite(),
        "my_db.sqlite"
)
```

```r
# Copy and Verify Data

# No tables are in the db.
DBI::dbListTables(db)
# character(0)

# Copy a data.frame to a DB table
DBI::dbWriteTable(db, "mtcars", mtcars)

# Verify copy by viewing ALL tables in the Database
dbListTables(db)
# [1] "mtcars"

# See column / variable names for a specific table
dbListFields(db, "mtcars")
#  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear" "carb"
```

Data Transference

… transforming data from *R* to a Database via *DBI* …

```r
# Retrieve data from a DB
# Convert a table in a Database to a data.frame in R

my_local_mtcars = DBI::dbReadTable(db, "mtcars")

# Modify the local table
my_local_mtcars$mpg = my_local_mtcars$mpg + 20
my_local_mtcars

# Note that the table in the DB did not change.
DBI::dbReadTable(db, "mtcars")

# How could we update the table in the database?
```

Data Retrieval and Updates

… local vs. database …

Direct Queries

... retrieving only a data.frame ...

```r
# Construct a SQL query
my_df_mtcars = DBI::dbGetQuery(db, "SELECT * FROM mtcars")

# Equivalent to using
my_df_mtcars = DBI::dbReadTable(db, "mtcars")

# Subset the data
subset_mtcars = DBI::dbGetQuery(db, "SELECT mpg, wt FROM mtcars")

# Subset the data with a WHERE statement
high_mpg_mtcars = DBI::dbGetQuery(db, "SELECT * FROM mtcars WHERE mpg > 20")
```

Direct Queries

... retrieving only a data.frame ...

```r
# Construct a SQL query
my_df_mtcars = DBI::dbGetQuery(db, "SELECT * FROM mtcars")

# Equivalent to using
my_df_mtcars = DBI::dbReadTable(db, "mtcars")

# Subset the data
subset_mtcars = DBI::dbGetQuery(db, "SELECT mpg, wt FROM mtcars")

# Subset the data with a WHERE statement
high_mpg_mtcars = DBI::dbGetQuery(db, "SELECT * FROM mtcars WHERE mpg > 20")
```

Alternative Direct Query

… delayed pull of data …

```r
# Create a result object
rs = DBI::dbSendQuery(db, "SELECT * FROM mtcars")

# Retrieve column information
DBI::dbColumnInfo(rs)

# Fetch all the results
DBI::dbFetch(rs)

# Clear the results
DBI::dbClearResult(rs)
```

Paginated Queries

… retrieving chunks of the table as a data.frame …

```r
# Create a result object
rs = DBI::dbSendQuery(db, "SELECT * FROM mtcars")

# Process result set until all data has been retrieved
while (!DBI::dbHasCompleted(rs)) {
  # Retrieve chunks of data
  chunk = DBI::dbFetch(rs, 25)
  # See chunk sizes
  print(nrow(chunk))
}

# Clear the results
DBI::dbClearResult(rs)
```

SQL Joins

Joining Paradigms

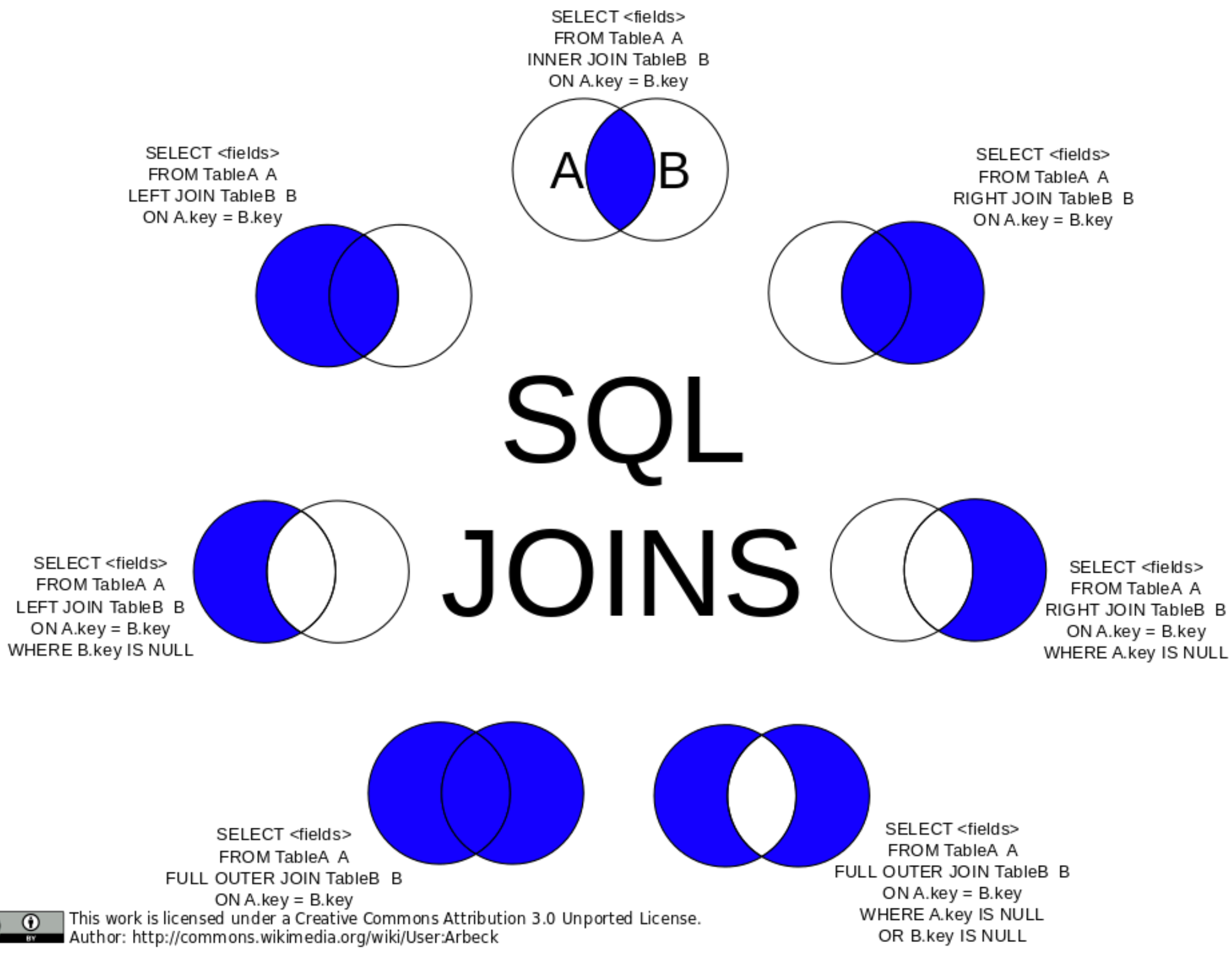
… the many possible ways to merge data …

dplyr	Base R merge()	SQL
inner_join(x, y, by = "z")	merge(x, y, by = "z")	SELECT * FROM x **INNER JOIN** y **USING** (z)
inner_join(x, y, by = c("a" = "b"))	merge(x, y, by = c("a" = "b"))	SELECT * FROM x **INNER JOIN** y **ON** x.a = y.b
left_join(x, y, by = "z")	merge(x, y, by = "z", all.x = TRUE)	SELECT * FROM x **LEFT OUTER JOIN** y **USING** (z)
right_join(x, y, by = "z")	merge(x, y, by = "z", all.y = TRUE)	SELECT * FROM x **RIGHT OUTER JOIN** y **USING** (z)
full_join(x, y, by = "z")	merge(x, y, by = "z", all.x = TRUE, all.y = TRUE)	SELECT * FROM x **FULL OUTER JOIN** y **USING** (z)

* Note that **SQLite** *only* supports **three** types of JOINS:
INNER JOIN, LEFT OUTER JOIN (e.g. LEFT JOIN), and CROSS JOIN.

SQL Joins

… Venn Diagram overview with code …



Source

Visual Joins

… joins in real time …

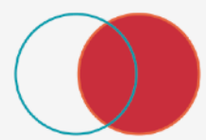
Visual JOIN

Understand how joins work by interacting and see it visually

INNER JOIN
(or JOIN)

LEFT JOIN

RIGHT JOIN



OUTER JOIN
(with UNION)

SQL

```
SELECT users.name, likes.like FROM users JOIN likes ON users.id = likes.user_id;
```

Description »

Users

ID	Name
1	Patrik
~~2~~	~~Albert~~
3	Maria
4	Darwin
~~5~~	~~Elizabeth~~

JOIN

Name	Like
Maria	Stars
Patrik	Climbing
Patrik	Code
Darwin	Apples

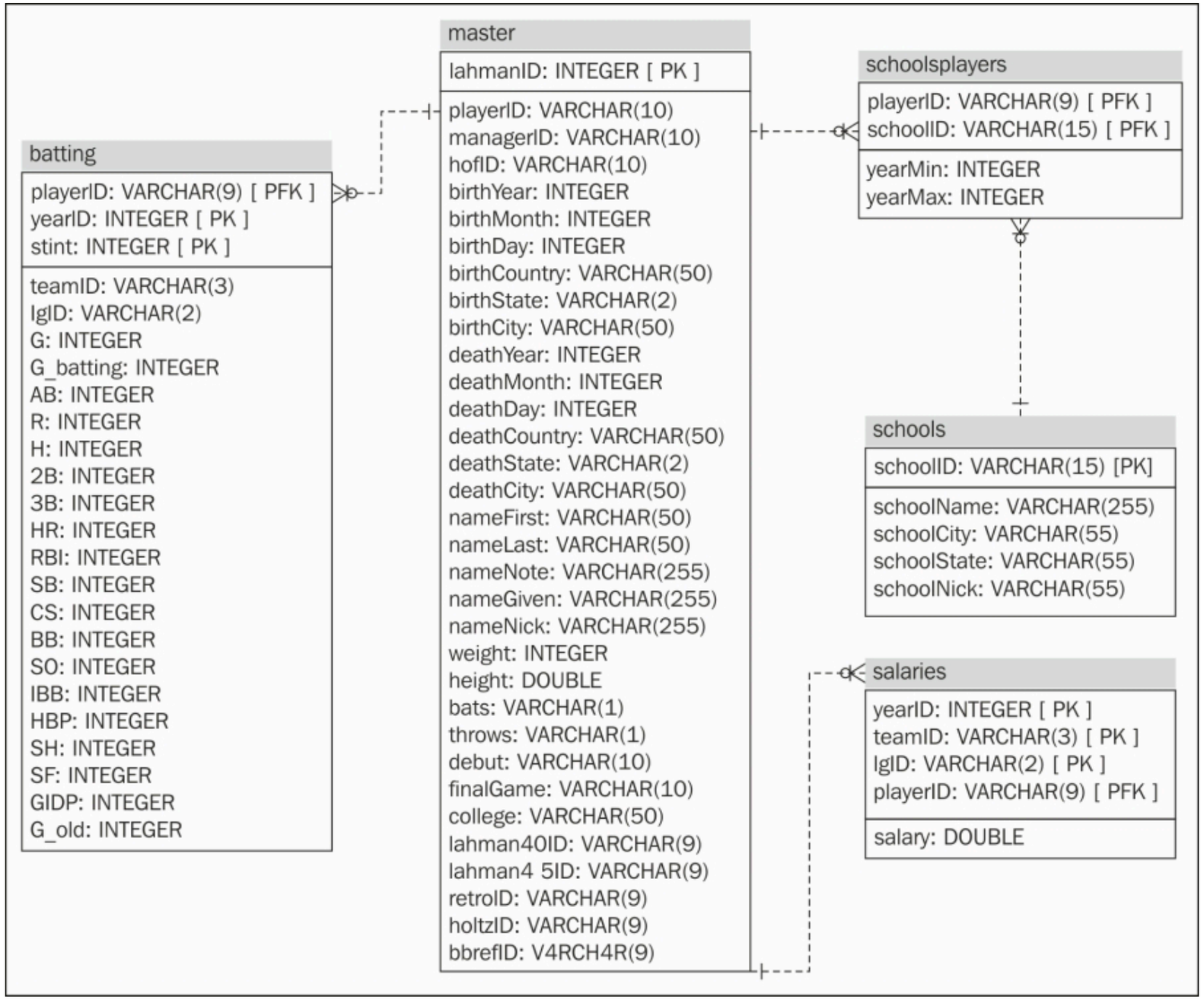
Likes

User ID	Like
3	Stars
1	Climbing
1	Code
~~6~~	~~Rugby~~
4	Apples

http://joins.spathon.com/

Lahman Table Schema

… small sampling of database schema …



```
-- JOINS
-- When joins go bad...

-- Note that the join here is only on playerID... However, players can play for multiple years
SELECT playerID, HR, salary
FROM Batting
JOIN Salaries USING(playerID)
LIMIT 10

-- Need to provide a second key to avoid the issue
SELECT yearID, playerID, HR, salary
FROM Batting
JOIN Salaries USING(yearID, playerID)
LIMIT 10
```

Joins with Duplicates

... adding additional structure to the join ...

Variant of INNER JOIN

… two ways to join SQL tables together …

```
-- Join tables via JOIN … USING ( … )
SELECT yearID, playerID, HR, salary
FROM Batting
JOIN Salaries USING(yearID, playerID)
LIMIT 10
```

JOIN … USING (…)
merges on column specified

```
SELECT Salaries.yearID, Salaries.playerID, HR, salary
FROM Batting
INNER JOIN Salaries ON
Salaries.playerID = Batting.playerID AND
Salaries.yearID = Batting.yearID
LIMIT 10
```

INNER JOIN … ON …
requires a condition
(comparison) to merge data

Your Turn

Retrieve the salaries of the top 5 Average Home Run (HR) hitting players across the years

dbplyr

dbplyr

… dynamically generating *SQL* for *R* code written with *dplyr*…

dbplyr functions	Description
	Build a SQL string.
copy_to	Copy a local data frame to a DBI backend.
do	Perform arbitrary computation on remote backend
escape, sql_vector	Escape/quote a string.
ident, ident_q, is.ident	Flag a character vector as SQL identifiers
in_schema	Refer to a table in a schema
inner_join, left_join, right_join, full_join, semi_join, anti_join	Join sql tbls.
memdb_frame, src_memdb	Create a database table in temporary in-memory database.
sql, is.sql, as.sql	SQL escaping.
src_dbi, **tbl**	dplyr backend for any DBI-compatible database
translate_sql, translate_sql_	Translate an expression to sql.

https://dbplyr.tidyverse.org/reference/index.html

Connections via dbplyr + DBI

… writing code using the *dplyr* backend to *SQL*…

```
# Setup database and use local data sets
db = DBI::dbConnect(RSQLite::SQLite(), path = ":memory:")

# Copy to database via dplyr
dplyr::copy_to(db, mtcars, "mtcars")
dplyr::copy_to(db, iris, "iris")

# View listed tables
dbListTables(db)

# Cannot use a db established with dbConnect with dbplyr's table view.
src_tbls(db)
# Error in UseMethod("src_tbls") :
#  no applicable method for 'src_tbls' applied to an object of class
# "c('SQLiteConnection', 'DBIConnection', 'DBIObject')"
```

Connections via only dbplyr

… writing code using the *dplyr* backend to *SQL*…

```
# Establish a source
db_lahman = src_sqlite("lahman2016.sqlite")

# View tables in database
src_tbls( db_lahman )
# [1] "AllstarFull"        "Appearances"        "AwardsManagers"
# [4] "AwardsPlayers"      "AwardsShareManagers" ...

# Specify a table inside of the database
table_batting = tbl(db_lahman, "Batting")
# Source:   table<Batting> [?? x 24]
# Database: sqlite 3.22.0 [/cloud/project/lahman2016.sqlite]
#   playerID yearID stint teamID lgID      G G_batting    AB     R     H  `2B`  `3B`    HR
#   <chr>     <int> <int> <chr>  <chr> <int>     <int> <int> <int> <int> <int> <int> <int>
# 1 aardsda…   2004     1 SFN    NL       11        NA     0     0     0     0     0     0
# 2 aardsda…   2006     1 CHN    NL       45        NA     2     0     0     0     0     0
```

New header with **DB** info

dbplyr

… augmenting *R* code written under dplyr to …

```
# Use the db to compute averages
table_batting %>%
  summarise(avg_hr = mean(HR),
            avg_h = mean(H)) %>%
            collect()                    # Pull data from the DB into R via collect
#   avg_hr        avg_h
#    <dbl>        <dbl>
# 2.813599   37.13993
```

Generate SQL

… see how *dplyr* code translates to *SQL* …

```
# Store the analytical pipeline using the table source
top5 = table_batting %>%
  head(5) %>%
  select(playerID, yearID, AB, H, HR)

# Show result of querying data
top5

# View underlying query
top5 %>%
  show_query()
# <SQL>
# SELECT `playerID`, `yearID`, `AB`, `H`, `HR`
# FROM (SELECT *
# FROM `Batting`
# LIMIT 5)
```

Yuck. Can we do better?

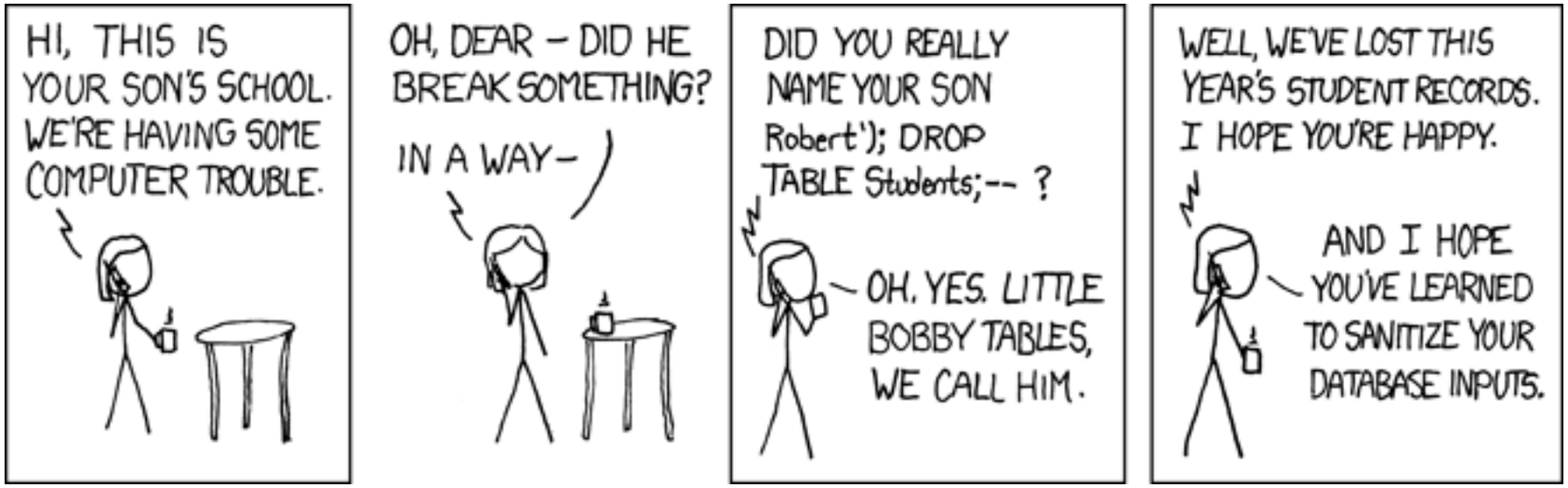
Your Turn

Translate the SQL query that found the salaries of the top 5 Average Home Run (HR) hitting players across the years to dplyr

SQL Injection

Definition:
SQL Injection refers to the act of using non-sanitized user input in a query.

Exploits of a Mom



https://xkcd.com/327/

Reality of Injections

… frequent, leak personal details, and are costly …

CHICAGO 5 | Home | News | Weather | Investigations | Entertainment | •••

How to Protect Your Vote This Election Season | 12 Russian Intel Officials Indicted in 2016 Election Hack | McDonald's Salads Linked to Rash of Intestinal Illnesses | Gavin Newsom Leads Pack in Governor's Race NBC4 SoCal

In Illinois, a majority of voting machines need upgrading and more than 58 percent of local elections jurisdictions said they did not feel they had the resources to adequately secure their voting systems, according to a survey conducted by NBC 5 Investigates.

NBC 5 sent all 108 local jurisdictions a brief survey to gauge readiness ahead of next month's mid-term elections. Of the half that responded, despite challenges, 94 percent said they felt well-prepared from a cyber-security standpoint.

It's an important distinction following the 2016 hack of the state-run voter registration database.

Months before the 2016 presidential election, the Illinois State Board of Elections suffered a stunning breach. The personal information of 76,000 voters, including names, birth dates, driver's license numbers, and in some cases, the last four digits of social security numbers, were viewed by cyber criminals.

The main culprit: Russia.

"It was basically like having a really good home security system, but you leave a window wide open and someone comes in," said Matt Dietrich, public information officer at the State Board of Elections.

The SBE noticed the error and patched it right away. Investigators said the hackers deployed what's called an "SQL Injection," which is commonly used to attack databases.

Source

Hackers breach web hosting provider for the second time in the past year

Company hacked again despite claiming to have boosted security measures and undergone a security audit.

By Catalin Cimpanu for Zero Day | October 11, 2018 -- 13:53 GMT (06:53 PDT) | Topic: Security

Source

abc NEWS | VIDEO | LIVE | SHOWS

Website-infecting SQL injection attacks hit 450,000 a day

By BYRON ACOHIDO and USA TODAY March 16, 2009

Share | Tweet

Cybercriminals are spreading invisible infections far and wide across the Internet by hammering hundreds of thousands of websites each day with so-called SQL injection attacks.

The trend started last summer and has continued to accelerate. IBM Internet Security Systems says it identified 50% more infected Web pages in the last three months of 2008 than it did in all of 2007.

Source

Setting Up Data

… using DBI to write a data.frame into a table in a database …

```r
# Retrieve data package for flights
install.packages("nycflights13")

# Establish an airport database
db_airports = DBI::dbConnect(RSQLite::SQLite(), path = ":memory:")

# Retrieve airports data and remove a column
airports = nycflights13::airports[, -which(colnames(nycflights13::airports) == "tzone")]

# Copy to database
dbWriteTable(db_airports, "airports", airports)
```

```r
# Static query with a fixed
dbGetQuery(db_flights,
    paste0("SELECT * FROM airports WHERE faa = 'GPT')
)
# Allow the FAA code for the airport to change
airport_code = "GPT"
dbGetQuery(db_flights,
    paste0("SELECT * FROM airports WHERE faa = '", airport_code ,"' ")
)
# Allow the FAA code for the airport to change
airport_code_inject = "GPT' or faa = 'MSY"
dbGetQuery(db_flights,
    paste0("SELECT * FROM airports WHERE faa = '", airport_code_inject ,"' ")
)
```

Single quote for value

Dynamic Selection

… woes of selection querying …

SQL Injected

… the many possible ways to merge data …

```
# Allow the FAA code for the airport to change
airport_code_inject = "GPT' or faa = 'MSY"
dbGetQuery(db_flights,
     paste0("SELECT * FROM airports WHERE faa = '", airport_code_inject ,"' ")
)

-- Evaluates to the following SQL query
SELECT * FROM airports WHERE faa = 'GPT' or faa = 'MSY'

-- We only wanted users to specify ONE code
SELECT * FROM airports WHERE faa = 'GPT'
```

Parameterized Input

… protecting the query and avoiding insanity of user input …

```
# Secure query
airport_code_safe = "GPT"
dbGetQuery(db_flights,
    "SELECT * FROM airports WHERE faa = :airport_code",
    params = list(airport_code = airport_code_safe)
)

# Protected against the SQL injection query
airport_code_inject = "GPT' or faa = 'MSY"
dbGetQuery(db_flights,
    "SELECT * FROM airports WHERE faa = :airport_code",
    params = list(airport_code = airport_code_inject)
)
```

Colon followed by **variable name**

Specify **variable name** and **value** for query

Resources

Databases in R

… an overview of drivers, querying, and more !!!

Databases using R

Not Secure | db.rstudio.com

Databases using R from RStudio

Overview | Getting Started | Best Practices | Databases | Blog

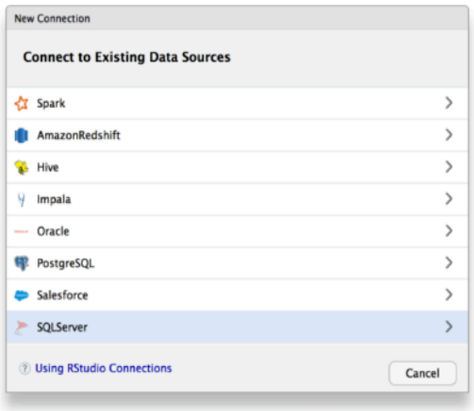
- Getting Started
  - Connect to a Database
  - Database Queries
- Packages
  - dplyr
  - DBI
  - odbc
  - pool
  - dbplot
  - tidypredict
- RStudio
  - Connections Pane
  - Professional Drivers
- Best Practices
  - Setting up ODBC Drivers
  - Run Queries Safely
  - Securing Deployed Content
  - Securing Credentials
  - Making Scripts Portable

Databases using R

At **RStudio**, we are working to make it as easy as possible to work with databases in R. This work focuses on **three key areas**:

1. RSTUDIO PRODUCTS

- The new RStudio Connections Pane makes it possible to easily connect to a variety of data sources, and **explore the objects and data** inside the connection
- To RStudio commercial customers, we offer RStudio Professional ODBC Drivers, these are data connectors that help you connect to some of the most popular databases.



2. USE BEST-IN-CLASS PACKAGES

Build and/or document how to use packages such as: dplyr, DBI, odbc, keyring and pool

3. PROMOTE BEST PRACTICES

This website is the main channel to provide support in this area. RStudio is also working through other delivery channels, such as upcoming webinars and in-person training during our RStudio conferences.

Read more →

Latest Announcements

https://db.rstudio.com/

dbplyr

… database-backend for dplyr …

A 'dplyr' Backend for Database

https://dbplyr.tidyverse.org

dbplyr part of the tidyverse

Reference Intro Articles News

Overview

dbplyr is the database backend for dplyr. If you are using dplyr to connect to databases, you generally will not need to use any functions from dbplyr, but you will need to make sure it's installed.

If you find any bugs, please file in dplyr.

Installation

```
# You can install the released version from CRAN
install.packages("dbplyr")

# Or the the development version from GitHub:
# install.packages("devtools")
devtools::install_github("tidyverse/dplyr")
```

Usage

```
library(dplyr, warn.conflicts = FALSE)

con <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
copy_to(con, mtcars)

mtcars2 <- tbl(con, "mtcars")
mtcars2
#> Source:     table<mtcars> [?? x 11]
#> Database:   sqlite 3.11.1 [:memory:]
#>
```

Links

Browse source code at https://github.com/tidyverse/dbplyr

Report a bug at https://github.com/tidyverse/dplyr/issues

License

MIT + file LICENSE

Developers

Hadley Wickham
Author, maintainer

RStudio
Copyright holder, funder

Dev status

build passing

CRAN 0.7.7

coverage 78%

https://dbplyr.tidyverse.org/

Acknowledgements

Acknowledgements

- Edgar Ruiz for both db.rstudio.com and the **dbplyr** package.
- Kirill Müller for the **DBI** package.
- Hadley Wickham for the **dbplyr** package.