Data Wrangling: Munging, Tidy Data, and Working with Multiple Data Tables (III)

Nicholas Mattei, *Tulane University*

CMPS3660 – Introduction to Data Science – Fall 2019

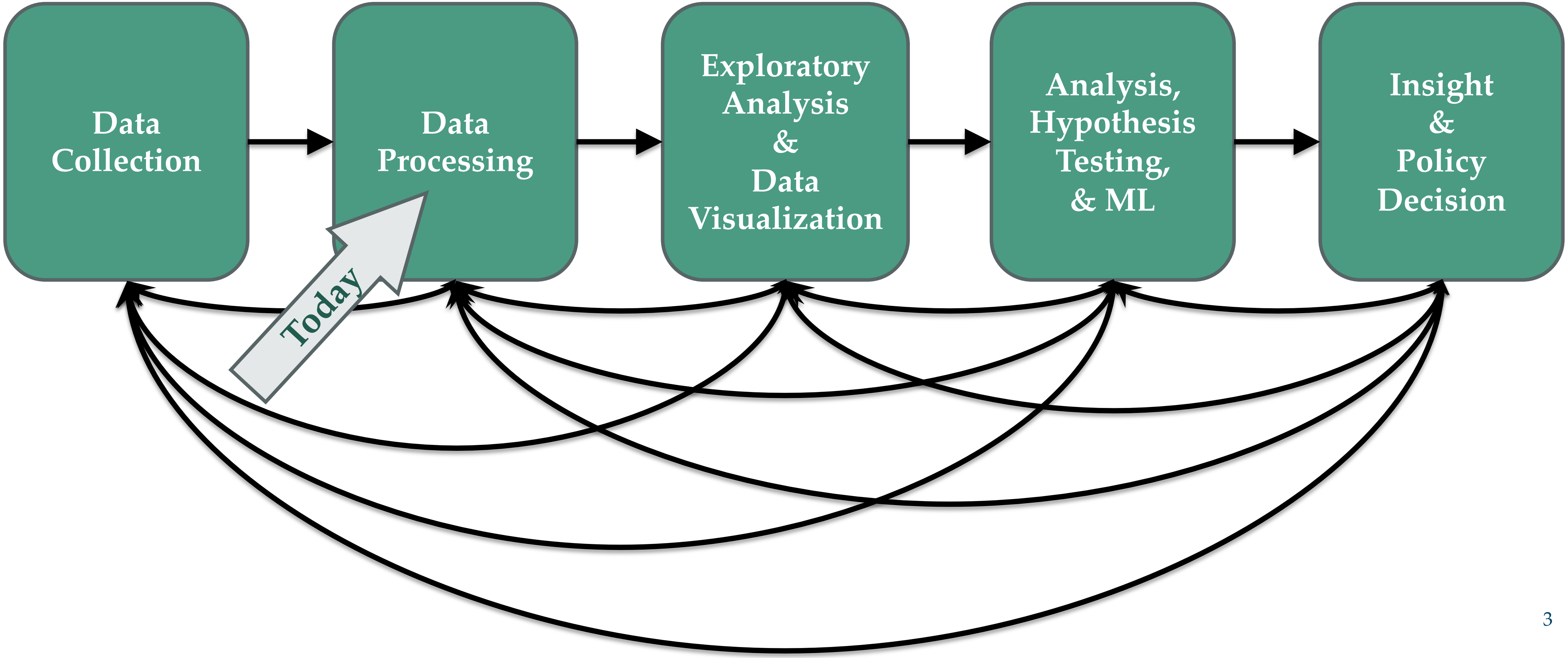
https://rebrand.ly/TUDataScience

Many Thanks
Slides based off Introduction to Data Science from John P. Dickerson -
https://cmsc320.github.io/

Announcements

- Labs Posted
- Lxml fix
- Groups access
- Merge/Join Terms
- Early Office Hours

The Data LifeCycle


Data Collection
Data Processing
Exploratory Analysis & Data Visualization
Analysis, Hypothesis Testing, & ML
Insight & Policy Decision
Today

SQL And Relational Data

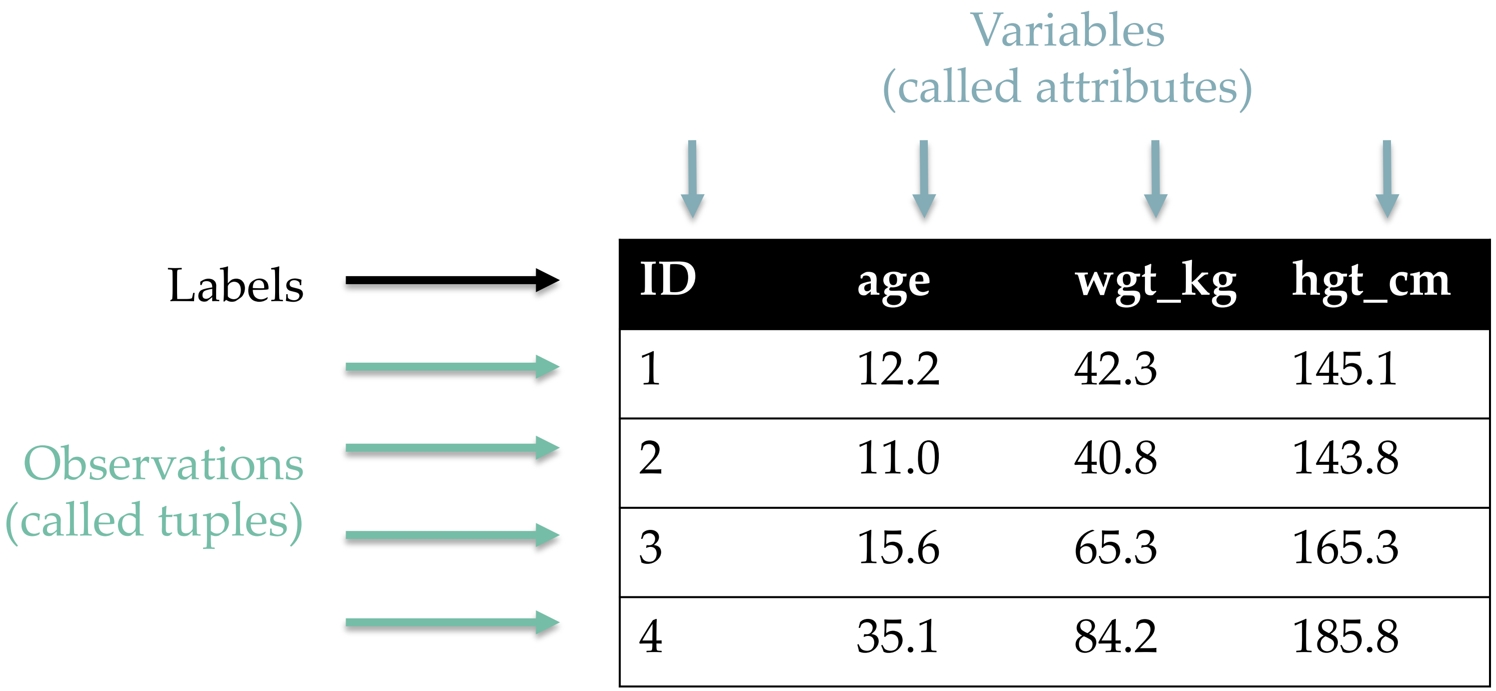
- Relational data:
  - What is a relation, and how do they interact?
- Querying databases:
  - SQL
  - SQLite
  - How does this relate to `pandas`?

- Joins in SQL

Thanks to Zico Kolter for some structure for this lecture!

Relation

- Simplest relation: a table aka tabular data full of **unique** tuples



Variables
(called attributes)

Labels

Observations
(called tuples)

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

Primary keys

ID	age	wgt_kg	hgt_cm	nat_id
1	12.2	42.3	145.1	1
2	11.0	40.8	143.8	1
3	15.6	65.3	165.3	2
4	35.1	84.2	185.8	1
5	18.1	62.2	176.2	3
6	19.6	82.1	180.1	1

ID	Nationality
1	USA
2	Canada
3	Mexico

- The primary key is a unique identifier for every tuple in a relation.
  - Each tuple has exactly one primary key

Wait, Aren’t These Called “indexes”?

- Yes, in Pandas; but not in the database world

- For most databases, an “index” is a data structure used to speed up retrieval of specific tuples

- For example, to find all tuples with nat_id = 2:
  - We can either scan the table – O(N)
  - Or use an “index” (e.g., binary tree) – O(log N)

Foreign keys

ID	age	wgt_kg	hgt_cm	nat_id
1	12.2	42.3	145.1	1
2	11.0	40.8	143.8	1
3	15.6	65.3	165.3	2
4	35.1	84.2	185.8	1
5	18.1	62.2	176.2	3
6	19.6	82.1	180.1	1

ID	Nationality
1	USA
2	Canada
3	Mexico

- Foreign keys are attributes (columns) that point to a different table's primary key.
  - A table can have multiple foreign keys

Relation Schema

- A list of all the attribute names, and their *domains*

```
create table department
   (dept_name varchar(20),
    building varchar(15),
    budget numeric(12,2) check (budget > 0),
    primary key (dept_name)
   );
```

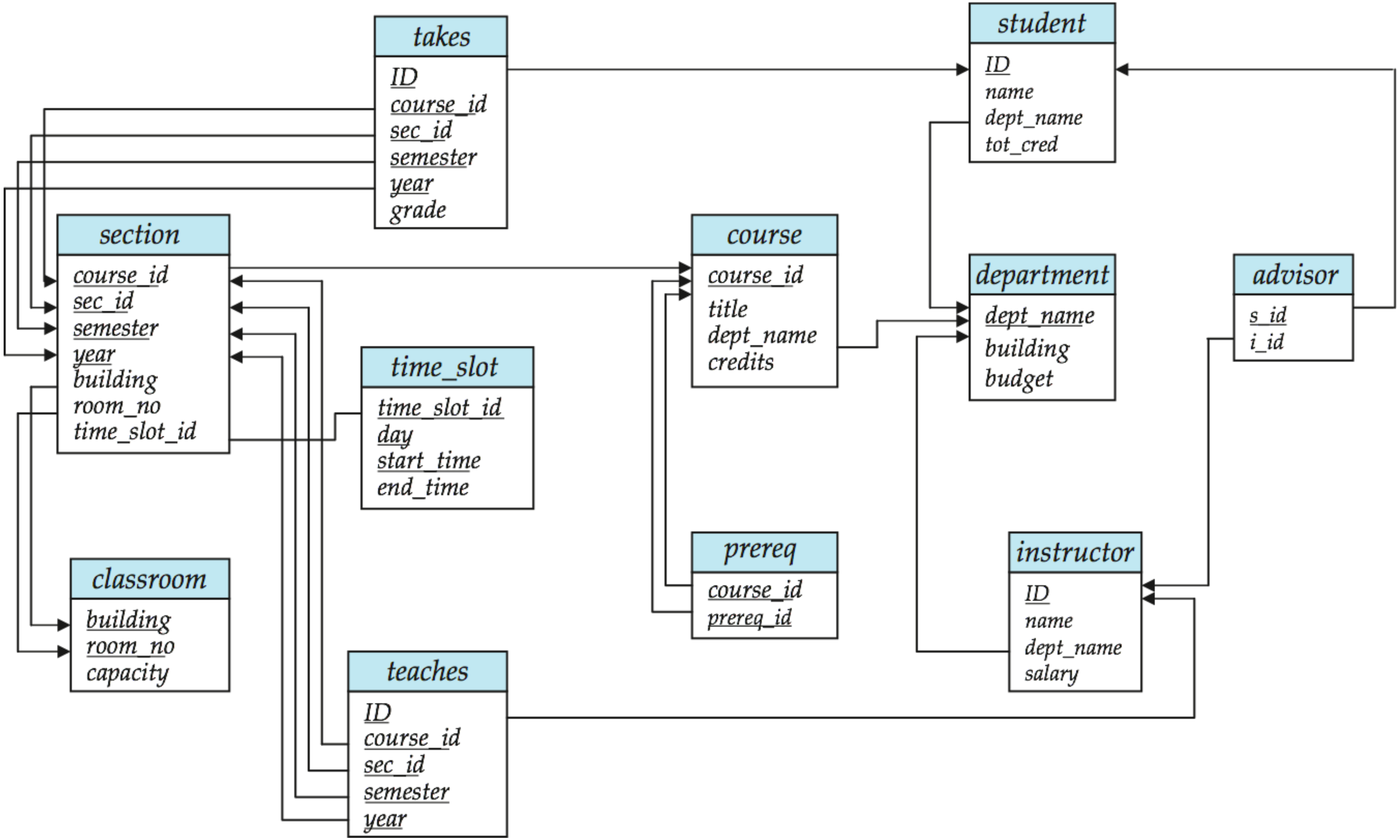
```
create table instructor (

    ID       char(5),
    name   varchar(20) not null,
    dept_name  varchar(20),
    salary   numeric(8,2),

    primary key (ID),
    foreign key (dept_name) references department
)
```

SQL Statements
To create Tables

Schema Diagrams



takes
ID
course_id
sec_id
semester
year
grade

student
ID
name
dept_name
tot_cred

section
course_id
sec_id
semester
year
building
room_no
time_slot_id

time_slot
time_slot_id
day
start_time
end_time

course
course_id
title
dept_name
credits

department
dept_name
building
budget

advisor
s_id
i_id

prereq
course_id
prereq_id

instructor
ID
name
dept_name
salary

classroom
building
room_no
capacity

teaches
ID
course_id
sec_id
semester
year

Searching for elements

- Find all people with nationality Canada (nat_id = 2):
  - ???????????????

ID	age	wgt_kg	hgt_cm	nat_id
1	12.2	42.3	145.1	1
2	11.0	40.8	143.8	1
3	15.6	65.3	165.3	2
4	35.1	84.2	185.8	1
5	18.1	62.2	176.2	3
6	19.6	82.1	180.1	1

O(n) ☹

Indexes

- Like a hidden sorted map of references to a specific attribute (column) in a table.
  - Allows O(log n) lookup instead of O(n)

loc	ID	age	wgt_kg	hgt_cm	nat_id
0	1	12.2	42.3	145.1	1
128	2	11.0	40.8	143.8	2
256	3	15.6	65.3	165.3	2
384	4	35.1	84.2	185.8	1
512	5	18.1	62.2	176.2	3
640	6	19.6	82.1	180.1	1

nat_id	locs
1	0, 384, 640
2	128, 256
3	512

INdexes

- Actually implemented with data structures like B-trees
  - In a full Databases course you would learn how to store and make these!
- But: indexes are not free
  - Takes memory to store
  - Takes time to build
  - Takes time to update (add/delete a row, update the column)
- But, but: one index is (mostly) free
  - Index will be built automatically on the primary key
- **Think before you build/maintain an index on other attributes!**

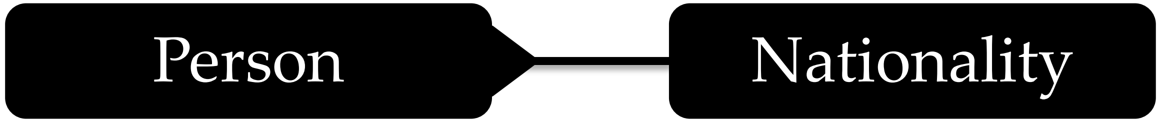
Relationships

- Primary keys and foreign keys define interactions between different tables aka entities. Four types:
  - One-to-one
  - One-to-one-or-none
  - One-to-many and many-to-one
  - Many-to-many

- Connects (one, many) of the rows in one table to (one, many) of the rows in another table

One-to-many & Many-to-one

- **One person** can have **one nationality** (in this example), but one nationality can include **many people**.



ID	age	wgt_kg	hgt_cm	nat_id
1	12.2	42.3	145.1	1
2	11.0	40.8	143.8	1
3	15.6	65.3	165.3	2
4	35.1	84.2	185.8	1
5	18.1	62.2	176.2	3
6	19.6	82.1	180.1	1

ID	Nationality
1	USA
2	Canada
3	Mexico

One-to-One

- Two tables have a one-to-one relationship if every tuple in the first tables corresponds to **exactly one** entry in the other

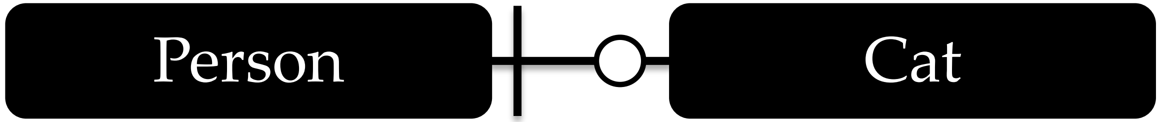
- In general, you won’t be using these (why not just merge the rows into one table?) unless:
  - Split a big row between SSD and HDD or distributed
  - Restrict access to part of a row (some DBMSs allow column-level access control, but not all)
- Caching, partitioning, & other serious stuff that we won’t cover.

One-to-One-Or-None

- Say we want to keep track of people’s cats:

Person ID	Cat1	Cat2
1	Chairman Meow	Fuzz Aldrin
4	Anderson Pooper	Meowly Cyrus
5	Gigabyte	Megabyte

- People with IDs 2 and 3 do not own cats*, and are not in the table.
  Each person has at most one entry in the table.



Person

Cat

- Is this data tidy?

*nor do they have hearts, apparently.

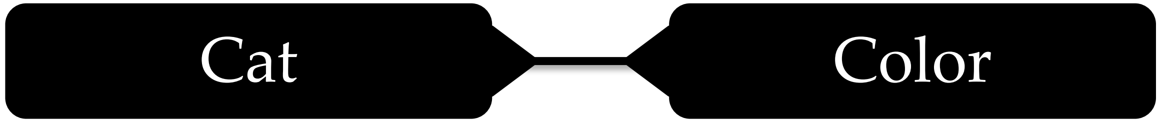
Many-to-Many

- Say we want to keep track of people's cats' colorings:

ID	Name
1	Megabyte
2	Meowly Cyrus
3	Fuzz Aldrin
4	Chairman Meow
5	Anderson Pooper
6	Gigabyte

Cat ID	Color ID	Amount
1	1	50
1	2	50
2	2	20
2	4	40
2	5	40
3	1	100

- One column per color, too many columns, too many nulls
- Each cat can have many colors, and each color many cats



Cat

Color

Associative tables

Cats

ID	Name
1	Megabyte
2	Meowly Cyrus
3	Fuzz Aldrin
4	Chairman Meow
5	Anderson Pooper
6	Gigabyte

Cat ID	Color ID	Amount
1	1	50
1	2	50
2	2	20
2	4	40
2	5	40
3	1	100

Colors

ID	Name
1	Black
2	Brown
3	White
4	Orange
5	Neon Green
6	Invisible

- Typically used to model pure relationships, not entities.
- The Primary Keys are from other tables – here we have [CatID, ColorID]
- Pros:
  - Handles one-to-one, one-to-many, and many-to-one
  - Can be added without modifying existing tables.
- Cons:
  - Requries extra joins/queries to learn certain things.

Aside: Pandas

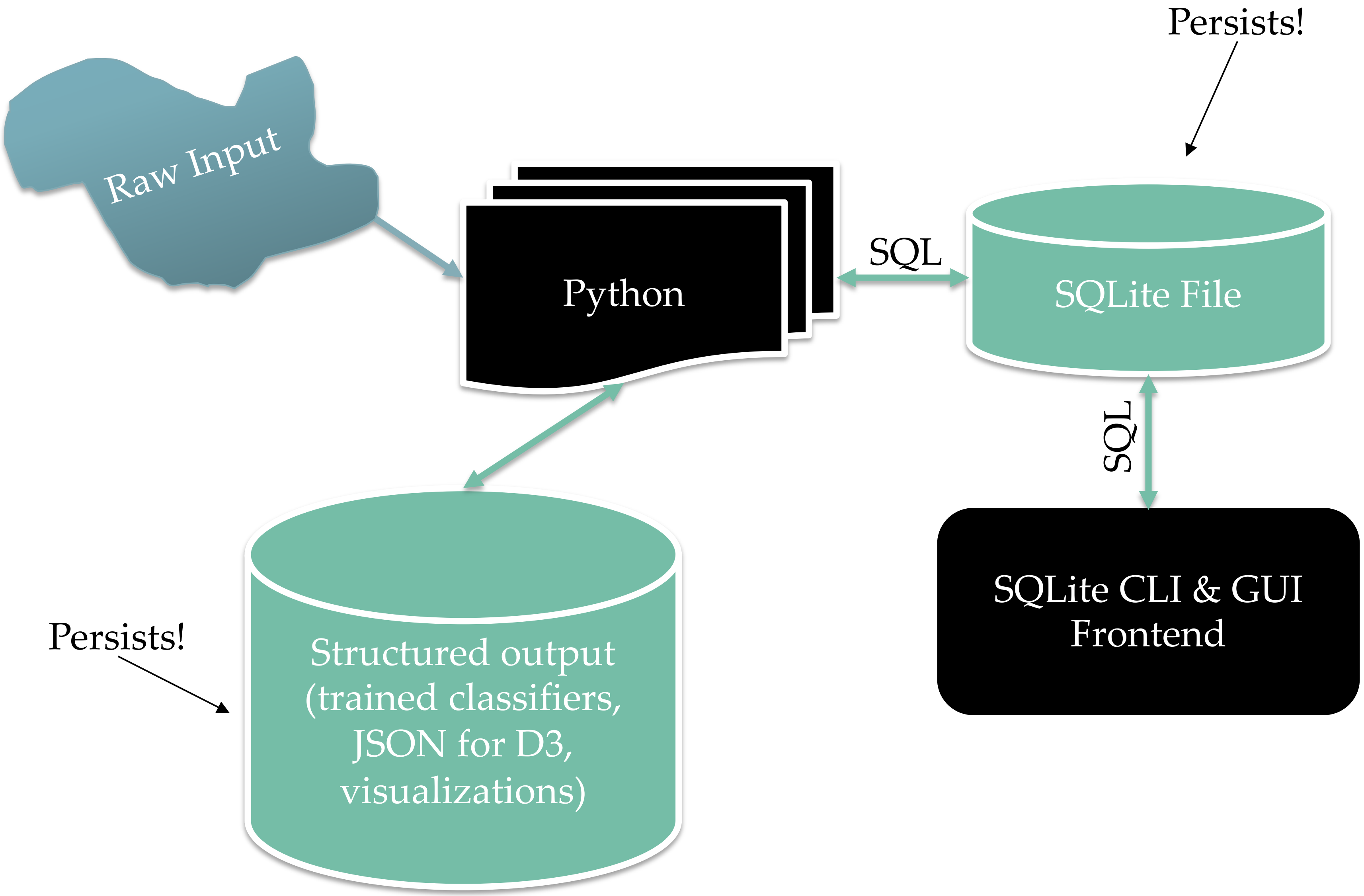
- So, this kinda feels like pandas …
  - And pandas kinda feels like a relational data system …
- Pandas is **not strictly** a relational data system:
  - No notion of primary / foreign keys
- It does have indexes (and multi-column indexes):
  - pandas.Index: ordered, sliceable set storing axis labels
  - pandas.MultiIndex: hierarchical index

- **Rule of thumb: do heavy, rough lifting at the relational DB level, then fine-grained slicing and dicing and visualization with pandas**

SQLite

- **On-disk** relational database management system (RDMS)
  - Applications connect directly to a **file.**
- Most RDMSs have applications connect to a **server:**
  - Advantages include greater concurrency, less restrictive locking
  - Disadvantages include, for this class, setup time ☺
- Installation:
  - `conda install -c anaconda sqlite`
  - (Should come preinstalled, I think?)
- All interactions use Structured Query Language (SQL)

Using a DB with Pandas!


Persists!
Raw Input
Python
SQL
SQLite File
SQL
SQLite CLI & GUI Frontend
Persists!
Structured output (trained classifiers, JSON for D3, visualizations)

Crash Course in SQL (in python)

```
import sqlite3

# Create a database and connect to it
conn = sqlite3.connect(“cmsc320.db”)
cursor = conn.cursor()

# do cool stuff
conn.close()
```

- Cursor: temporary work area in system memory for manipulating SQL statements and return values
- If you do not close the connection (`conn.close()`), any outstanding transaction is rolled back
- (More on this in a bit.)

Crash Course in SQL (in python)

```
# Make a table
cursor.execute(“””
CREATE TABLE cats (
    id INTEGER PRIMARY KEY,
    name TEXT
)”””)
```

?????????

id	name

cats

- Capitalization doesn’t matter for SQL reserved words
- SELECT = select = SeLeCt
- Rule of thumb: capitalize keywords for readability

Crash Course in SQL (in python)

```
# Insert into the table
cursor.execute(“INSERT INTO cats VALUES (1, ’Megabyte’)”)
cursor.execute(“INSERT INTO cats VALUES (2, ‘Meowly Cyrus’)”)
cursor.execute(“INSERT INTO cats VALUES (3, ‘Fuzz Aldrin’)”)
conn.commit()
```

id	name
1	Megabyte
2	Meowly Cyrus
3	Fuzz Aldrin

```
# Delete row(s) from the table
cursor.execute(“DELETE FROM cats WHERE id == 2”);
conn.commit()
```

id	name
1	Megabyte
3	Fuzz Aldrin

Crash Course in SQL (in python)

```
# Read all rows from a table
for row in cursor.execute(”SELECT * FROM cats”):
    print(row)
```

```
# Read all rows into pandas DataFrame
pd.read_sql_query(“SELECT * FROM cats”, conn, index_col=”id”)
```

id	name
1	Megabyte
3	Fuzz Aldrin

- index_col=“id”: treat column with label “id” as an index
- index_col=1: treat column #1 (i.e., “name”) as an index
- (Can also do multi-indexing.)

Joining data

- A join operation merges two or more tables into a single relation. Different ways of doing this:
- Inner
- Left
- Right
- Full Outer

- Join operations are done on columns that explicitly link the tables together

Inner Joins

id	name
1	Megabyte
2	Meowly Cyrus
3	Fuzz Aldrin
4	Chairman Meow
5	Anderson Pooper
6	Gigabyte

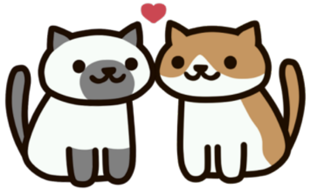
`cats`

cat_id	last_visit
1	02-16-2017
2	02-14-2017
5	02-03-2017

`visits`

- Inner join returns merged rows that share the same value in the column they are being joined on (`id` and `cat_id`).

id	name	last_visit
1	Megabyte	02-16-2017
2	Meowly Cyrus	02-14-2017
5	Anderson Pooper	02-03-2017



Inner Joins

```
# Inner join in pandas
df_cats = pd.read_sql_query(“SELECT * from cats”, conn)
df_visits = pd.read_sql_query(“SELECT * from visits”, conn)
df_cats.merge(df_visits, how = “inner”,
              left_on = “id”, right_on = ”cat_id”)
```

```
# Inner join in SQL / SQLite via Python
cursor.execute(“””
               SELECT
                   *
               FROM
                   cats, visits
               WHERE
                   cats.id == visits.cat_id
               ”””)
```

Left Joins

- Inner joins are the most common type of joins (get results that appear in both tables)
- Left joins: all the results from the left table, only some matching results from the right table
- Left join (`cats`, `visits`) on (`id`, `cat_id`) ???????????

id	name	last_visit
1	Megabyte	02-16-2017
2	Meowly Cyrus	02-14-2017
3	Fuzz Aldrin	NULL
4	Chairman Meow	NULL
5	Anderson Pooper	02-03-2017
6	Gigabyte	NULL

Right Joins

- Take a guess!
- Right join (`cats, visits`) on (`id, cat_id`) ???????????

id	name
1	Megabyte
2	Meowly Cyrus
3	Fuzz Aldrin
4	Chairman Meow
5	Anderson Pooper
6	Gigabyte

`cats`

cat_id	last_visit
1	02-16-2017
2	02-14-2017
5	02-03-2017
7	02-19-2017
12	02-21-2017

`visits`

id	name	last_visit
1	Megabyte	02-16-2017
2	Meowly Cyrus	02-14-2017
5	Anderson Pooper	02-03-2017
7	NULL	02-19-2017
12	NULL	02-21-2017

Left/Right Joins

```
# Left join in pandas
df_cats.merge(df_visits, how = “left”,
              left_on = “id”, right_on = ”cat_id”)
```

```
# Left join in SQL / SQLite via Python
cursor.execute(“SELECT * FROM cats LEFT JOIN visits ON
                   cats.id == visits.cat_id”)
```

```
# Right join in pandas
df_cats.merge(df_visits, how = “right”,
              left_on = “id”, right_on = ”cat_id”)
```

```
# Right join in SQL / SQLite via Python
☹
```

Full Outer Join

- Combines the left and the right join ???????????

id	name	last_visit
1	Megabyte	02-16-2017
2	Meowly Cyrus	02-14-2017
3	Fuzz Aldrin	NULL
4	Chairman Meow	NULL
5	Anderson Pooper	02-03-2017
6	Gigabyte	NULL
7	NULL	02-19-2017
12	NULL	02-21-2017

```
# Outer join in pandas
df_cats.merge(df_visits, how = “outer”,
              left_on = “id”, right_on = ”cat_id”)
```

Google Image Search One Slide SQL Join Visual

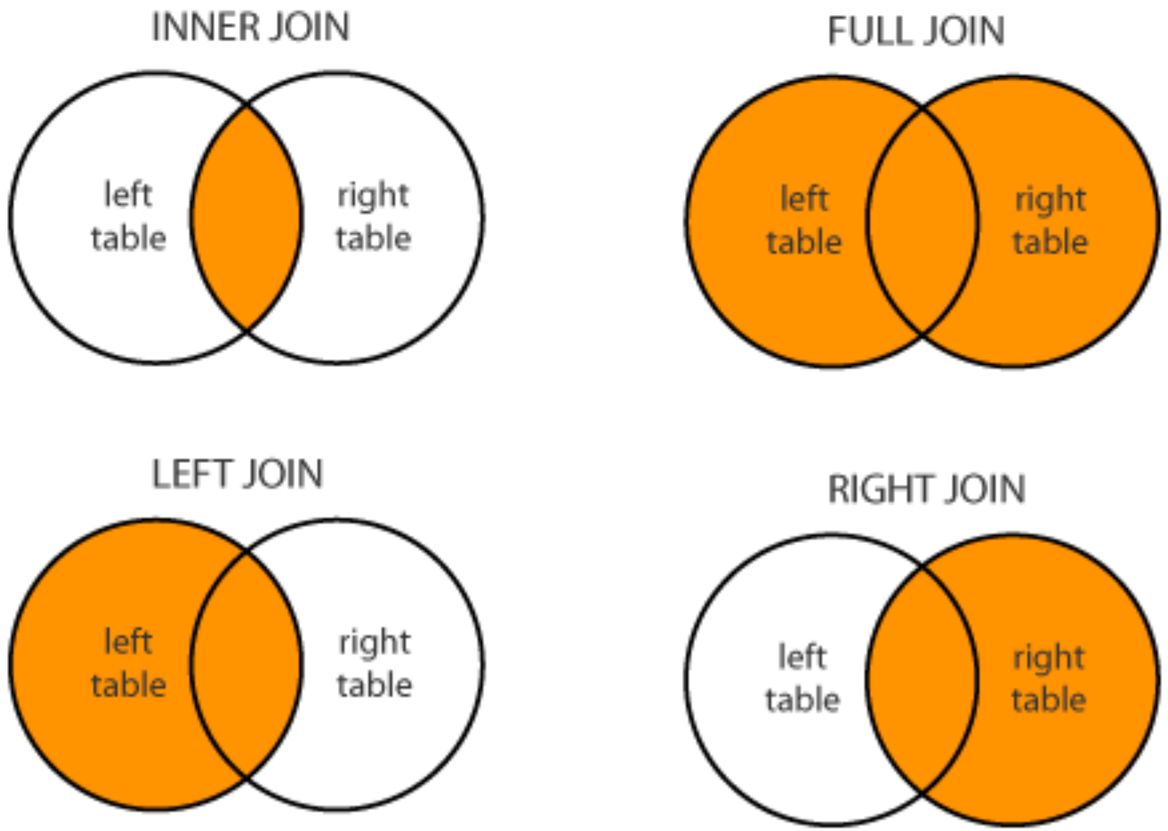


Image credit: http://www.dofactory.com/sql/join

Group by Aggregates

```
SELECT nat_id, AVG(age) as average_age
FROM persons GROUP BY nat_id
```

ID	age	wgt_kg	hgt_cm	nat_id
1	12.2	42.3	145.1	1
2	11.0	40.8	143.8	1
3	15.6	65.3	165.3	2
4	35.1	84.2	185.8	1
5	18.1	62.2	176.2	3
6	19.6	82.1	180.1	1

nat_id	average_age
1	19.48
2	15.6
3	18.1

Raw SQL in Pandas

- If you “think in SQL” already, you’ll be fine with pandas:
  - `conda install -c anaconda pandasql`
- Info: http://pandas.pydata.org/pandas-docs/stable/comparison_with_sql.html

```
# Write the query text
q = """
    SELECT
        *
    FROM
        cats
    LIMIT 10;"""

# Store in a DataFrame
df = sqldf(q, locals())
```