

# Import



# DBI

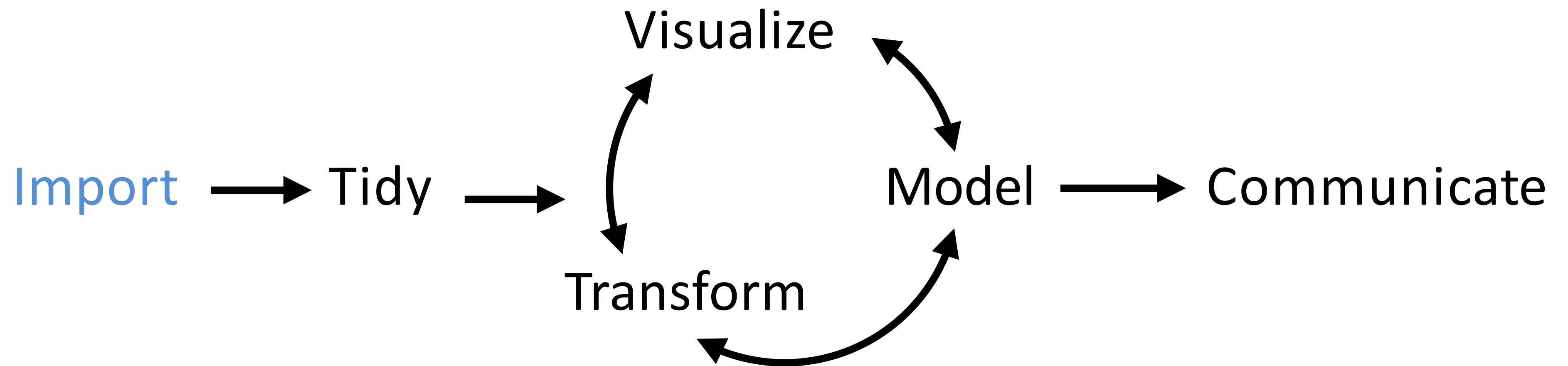


“I ~~rob banks~~ use databases  
because its where the ~~money~~  
data is.”

—Willie Sutton



# (Applied) Data Science



Program

# SQL





“SQL is a domain specific  
language used in programming  
and ... data held in a relational  
database management  
system”

—Wikipedia



# Structuring a Query



# QUERIES

ORDER	CLAUSE	FUNCTION
1	from	Choose and join tables to get base data.
2	where	Filters the base data.
3	group by	Aggregates the base data.
4	having	Filters the aggregated data.
5	select	Returns the final data.
6	order by	Sorts the final data.
7	limit	Limits the returned data to a row count.

Source: [periscope data](#)

# EXERCISE



- ▶ Run `apps/wprdc_sql.R`
- ▶ Build a query that selects all of the crimes by neighborhood from the [City of Pittsburgh Police Blotter](#)
  - ▶ Hint 1: FROM would be the resource ID (1797ead8-8262-41cc-9099-cbc8a161924b)
  - ▶ Hint 2: The WPRDC uses a Postgresql backend
    - ▶ This means that anything that contains number or capital letters have to be wrapped in quotes

A small, colorful, abstract graphic with a grid-like pattern in shades of yellow, orange, and red, serving as a background for the timer text.

**2:00**





# SOLUTION

```
SELECT * FROM "1797ead8-8262-41cc-9099-cbc8a161924b"
```

# WHERE





# BETWEEN ... AND

- ▶ BETWEEN

- ▶ *Grab Values between two other values, like IN but for numeric values*
- ▶ *Works like < and >*

```
SELECT column_name(s)
FROM table_name
WHERE column_name BETWEEN value1
AND value2;
```


# IN STATEMENTS

- ▶ Useful for when you have an input that returns multiple
- ▶ This works the same way `%in%` does in R
- ▶ Checks to see if the value in the column matches *any* of the values in your list

```
SELECT column_name(s)  
FROM table_name  
WHERE column_name IN (value1, value2, ...)
```



# EXERCISE

- 
- ▶ Run apps/wprdc\_sql.R
  - ▶ This time let's target 311 requests (resource ID:76fda9d0-69be-4dd5-8108-0de7907fc5a4)
  - ▶ Use the BETWEEN function as a WHERE filter to get 311 requests from from the last week.
    - ▶ Stretch goal: Use the IN Filter to only get requests of the Potholes, Weeds/Debris and Overgrowth call types.



5:00



# SOLUTION

```
SELECT * FROM "76fda9d0-69be-4dd5-8108-0de7907fc5a4"  
WHERE "CREATED_ON" BETWEEN '2019-09-30' AND '2019-  
10-06'
```

```
SELECT * FROM "76fda9d0-69be-4dd5-8108-0de7907fc5a4"  
WHERE "CREATED_ON" BETWEEN '2019-09-30' AND '2019-10-06'  
AND "REQUEST_TYPE" IN ('Potholes', 'Weeds/Debris',  
'Overgrowth')
```



# SELECT Functions and GROUP BY



# SQL FUNCTIONS

- ▶ Sometimes you don't just want the raw data
- ▶ You want to aggregate the data in SQL before you load it into R
  - ▶ Use another server to do the heavy lifting so you don't have to!
- ▶ This is where



# DISTINCT

- ▶ DISTINCT()
  - ▶ Every unique value of a column.
  - ▶ Placing TWO columns inside will return unique instances of both columns:

```
DISTINCT("REQUEST_TYPE", "DEPARTMENT")
```

# MATH FUNCTIONS

- ▶ **MIN()**
  - ▶ Returns minimum value in a column(s)
- ▶ **MAX()**
  - ▶ Return max value in a column(s)
- ▶ **COUNT()**
  - ▶ Return



# COUNT, AVERAGE, SUM

- ▶ COUNT() - returns the number of rows that your query returns
  - ▶ SELECT COUNT(column\_name)  
FROM table\_name
- ▶ AVG() - returns the average value of a numeric column.
  - ▶ SELECT AVG(column\_name)  
FROM table\_name
- ▶ SUM() - function returns the total sum - numeric columns only
  - ▶ SELECT SUM(column\_name)  
FROM table\_name

# GROUP BY

- ▶ This is helpful for when you are doing any of the summary functions mentioned in the previous slides. (COUNT, SUM, MAX etc)
- ▶ Any column that isn't handled with a function should be included in your GROUP BY

```
SELECT column_name(s), max(column_name)
FROM table_name
WHERE condition
GROUP BY column_name(s)
```

# EXERCISE



- ▶ Run apps/wprdc\_sql.R
  - ▶ Build a query that counts the number crimes by neighborhood from the [City of Pittsburgh Police Blotter](#)

**5:00**





# SOLUTION

```
SELECT
"INCIDENTNEIGHBORHOOD",
COUNT("CCR")
FROM "1797ead8-8262-41cc-9099-cbc8a161924b"
GROUP BY "INCIDENTNEIGHBORHOOD"
```

# Other Functions



# CASE

- ▶ CASE statements are for when you want to return categorical values based off of something else.

```
SELECT OrderID, Quantity,  
CASE  
    WHEN Quantity > 30 THEN "The quantity is greater than 30"  
    WHEN Quantity = 30 THEN "The quantity is 30"  
    ELSE "The quantity is under 30"  
END AS QuantityText  
FROM OrderDetails;
```



# CONCAT

- ▶ CONCAT()
  - ▶ This is mostly used when you have multiple columns you need.
  - ▶ May look different depending on DB server

```
SELECT CONCAT(column1, " ", column2) AS ConcatenatedString;
```

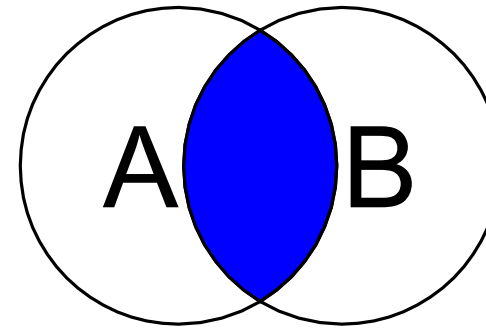
OR

```
SELECT column1 || " " || column2 AS ConcatenatedString;
```

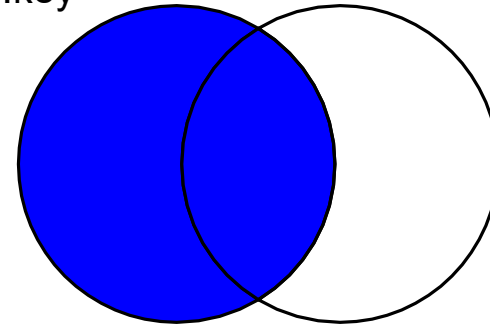
# JOINS



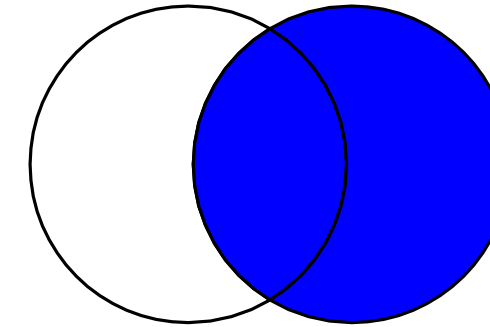
SELECT <fields>  
FROM TableA A  
INNER JOIN TableB B  
ON A.key = B.key



SELECT <fields>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.key = B.key



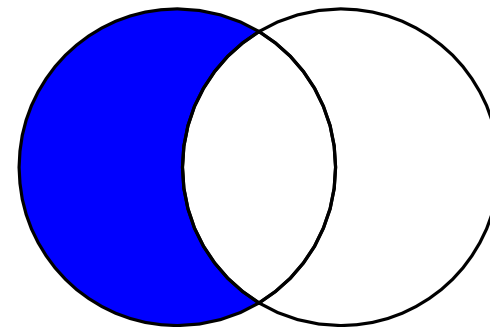
SELECT <fields>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.key = B.key



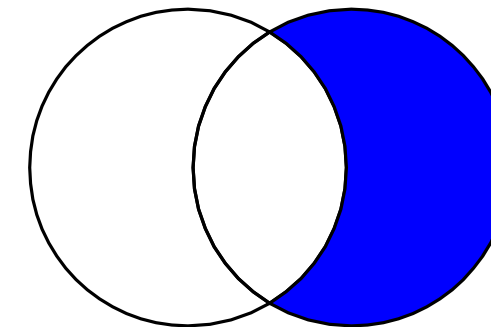
# SQL

# JOINS

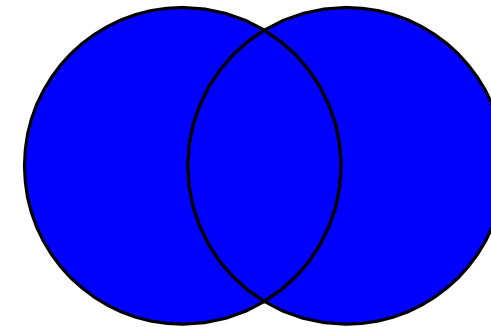
SELECT <fields>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.key = B.key  
WHERE B.key IS NULL



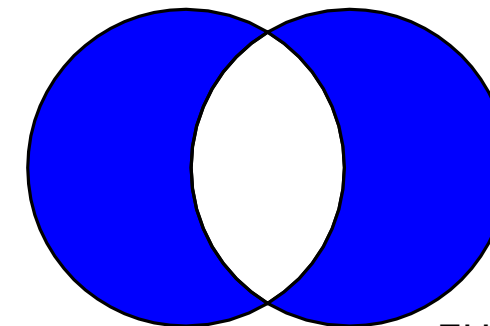
SELECT <fields>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.key = B.key  
WHERE A.key IS NULL



SELECT <fields>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.key = B.key



SELECT <fields>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.key = B.key  
WHERE A.key IS NULL  
OR B.key IS NULL

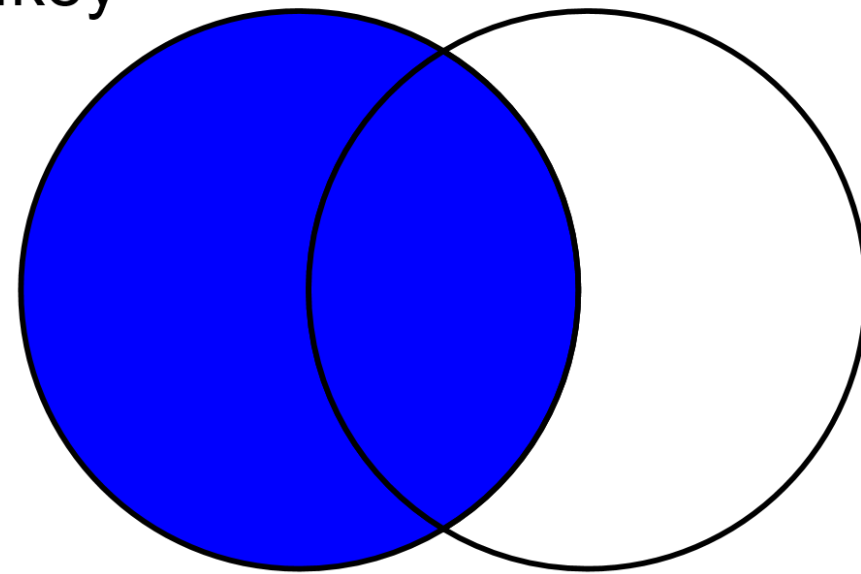


This work is licensed under a Creative Commons Attribution 3.0 Unported License.  
Author: <http://commons.wikimedia.org/wiki/User:Arbeck>

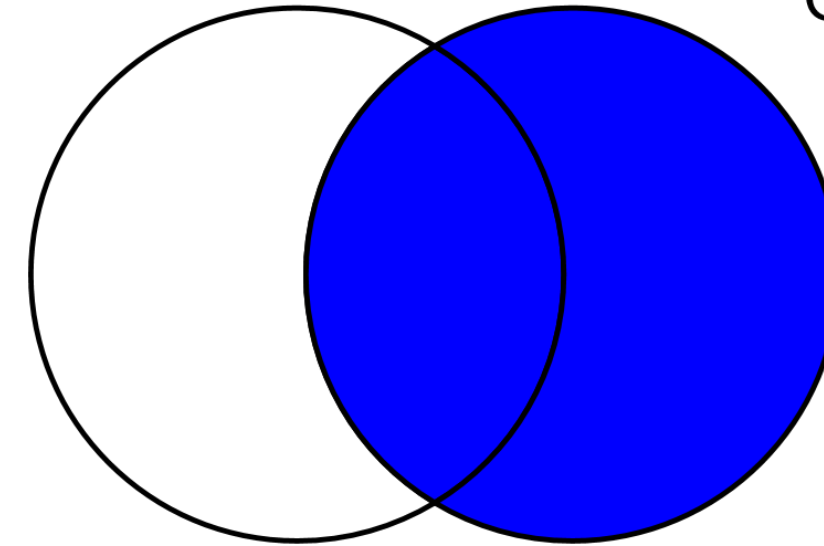


# Left/Right Join

```
SELECT <fields>  
FROM TableA A  
LEFT JOIN TableB B  
ON A.key = B.key
```

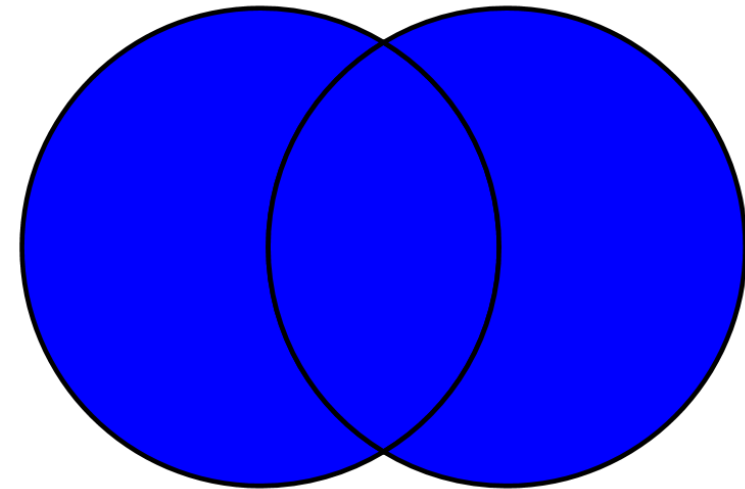


```
SELECT <fields>  
FROM TableA A  
RIGHT JOIN TableB B  
ON A.key = B.key
```

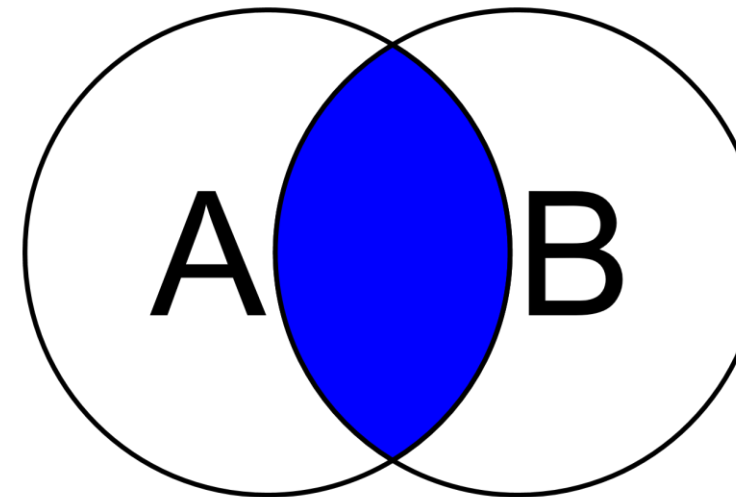


# Inner/Outer Join

```
SELECT <fields>  
FROM TableA A  
FULL OUTER JOIN TableB B  
ON A.key = B.key
```

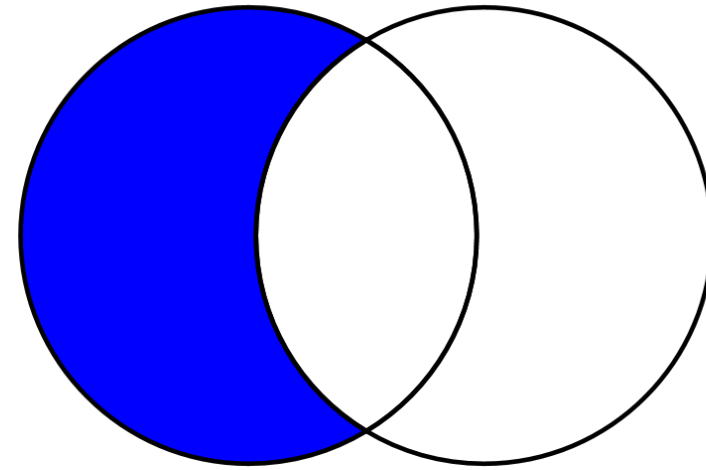


```
SELECT <fields>  
FROM TableA A  
INNER JOIN TableB B  
ON A.key = B.key
```

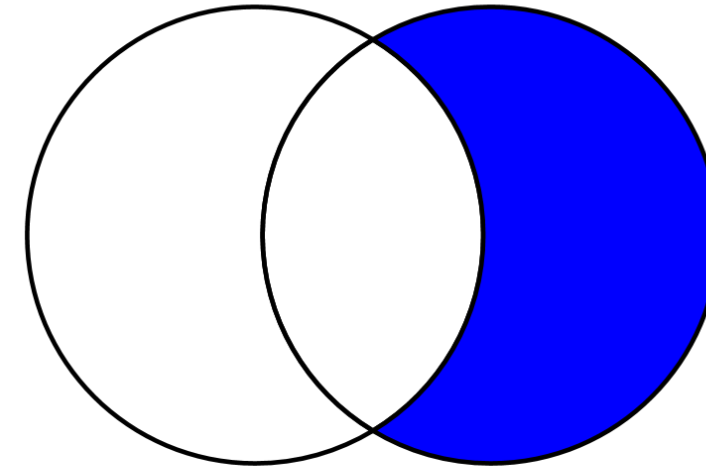


# Anti-Joins

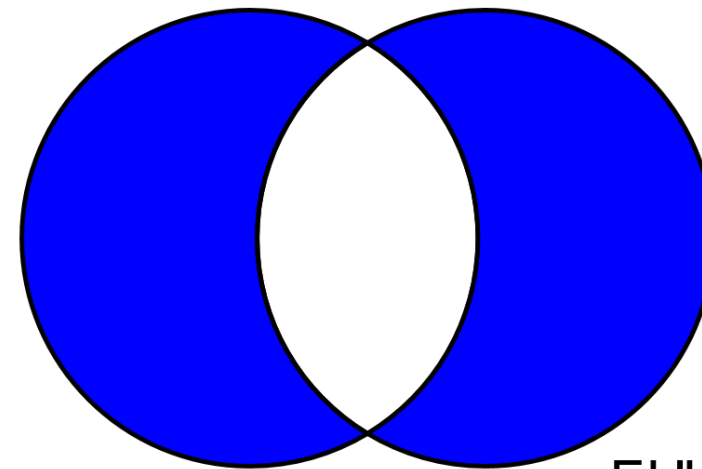
```
SELECT <fields>
FROM TableA A
LEFT JOIN TableB B
ON A.key = B.key
WHERE B.key IS NULL
```



```
SELECT <fields>
FROM TableA A
RIGHT JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
```



```
SELECT <fields>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.key = B.key
WHERE A.key IS NULL
OR B.key IS NULL
```



# Writing SQL





# SQL IDE'S

- ▶ There are a bunch of SQL IDE's each database provider has their own
- ▶ If you're in a workplace like mine with no standard then I suggest something like DBeaver because it connects to pretty much everything
- ▶ If not, then use whatever comes standard with the platform

DB

Connections



# CONNECTING

- ▶ Database connectors require that your computer has the necessary software.
  - ▶ This will depend on what database type you are trying to connect to



# ALLOWING HANDSHAKES

- ▶ To setup database connections you will need to install the proper drivers.
  - ▶ The steps for this can be found here: <https://db.rstudio.com/best-practices/drivers/>
  - ▶ In general setup on Windows is a little bit easier since ODBC Data Source Administrator can be used
- ▶ Your machine may already have drives installed if you've already installed SQL IDE's such as: pgAdmin, DBeaver, or the MySQL Workbench



# Storing Credentials



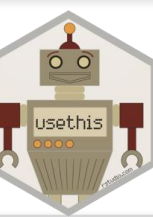
# ENVIRONMENTAL VARIABLE OR FILE

- ▶ You should never “hard code” your credentials into an app.
- ▶ Instead you should store them as environmental variables, or in a hidden file that you ignore in the Git Repository

- ▶ Why?

If something requires that you to login, we can assume that not just anybody should be able to access it.

Think of your credentials like your debit card and pin number



# BUILDING AN ENVIRON FILE

- ▶ The usethis package has a function that will build your .Renviron file in your directory or for your entire profile.

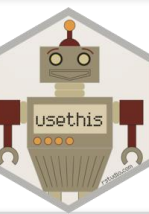
```
usethis::edit_r_environ("project")
```

- ▶ How are .Renviron Files structured?

```
uid=some_username  
pwd=aPassword
```

A new line for each variable

No spaces between variable name and value



# LOADING VARIABLES

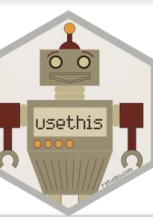
- ▶ Small difference between credentials in your profile or the project folder.
- ▶ The string argument is the name you gave your variable

## Profile

```
uid <- Sys.getenv("uid")  
pwd <- Sys.getenv("pwd")
```

## .Renviron

```
readRenviron(".Renviron")  
  
uid <- Sys.getenv("uid")  
pwd <- Sys.getenv("pwd")
```



# ESTABLISHING CONNECTIONS

- ▶ Each data base type has a different connection string and list of requirements.

```
conn <- dbConnect(odbc::odbc(), driver = "FreeTDS", server = "IP_or_HOST_ADDRESS", port  
= 1433, database = "DBName", uid = un, pwd = pw, TDS_Version = "8.0")
```

- ▶ More on connection strings: <https://db.rstudio.com/best-practices/drivers/#connecting-to-a-database-in-r>



# Your Turn

Go to Rstudio and open the dbi\_example.Rmd

