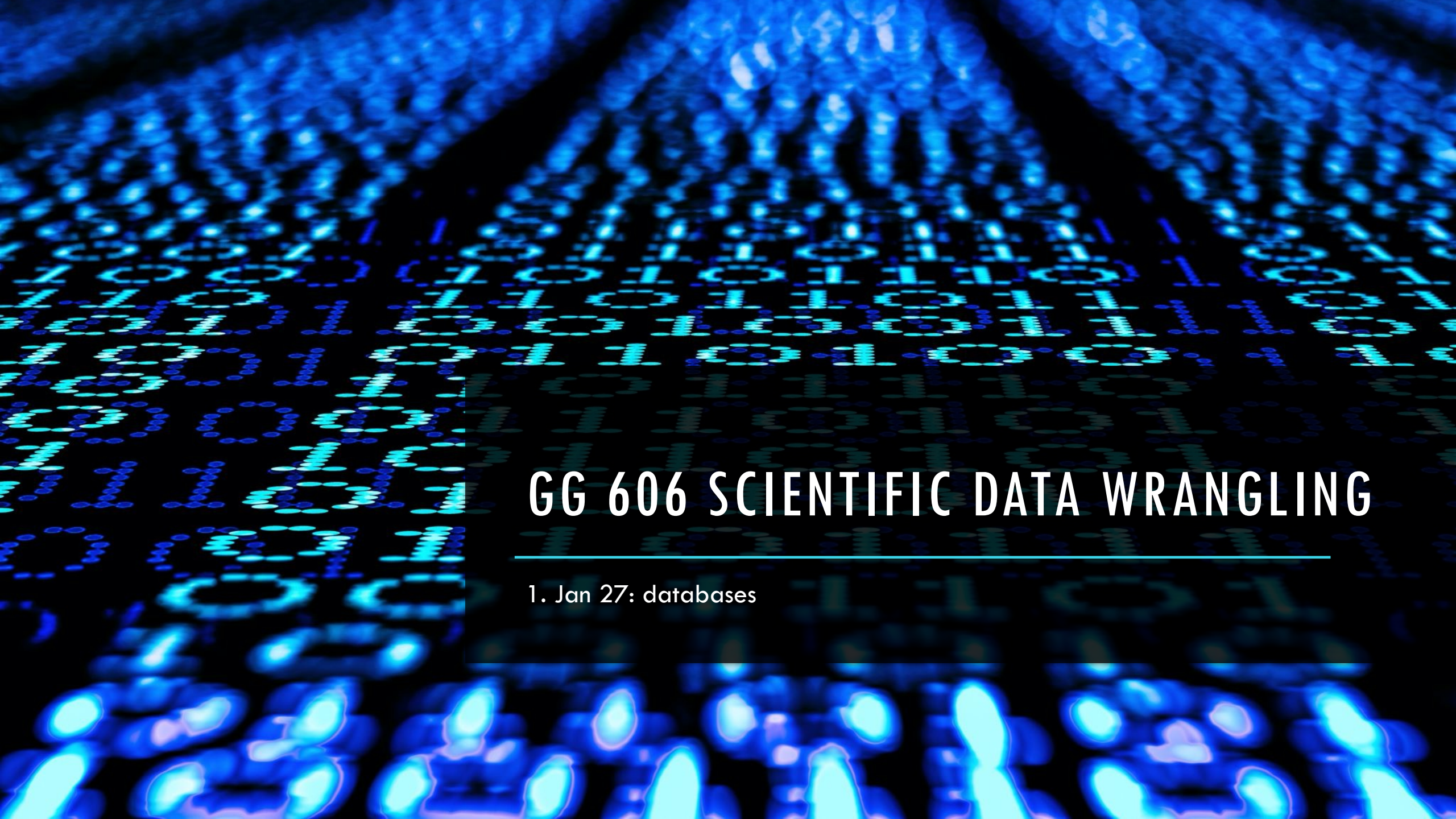
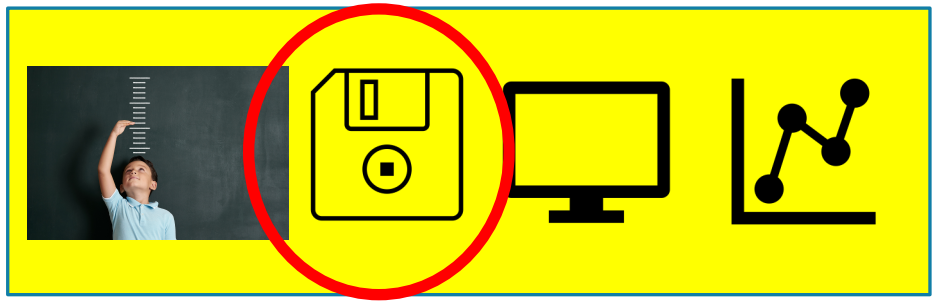


GG 606 SCIENTIFIC DATA WRANGLING

1. Jan 27: databases

WHY LEARN ABOUT DATABASES IN R

- Almost all *real world* data systems use some version of relational datbases for data storage and management
- Classic R workflow meant for small flat files has limitations
  - requires storing all data in a file
  - have to read entire dataset into memory
    - major limitations for big data analysis
- Database-connections circumvent the step of writing data out to a local file



FLAT FILES

- Contains all data in a large file
  - **Usually** stored as text
- Format is very inflexible (difficult to modify database structure)
- An ESRI interchange file (.e00 file) is a flat file (all data are stored in one file)
- DBFs are binary flat files
- CSV files are text-based flat files

FLAT FILE ‘DATABASE’

```
10118,10024,TST GF 01 GRD 02,42,4150,0,290.49525413377779,0,20,20,25,25.000437393120837,3 Inch Core,,,
10201,10024,TST GF 01 GRD 02,42,4150,4150,290.49525413377779,290.49525413377779,20,20,25,25.0004373931
10289,10017,TST GF 01 GRD 02,38,23584,23584,2702.5348482671643,2702.5348482671643,36.1875,36.1875,58.2
10390,10016,TST GF 01 GRD 01,36,5000,5000,273.74552777666668,273.74552777666668,18.25,18.25,26.9708733
10461,10016,TST GF 01 GRD 01,36,5000,5000,273.74552777666668,273.74552777666668,18.25,18.25,26.9708733
10540,10016,,36,0,0,0,0,18.25,18.25,26.970873356204589,0,3 Inch Core,,,,-32768,Mar  7 2003  4:03PM,,2,
10605,10010,TST GF 01 GRD 01,36,23507,23507,2551.9369961077359,2551.9369961077359,36.1875,36.1875,58.2
10702,10014,TST GF 01 GRD 01,36,23659,0,3548.7920228318667,0,50,50,58.25,0,3 Inch Core,,,,-32765,Mar 2
10786,10010,TST GF 01 GRD 01,36,23659,23659,2568.4382265245636,2568.4382265245636,36.1875,36.1875,58.2
10955,10010,TST GF 01 GRD 01,36,23659,23659,2568.4382265245636,2568.4382265245636,36.1875,36.1875,58.2
11069,10010,TST GF 01 GRD 01,36,23659,23659,2568.4382265245636,2568.4382265245636,36.1875,36.1875,58.2
11153,10010,TST GF 01 GRD 01,36,23507,23507,2551.9369961077359,2551.9369961077359,36.1875,36.1875,58.2
11231,10014,TST GF 01 GRD 01,36,23507,23507,3525.9923953129337,3525.9923953129337,50,50,58.25,58.06284
11321,10014,TST GF 01 GRD 01,36,23507,23507,3525.9923953129337,3525.9923953129337,50,50,58.25,58.06284
11438,10021,TST GF 01 GRD 01,36,23394,23394,3509.0426722232,3509.0426722232,50,50,58.25,58.01615543717
11506,10020,TST GF 01 GRD 01,36,23584,23584,1432.7045936818561,1432.7045936818561,20.25,20.25,58.25,58
11665,10020,TST GF 01 GRD 01,36,23584,23584,1432.7045936818561,1432.7045936818561,20.25,20.25,58.25,58
11748,10020,TST GF 01 GRD 01,36,23584,23584,1432.7045936818561,1432.7045936818561,20.25,20.25,58.25,58
11824,10014,TST GF 01 GRD 01,36,23659,23659,3548.7920228318667,3548.7920228318667,50,50,58.25,58.24953
11932,10014,TST GF 01 GRD 01,36,23659,23659,3548.7920228318667,3548.7920228318667,50,50,58.25,58.24953
12022,10014,TST GF 01 GRD 01,36,23659,23659,3548.7920228318667,3548.7920228318667,50,50,58.25,58.24953
12143,10010,TST GF 01 GRD 01,36,23659,23659,2568.4382265245636,2568.4382265245636,36.1875,36.1875,58.2
12233,10027,TST GF 01 GRD 01,36,23659,23659,2874.5215384938119,2874.5215384938119,40.5,40.5,58.0625,58
12329,10017,TST GF 01 GRD 01,36,23584,23584,2560.2961720425765,2560.2961720425765,36.1875,36.1875,58.2
12419,10021,TST GF 01 GRD 01,36,23584,23584,3537.5422066218671,3537.5422066218671,50,50,58.25,58.25035
12545,10021,TST GF 01 GRD 01,36,23584,23584,3537.5422066218671,3537.5422066218671,50,50,58.25,58.25035
12629,10021,TST GF 01 GRD 01,36,23584,23584,3537.5422066218671,3537.5422066218671,50,50,58.25,58.25035
12732,10017,TST GF 01 GRD 01,36,23584,23584,2560.2961720425765,2560.2961720425765,36.1875,36.1875,58.2
12828,10028,TST GF 01 GRD 01,36,23584,23584,2865.4091873637121,2865.4091873637121,40.5,40.5,58.015625,
```

DATABASE MANAGEMENT SYSTEM (DBMS)

- Collection of related data stored together with controlled redundancy to serve one or more applications
- A DBMS manages attribute data in separate data structures
- GIS requires traditional database functions PLUS spatial data handling functions

FUNCTIONS OF A DBMS

- Creation, modification, deletion of DB structure
- Creating, updating, deleting records
- Extracting Information
  - Sorting
  - Summarizing
  - Querying
- Maintenance of data security and integrity
- Serves application(s)
- Simultaneous multi-user access

DBMS VS FILE BASED SYSTEMS

- Collecting data at a single location reduces redundancy and duplication
- Lower maintenance cost due to better organization and decreased data duplication
- Multiple applications can use the same data and can evolve separately over time

DBMS VS FILE BASED SYSTEMS

- User knowledge can be transferred between applications more easily because database remains constant
- Facilitates data sharing/editing
- Security and standards for data and data access can be established and enforced
  - Database roles
- Downside of DBMS: structure of data must be known beforehand!! Difficult for unstructured data

RELATIONAL DATABASE (RDBMS)

- Most common/widely used (in GIS and otherwise)
- Data are stored as ordered records or rows of attribute values (rows also called *records* or *tuples*)
- Each column represents data for a single attribute for the entire dataset
- Queries for individual tables **or multiple tables**
- **Each row MUST be unique**

THE CONCEPT OF KEYS

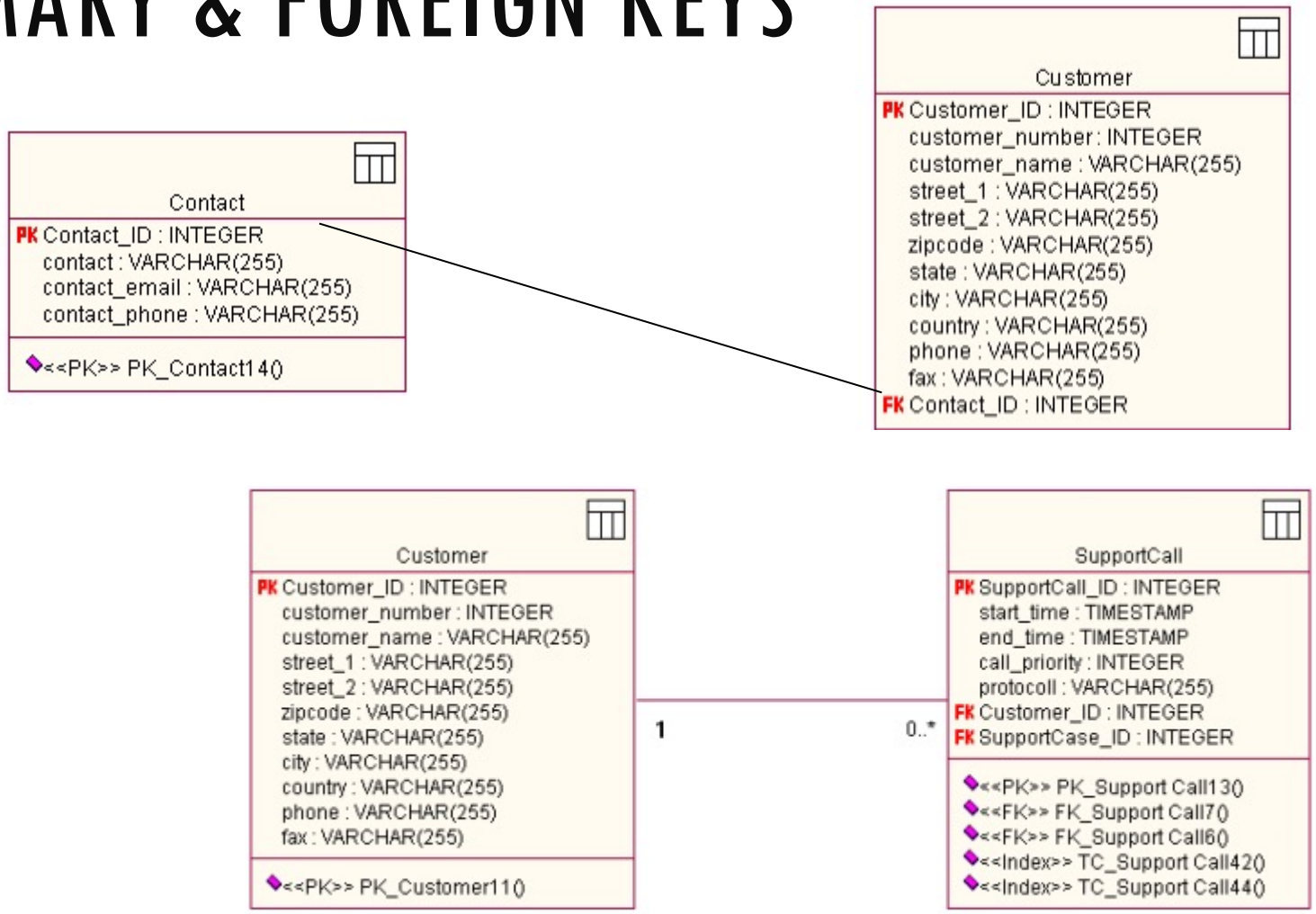
- A key is a column (or columns) in a table that satisfies certain constraints
  - e.g., must be unique
  - e.g., must be non NULL

- Two types of keys are extremely important to RDBMS

- 1) Primary Keys
  - think of this as an ID column in a table (e.g., uniquely identify each row)
  - constraints: unique, non-null
  - can be composed of multiple columns

- 2) Foreign Keys
  - a column that is a primary key in another table
  - **this creates a relationship between the two tables***

PRIMARY & FOREIGN KEYS



Data types are software dependent!

General types are: Doubles, Float, Integer, Text, Date, Geometry (in a spatial database)

RELATIONAL DATABASE

- Advantages
  - Simple organization of data
  - Capable of doing relational joins, **as long as there is at least one column common to the tables to be joined**
  - Flexibility in design and querying

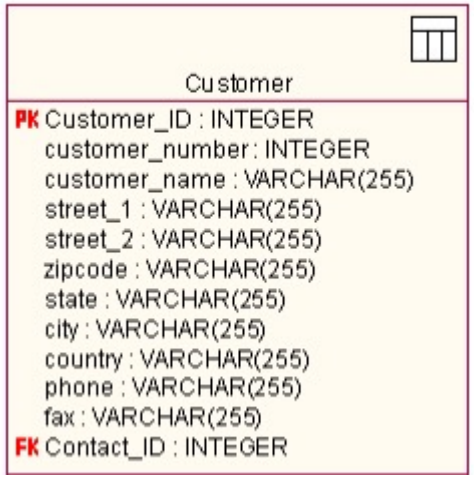
- Disadvantages
  - Querying across multiple tables with many complex joins can be slow (and complicated)
  - Need to know the structure of the data before-hand

TABLE JOINS

- Tables are joined together using common columns
  - E.g, parcel ID in owners table and parcel ID in parcel table
- After joining two or more tables, a new table is created which contains all the values of the joined tables
- Database tables can be joined together to create new relations, or views of the database

SQL (STANDARD QUERY LANUAGE)

- Two core components of SQL
  - 1) schema level → Data Definition Language
    - gives users the ability to design the logical structure of the database without reference to the underlying physical data storage or access paths
      - `CREATE TABLE, DROP TABLE, ALTER TABLE, CREATE VIEW, DROP VIEW`
  - 2) data level → Data Manipulation Language
    - access to data and relationships between records
      - `SELECT, UPDATE, INSERT`
- Separation of logical and physical view of the database

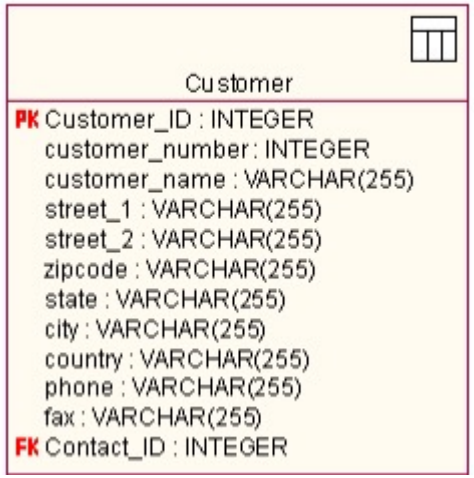


SQL (STANDARD QUERY LANUAGE)

SELECT column names

FROM table names

WHERE attribute conditions



Find the contact phone number for the employee responsible for customer ‘Henrik Sedin”

```
SELECT contact.contact_phone

FROM   contact, customer

WHERE  contact.Contact_ID = customer.Contact_ID
```

AND …

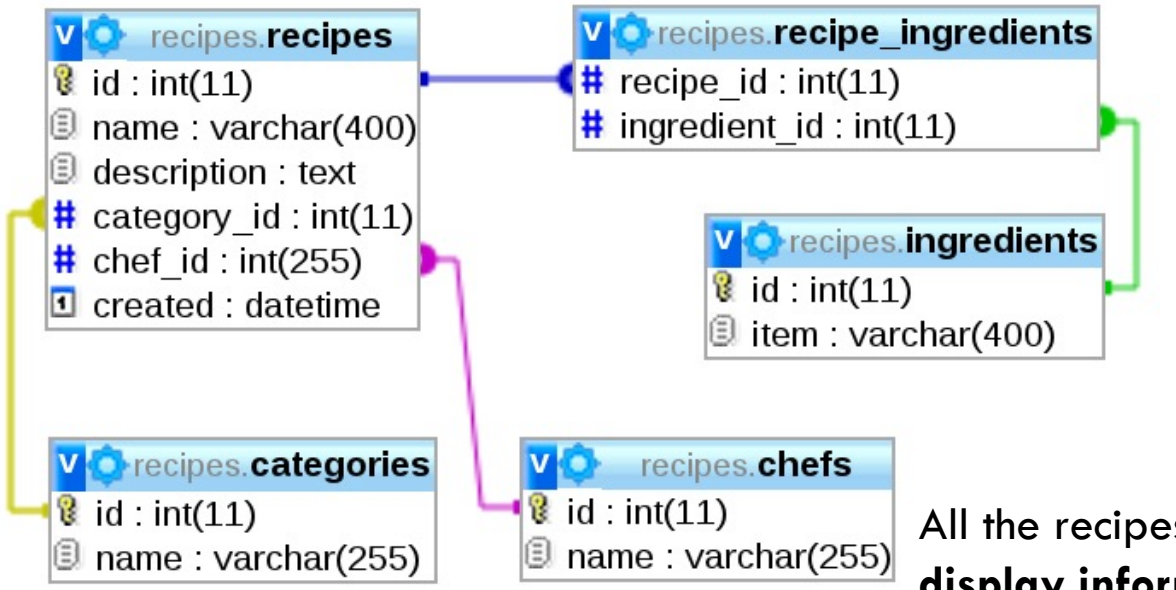
QUERIES

- Queries are built up expressions based on relational algebra using commands such as
  - `SELECT`
  - `UDPATE`
  - `CREATE`
- Used to extract subsets of information from a table or multiple tables (through joining on common keys)
- Use the = sign in the WHERE clause to join columns in multiple tables

SQL CONSTRUCTS

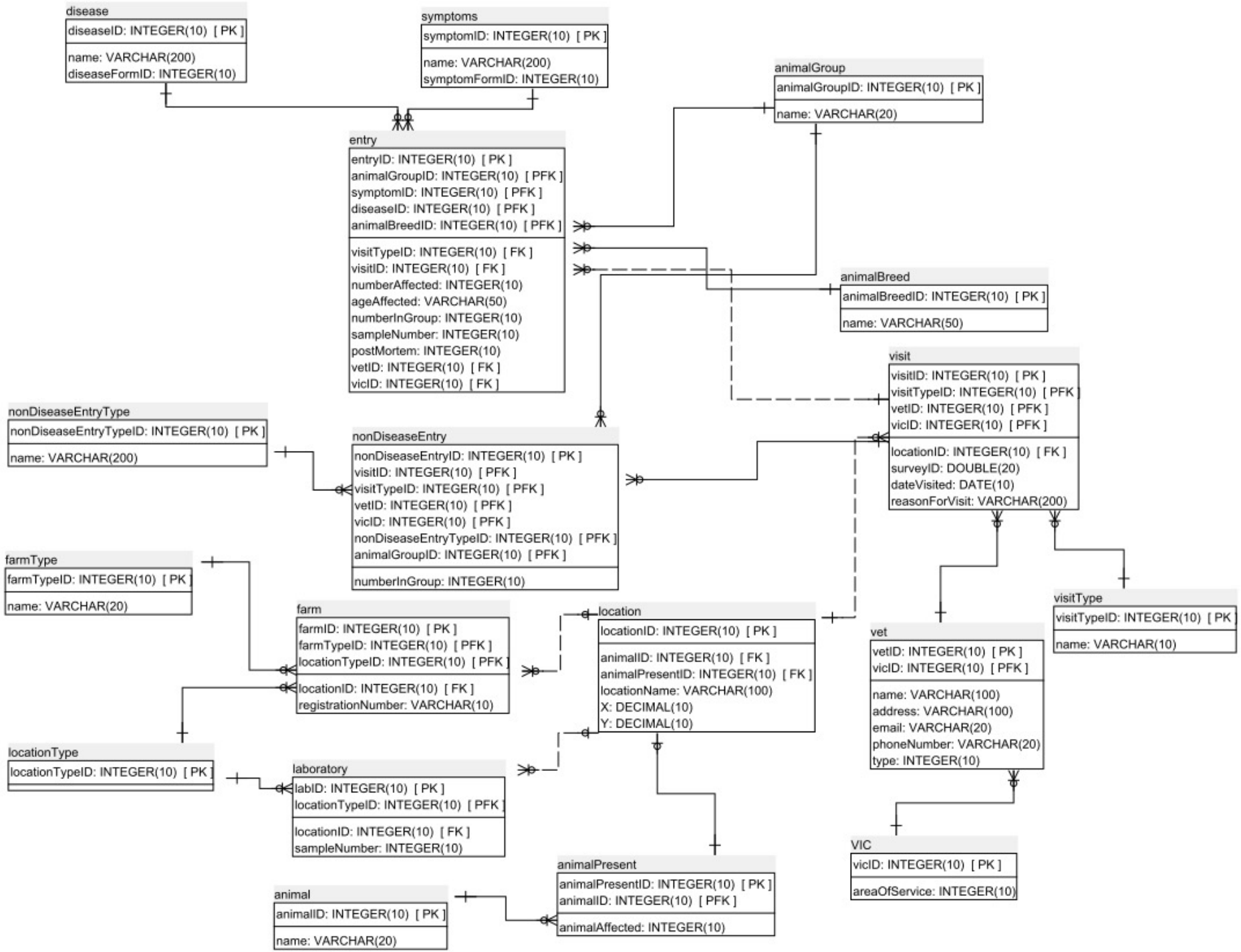
http://www.tutorialspoint.com/sql/pdf/sql-logical-operators.pdf

- STATEMENTS
  - `SELECT, INSERT, UPDATE, DELETE, ALTER, DROP, CREATE`
- OPERATORS
  - Arithmetic → +, -, *, /, %
  - Comparison → =,!=, <>, >, <, >=, <=, !<, !>
  - Logical → `AND, ANY, BETWEEN, EXISTS, IN, LIKE, NOT, OR`
- FUNCTIONS
  - Aggregate → `COUNT, MAX, MIN, AVG, SUM`
  - Numeric → `STD, SQRT, EXP, COS, SIN`
  - String → `LENGTH, RIGHT, LEFT, LOWER, UPPER`



All the recipes, their ingredients, and **the actual display information** for those ingredients

```
mysql> SELECT r.id AS recipe_id, r.name, ri.ingredient_id, i.item
FROM recipes r
JOIN recipe_ingredients ri ON (r.id = ri.recipe_id)
JOIN ingredients i ON (ri.ingredient_id = i.id);
+-----------+---------------+---------------+--------------+
| recipe_id | name          | ingredient_id | item         |
+-----------+---------------+---------------+--------------+
|         1 | Apple Crumble |             1 | apple        |
|         1 | Apple Crumble |             5 | flour        |
|         1 | Apple Crumble |             7 | butter       |
|         1 | Apple Crumble |             8 | sugar        |
|         2 | Fruit Salad   |             6 | fruit juice  |
|         2 | Fruit Salad   |             2 | banana       |
|         2 | Fruit Salad   |             1 | apple        |
|         2 | Fruit Salad   |             3 | kiwi fruit   |
|         2 | Fruit Salad   |             4 | strawberries |
+-----------+---------------+---------------+--------------+
9 ROWS IN SET (0.00 sec)
```


disease
diseaseID: INTEGER(10) [PK]
name: VARCHAR(200)
diseaseFormID: INTEGER(10)
symptoms
symptomID: INTEGER(10) [PK]
name: VARCHAR(200)
symptomFormID: INTEGER(10)
animalGroup
animalGroupID: INTEGER(10) [PK]
name: VARCHAR(20)
entry
entryID: INTEGER(10) [PK]
animalGroupID: INTEGER(10) [PFK]
symptomID: INTEGER(10) [PFK]
diseaseID: INTEGER(10) [PFK]
animalBreedID: INTEGER(10) [PFK]
visitTypeID: INTEGER(10) [FK]
visitID: INTEGER(10) [FK]
numberAffected: INTEGER(10)
ageAffected: VARCHAR(50)
numberInGroup: INTEGER(10)
sampleNumber: INTEGER(10)
postMortem: INTEGER(10)
vetID: INTEGER(10) [FK]
vicID: INTEGER(10) [FK]
animalBreed
animalBreedID: INTEGER(10) [PK]
name: VARCHAR(50)
visit
visitID: INTEGER(10) [PK]
visitTypeID: INTEGER(10) [PFK]
vetID: INTEGER(10) [PFK]
vicID: INTEGER(10) [PFK]
locationID: INTEGER(10) [FK]
surveyID: DOUBLE(20)
dateVisited: DATE(10)
reasonForVisit: VARCHAR(200)
nonDiseaseEntryType
nonDiseaseEntryTypeID: INTEGER(10) [PK]
name: VARCHAR(200)
nonDiseaseEntry
nonDiseaseEntryID: INTEGER(10) [PK]
visitID: INTEGER(10) [PFK]
visitTypeID: INTEGER(10) [PFK]
vetID: INTEGER(10) [PFK]
vicID: INTEGER(10) [PFK]
nonDiseaseEntryTypeID: INTEGER(10) [PFK]
animalGroupID: INTEGER(10) [PFK]
numberInGroup: INTEGER(10)
farmType
farmTypeID: INTEGER(10) [PK]
name: VARCHAR(20)
farm
farmID: INTEGER(10) [PK]
farmTypeID: INTEGER(10) [PFK]
locationTypeID: INTEGER(10) [PFK]
locationID: INTEGER(10) [FK]
registrationNumber: VARCHAR(10)
location
locationID: INTEGER(10) [PK]
animalID: INTEGER(10) [FK]
animalPresentID: INTEGER(10) [FK]
locationName: VARCHAR(100)
X: DECIMAL(10)
Y: DECIMAL(10)
vet
vetID: INTEGER(10) [PK]
vicID: INTEGER(10) [PFK]
name: VARCHAR(100)
address: VARCHAR(100)
email: VARCHAR(20)
phoneNumber: VARCHAR(20)
type: INTEGER(10)
visitType
visitTypeID: INTEGER(10) [PK]
name: VARCHAR(10)
locationType
locationTypeID: INTEGER(10) [PK]
laboratory
labID: INTEGER(10) [PK]
locationTypeID: INTEGER(10) [PFK]
locationID: INTEGER(10) [FK]
sampleNumber: INTEGER(10)
animalPresent
animalPresentID: INTEGER(10) [PK]
animalID: INTEGER(10) [PFK]
animalAffected: INTEGER(10)
VIC
vicID: INTEGER(10) [PK]
areaOfService: INTEGER(10)
animal
animalID: INTEGER(10) [PK]
name: VARCHAR(20)

CONNECTING TO DATABASES FROM R

- The DBI (**D**ata**B**ase **I**nterface) package provides a simple, consistent interface between R and database management systems (DBMS)
- Each supported DBMS is supported by its own R package that implements the DBI specification
- Examples of supported DBMS include
  - MySQL
  - Postgres
  - SQLite
  - ~30 others

this can greatly expand the reach of R-based analysis…

CONNECTING TO DATABASES FROM R

1. Open up a connection object by specifying parameters for the `dbConnect` function

2. Use the connection object with other functions to see tables, query, etc.

3. Close the connection object with `dbDisconnect`

```
library(DBI)

con <- dbConnect(
  RMariaDB::MariaDB(),
  host = "relational.fit.cvut.cz",
  port = 3306,
  username = "guest",
  password = "relational",
  dbname = "sakila"
)

dbListTables(con)
```

which database driver to use

```
##  [1] "actor"         "address"       "category"      "city"
##  [5] "country"       "customer"      "film"          "film_actor"
##  [9] "film_category" "film_text"     "inventory"     "language"
## [13] "payment"       "rental"        "staff"         "store"
```

```
dbDisconnect(con)
```

READ A TABLE INTO A DATA FRAME

- `dbReadTable()` reads an entire table and returns it as a data frame
- It is equivalent to the SQL query `SELECT * FROM <name>`
- DBI and the database backends attempt to coerce data to equivalent R data types though these should always be verified

QUERY A DATABASE TABLE

- `dbGetQuery()` passes an SQL query to the database referenced in the connection object

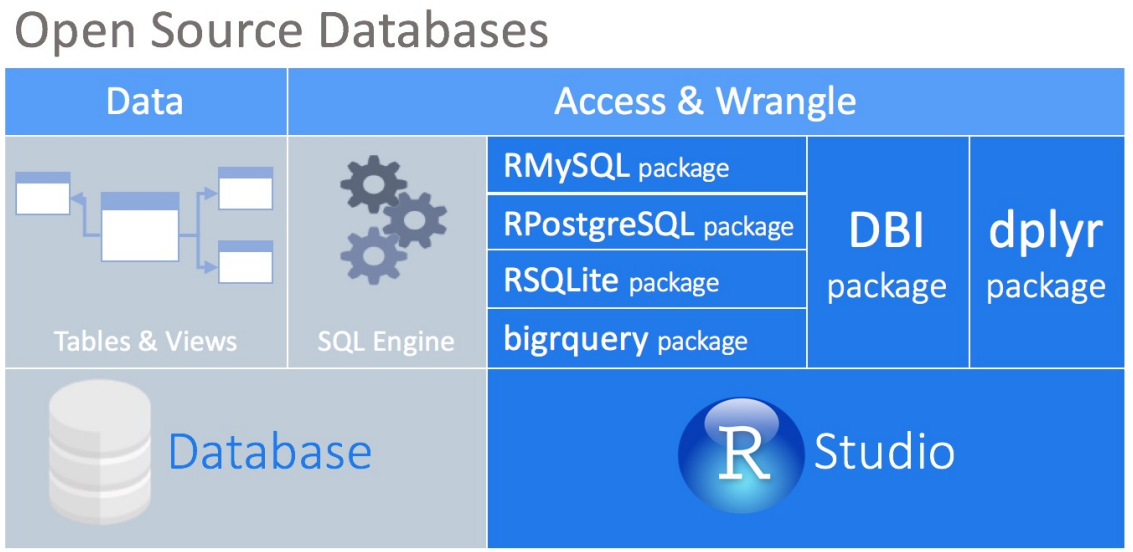
```
con <- dbConnect(RMariaDB::MariaDB(), username = "guest", password = "relational", host = "relational.fit.cvut.cz",
port = 3306, dbname = "sakila")
dbListFields(con, "film")
 [1] "film_id"              "title"                "description"          "release_year"         "language_id"
 [6] "original_language_id" "rental_duration"      "rental_rate"          "length"               "replacement_cost"
[11] "rating"               "special_features"     "last_update"
dbListTables(con)
 [1] "actor"         "address"       "category"      "city"          "country"       "customer"      "film"
 [8] "film_actor"    "film_category" "film_text"     "inventory"     "language"      "payment"       "rental"
[15] "staff"         "store"
df <- dbGetQuery(con, "SELECT film_id, title, description FROM film WHERE release_year = 2006")
head(df, 3)
  film_id            title
1       1 ACADEMY DINOSAUR
2       2   ACE GOLDFINGER
3       3 ADAPTATION HOLES
```

TIDY DATABASE ACCESS W/ dplyr

dplyr is able to interact with databases directly by translating the dplyr verbs into SQL queries

- Run data exploration routines over all of the data, instead of importing part of the data into R.
- computation is being pushed to the database.
- collect into R only a targeted dataset.
- All of your code is in R. Becausedplyr is used to communicate with the database, there is no need to alternate between languages or tools to perform the data exploration

- dplyr package depends on the DBI package for communication with databases



LAZY EXECUTION W/ dplyr

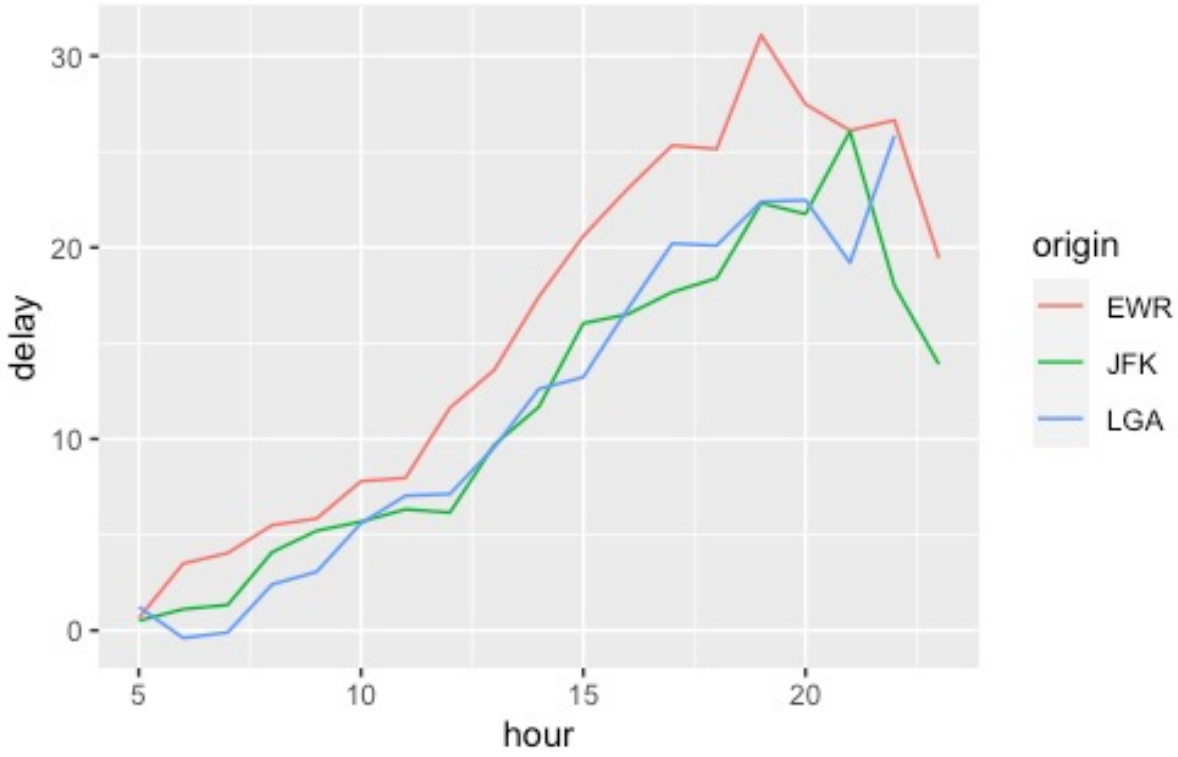
- your R code is translated into SQL and executed in the database on the remote server, not in R on your local machine
- dplyr tries to be as lazy as possible:
  - It never pulls data into R unless you explicitly ask for it.
  - It delays doing any work until the last possible moment: it collects together everything you want to do and then sends it to the database in one step.

```
tailnum_delay_db <- flights_db %>%
  group_by(tailnum) %>%
  summarise(
    delay = mean(arr_delay),
    n = n()
  ) %>%
  arrange(desc(delay)) %>%
  filter(n > 100)
```

Nothing is execute against the database yet, an SQL statement is built up, and can be viewed by calling `show_query()`

EXAMPLE – SQLITE DATABASE

```
library(DBI)
library(dplyr)
library(ggplot2)
# SQLite database
con <- dbConnect(RSQLite::SQLite(), ":memory:")
copy_to(con, nycflights13::flights, "FLIGHTS")
# ODBC databases (requires a live database connection)
#con <- dbConnect(odbc::odbc(), "SQL Server")
#con <- dbConnect(odbc::odbc(), "Snowflake")
# Query, collect results, and visualize
tbl(con, "FLIGHTS") %>% filter(distance > 75) %>%
group_by(origin, hour) %>% summarise(delay =
mean(dep_delay, na.rm = TRUE)) %>% collect() %>%
ggplot(aes(hour, delay, color = origin)) + geom_line()
```



DEMO:DGGS ANALYTICS SYSTEM

- https://thespatiallabatlaurier.github.io/nzdggs/Examples/GettingStarted/
- https://thespatiallabatlaurier.github.io/nzdggs/Examples/IDEAS/IDEAS-spatial-overlay/

REFERENCES AND ADDITIONAL RESOURCES

- R-studio documentation for database connections
  - https://db.rstudio.com/getting-started/database-queries

SUMMARY

- Connecting to database and using database as compute engine greatly expands the use cases for r-based analytics
- writing dblyr data transformation code automatically translated to databas—compliant SQL
- only read final results back to R session for final visualization or modelling, keeping bulk of data in the database
  - partially solves one of R's long-standing problems in being limited by memory size for computation

GG 606 SCIENTIFIC DATA WRANGLING

1. Jan 27: databases