DATA 200: DATABASE MANAGEMENT - RECAP

- Christine Gurek, Lan Nguyen, Long Bui -

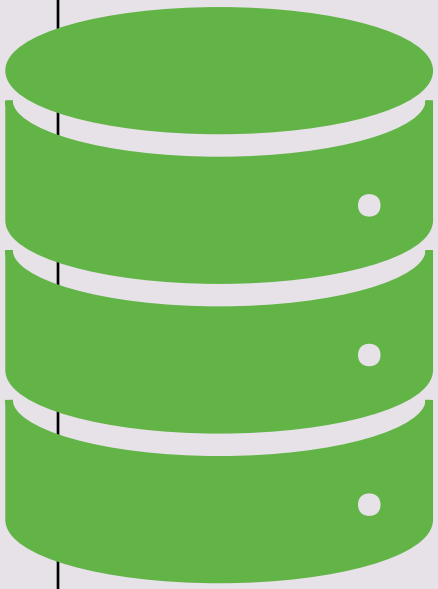
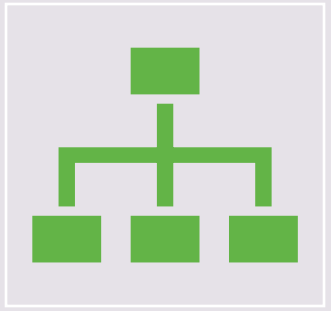


TABLE OF CONTENT

DATA SYSTEMS OVERVIEW

DATA SYSTEMS



A system is defined as

A regularly interacting or interdependent group of items forming a unified whole.

An organization forming a network for serving a common purpose.

Data systems are defined as

Systems whose focus is on the organization and structure of data

The means of which such data can be obtained from the provider of the data by the user or client of the data.

DATA SYSTEMS PRINCIPALS

Data System Provider:

- An entity, consisting of hardware, software, and possible network infrastructure, that hosts the data, provides organization, and allows access through a well-defined interface.

Data System Client:

- Local software and hardware whose goal is to acquire data from one or more providers and to manage and transform the data to make it useful for analysis.

Resource Owner:

- The entity that owns the resource (data, organizational resources, etc).

SOURCES OF DATA

- Local Files:
  - files downloaded or present in the file system.
  - Individual files have a format that specifies the structure of the data.
  - Common formats include `.txt` , `.docx` , `.csv` , and `.html` .
- Relational Databases:
  - provides data in the form of relational tables in response to queries .
  - Example systems include servers running Oracle, MySQL, and PostgreSQL.
- API (Application Programming Interface)
  - allows clients to access data over the network.
  - Automates data access + download for client
  - Sometimes free, sometimes requires payment
  - Data may be public or may be protected
  - Much quicker than web-scraping if you can get what you need using an API
- Web-Scraping
  - Writing code to “scrape” data from webpages in an automated way.
  - Sometimes called “crawling”
  - May be prohibited by the data provider (need to check terms of use)
  - Could provide you data that is not previously analyzed (due to barriers of accessing it)

DATA FORMS AND MODELS

- Possible Forms of Data:
  - Data could be organized in rows and columns in a tabular form.
  - Could be organized in a hierarchical tree structure, like eXtensible Markup Language (XML) or JavaScript Object Notation (JSON).
  - Database queries always return data organized in a relational table of rows and columns.

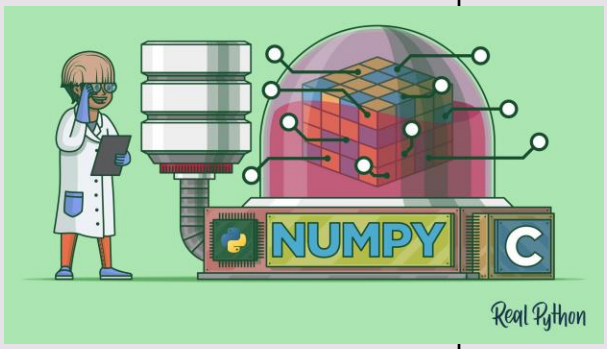
- Our data models define the following for each form of data.
  - The structure used for data employing the particular model.
  - The operations that may be executed against data structed as given by the model.
  - The constraints that govern valid values for data in the model, as well as properties to be held invariant in and among the data represented in the model.

- 3 Main Data Models:

1. Tabular data model
2. Relational data model
3. Hierarchical data model


PYTHON PACKAGES
FOR DATA
MANAGEMENT

NUMPY



- - Acronym for "Numerical python".
- - a Python package for high performance computing and data analysis created in 2005.
- - NumPy works with n-th dimensional arrays, aim to provide an array object that is up to 50x faster than a traditional list.

TECHNICAL SIDE

```
import numpy as np
import os
```

- NumPy package is not part of the standard Python, so you need to install and import it.

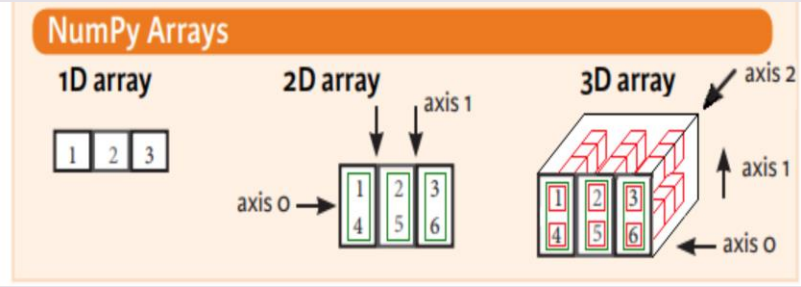
- To use numpy, the common coding form would be "np.function"

```
age = np.array([5,10,25,23,7])

print(age[1])

print(age[1:4])

print(age[2:])

print(age[::-1])

age.shape
```

TECHNICAL SIDE

Things can be done with numpy function:

+adding, removing, and sorting elements
+tell the shape and size of an array
+reshape an array
+index and slicing
+maximum, minimum, sum, mean, etc.
+math and trigonometric function




```
#create a numpy director of numbers between 2,10 with steps of 2
#this won't get 10
np.arange(2,10,2)
#this will get 10
np.arange(2,11,2)

#for fun
np.arange(0,100,10)
```

```
array([ 0, 10, 20, 30, 40, 50, 60, 70, 80, 90])
```

```
print(age)
#return age greater than 20
print(age[age>=20])
#age greater than 20 but less than 10:
print(age[(age>10)|(age<20)])
# or: |
#and: &
#not: ~
#not equals: !=
#equal: ==
```

```
[ 5 10 25 23  7]
[25 23]
[ 5 10 25 23  7]
```

```
#height

weight = [200,160,150,210,200,100]

mydata= np.array([height,weight])
print(mydata)

print(mydata.ndim)
print(mydata.size)
print(mydata.shape)

print(mydata[1,1])
print(mydata[1,1:])
```

```
[[ 72  70  69  38  90  72]
 [200 160 150 210 200 100]]
2
12
(2, 6)
160
[160 150 210 200 100]
```

```
x = np.arange(1,10)
print(x)
A= x.reshape(3,3)
print(A)

#resize() does the same, but overwrite

print(x)
new = np.resize(x,(3,3))
print(new)
```

```
[1 2 3 4 5 6 7 8 9]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[1 2 3 4 5 6 7 8 9]
[[1 2 3]
 [4 5 6]
 [7 8 9]]
```

PANDAS

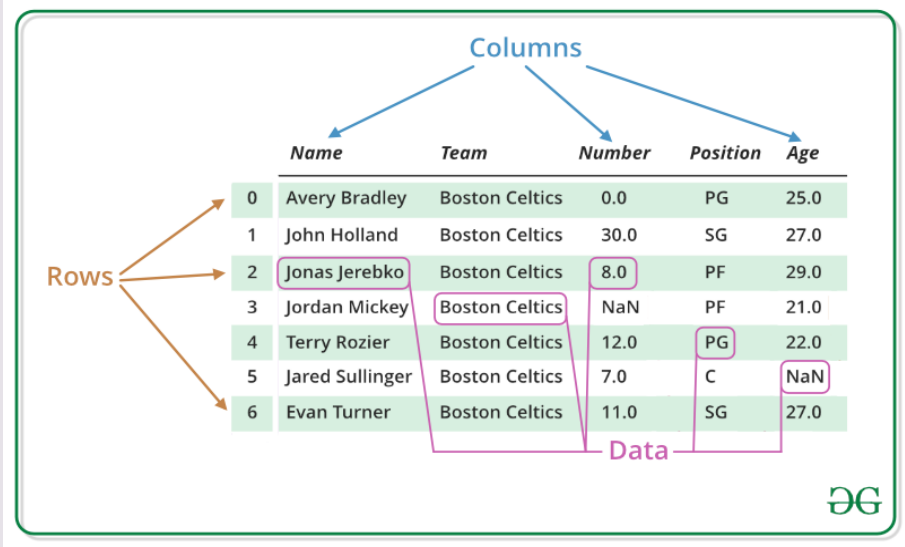
- An open-source Python library for data analysis, created in 2008
- The name "Pandas" has a reference to both "Panel Data", and "Python Data Analysis"

Why Use Pandas?

Pandas allows us to analyze big data and make conclusions based on statistical theories.

Pandas can clean messy data sets, and make them readable and relevant.

Relevant data is very important in data science.



FUNCTIONS

Dataframe functions:
- create dataframe
- Read CSV, JSON file
- Analyzing data

Data related technique:
-- Data manipulation
-- Cleaning data
-- Cleaning Empty cells
-- Cleaning Wrong format/data
-- Remove duplicate

```
import pandas as pd
pd.DataFrame({'A': [1, 2, 3]})
```

```
df = pd.read_csv('https://raw.githubusercontent.com/ernbilen/Data200_
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12540 entries, 0 to 12539
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   code     12540 non-null  object
 1   year     12540 non-null  int64
 2   country  12540 non-null  object
 3   pop      12434 non-null  float64
 4   gdp      9343 non-null   float64
 5   life     11296 non-null  float64
 6   cell     9178 non-null   float64
dtypes: float64(4), int64(1), object(2)
memory usage: 685.9+ KB
```

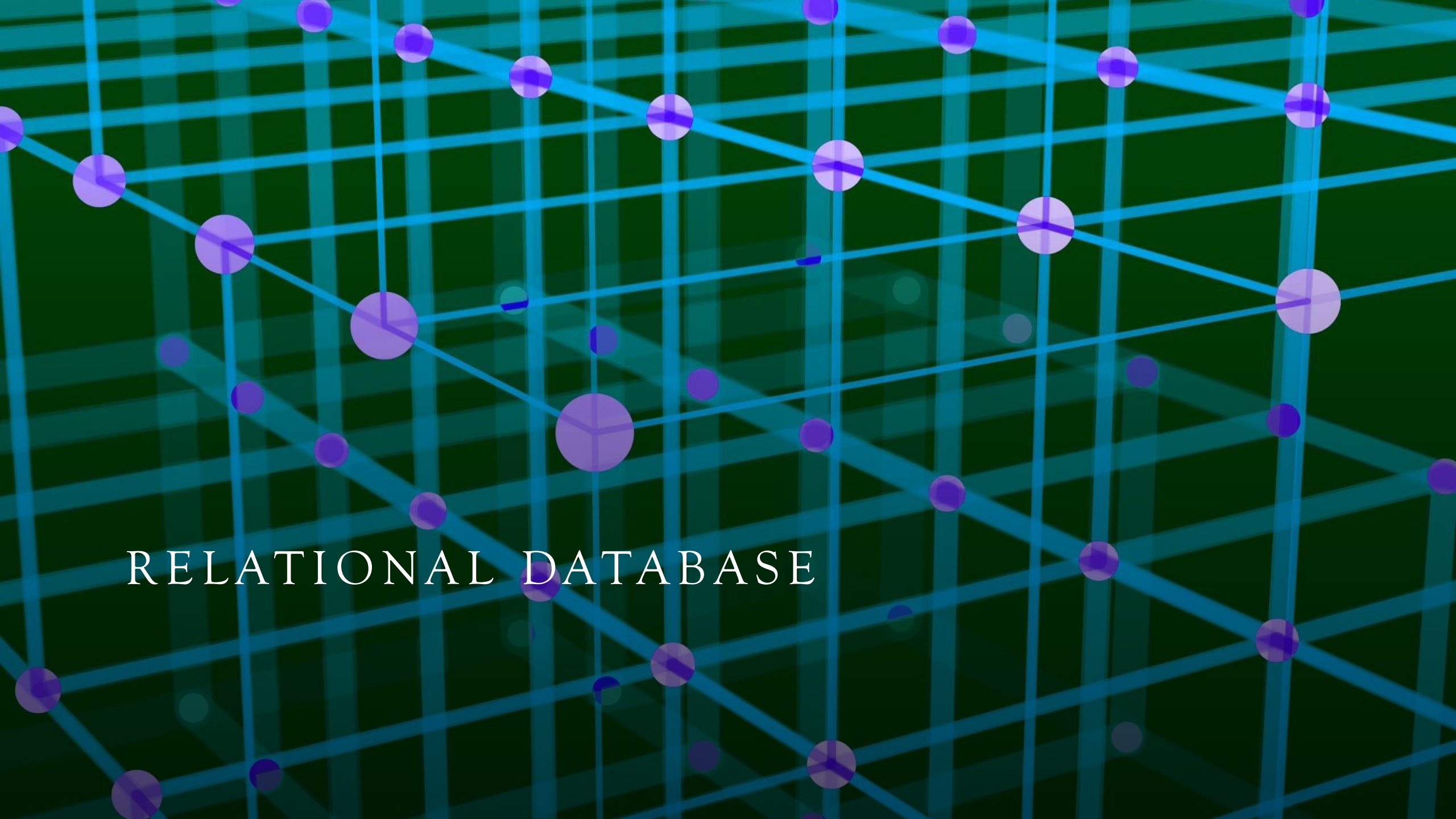
```
df.head()
```

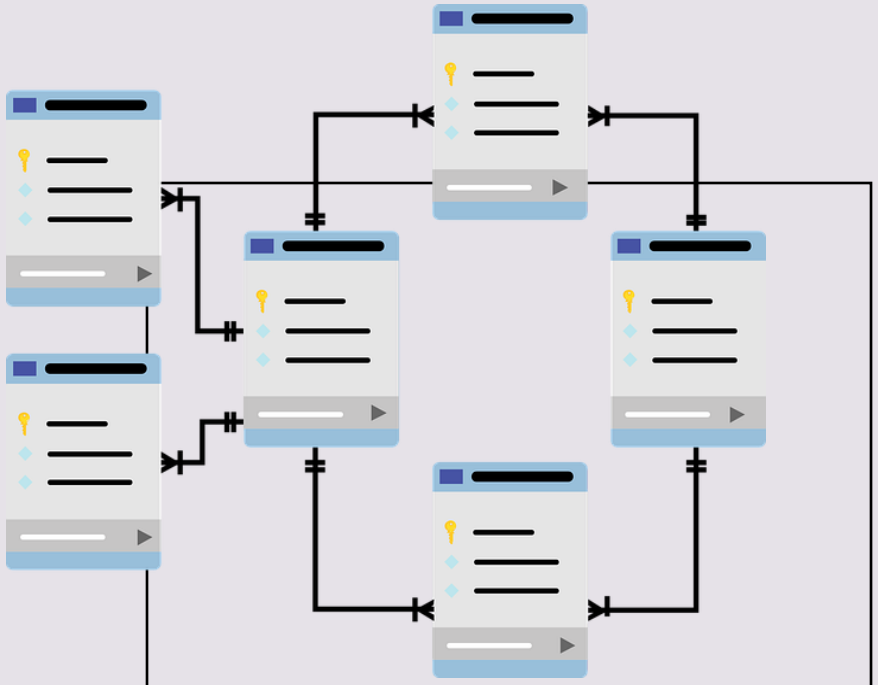
	code	year	country	pop	gdp	life	cell
0	ABW	1960	Aruba	0.05	NaN	65.66	0.0
1	ABW	1961	Aruba	0.06	NaN	66.07	NaN
2	ABW	1962	Aruba	0.06	NaN	66.44	NaN
3	ABW	1963	Aruba	0.06	NaN	66.79	NaN
4	ABW	1964	Aruba	0.06	NaN	67.11	NaN

```
df.index = df.code
```

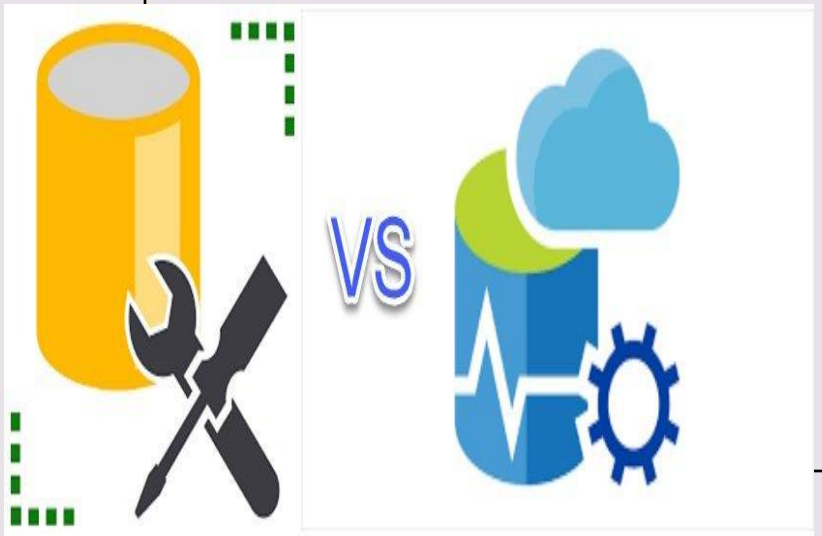
```
>>> df = pd.DataFrame({'Animal': ['Falcon', 'Falcon',
...                               'Parrot', 'Parrot'],
...                    'Max Speed': [380., 370., 24., 26.]})
>>> df
   Animal  Max Speed
0  Falcon      380.0
1  Falcon      370.0
2  Parrot       24.0
3  Parrot       26.0
>>> df.groupby(['Animal']).mean()
        Max Speed
Animal
Falcon      375.0
Parrot       25.0
```

```
>>> s1 = pd.Series(['a', 'b'])
>>> s2 = pd.Series(['c', 'd'])
>>> pd.concat([s1, s2])
0    a
1    b
0    c
1    d
dtype: object
```


RELATIONAL DATABASE

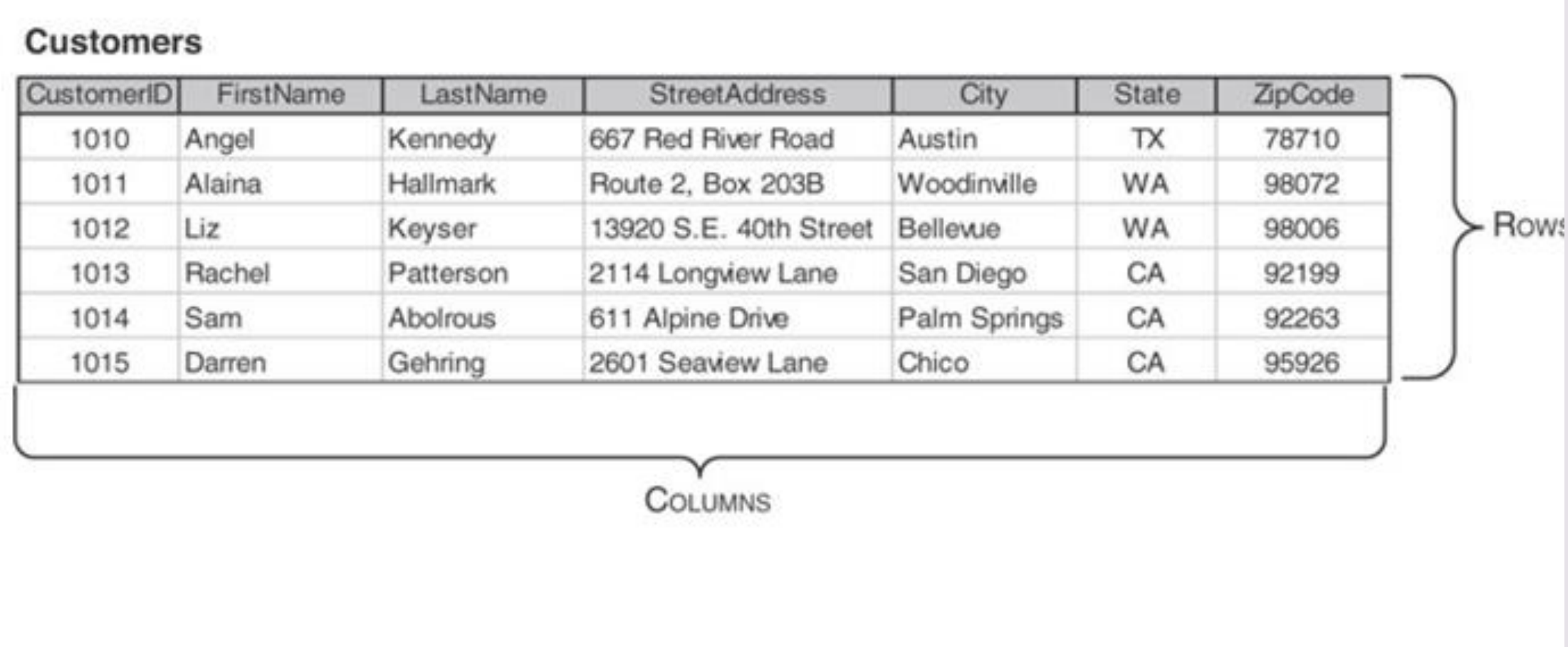


Relational Database is a type of database that stores and provides access to data points that are related to one another. (most common).



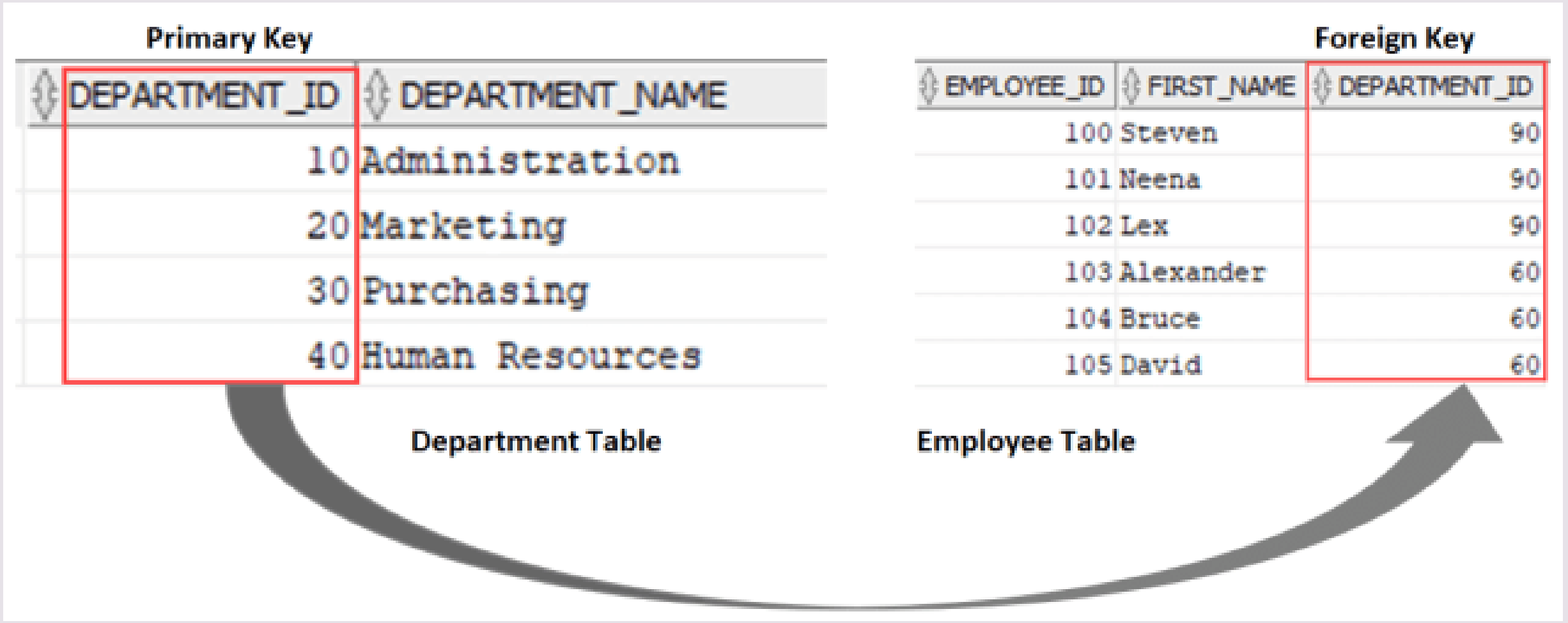
A DATABASE MANAGEMENT SYSTEM (DBMS)
IS **SOFTWARE** FOR MANAGING DATABASES AND PROVIDING ACCESS TO THEM. (CREATE, READ, ADD, UPDATE)

•ROWS & COLUMNS



Customers

CustomerID	FirstName	LastName	StreetAddress	City	State	ZipCode
1010	Angel	Kennedy	667 Red River Road	Austin	TX	78710
1011	Alaina	Hallmark	Route 2, Box 203B	Woodinville	WA	98072
1012	Liz	Keyser	13920 S.E. 40th Street	Bellevue	WA	98006
1013	Rachel	Patterson	2114 Longview Lane	San Diego	CA	92199
1014	Sam	Abolrous	611 Alpine Drive	Palm Springs	CA	92263
1015	Darren	Gehring	2601 Seaview Lane	Chico	CA	95926


Primary Key
DEPARTMENT_ID
DEPARTMENT_NAME
10 Administration
20 Marketing
30 Purchasing
40 Human Resources
Department Table
Foreign Key
EMPLOYEE_ID
FIRST_NAME
DEPARTMENT_ID
100 Steven 90
101 Neena 90
102 Lex 90
103 Alexander 60
104 Bruce 60
105 David 60
Employee Table

RELATIONAL DATABASE

- Functional Depencies: a relationship between two sets of attributes in a relation/table, where the value of one set of attributes determines the value of another attribute. (X --> Y)
- Ex: `courseid` ➞ `coursesubject`, `coursenum`, `coursetitle`, `coursehours`
- Its Importance:
- + Determines primary key for a table
- + Normalizing data table

  + Identify Relationships between different tabale: one-to-one, many-to-many, one-to-many

SQL Basics Cheat Sheet

LearnSQL.com

SQL

SQL, or *Structured Query Language*, is a language to talk to databases. It allows you to select specific data and to build complex reports. Today, SQL is a universal language of data. It is used in practically all technologies that process data.

SAMPLE DATA

COUNTRY

id	name	population	area
1	France	66600000	640680
2	Germany	80700000	357000
...	...	...	...

CITY

id	name	country_id	population	rating
1	Paris	1	2243000	5
2	Berlin	2	3460000	3
...	...	...	...	...

QUERYING SINGLE TABLE

Fetch all columns from the `country` table:

```
SELECT *
FROM country;
```

Fetch `id` and `name` columns from the `city` table:

```
SELECT id, name
FROM city;
```

Fetch city names sorted by the `rating` column in the default ASCending order:

```
SELECT name
FROM city
ORDER BY rating [ASC];
```

Fetch city names sorted by the `rating` column in the DESCending order:

```
SELECT name
FROM city
ORDER BY rating DESC;
```

ALIASES

COLUMNS

```
SELECT name AS city_name
FROM city;
```

TABLES

```
SELECT co.name, ci.name
FROM city AS ci
JOIN country AS co
  ON ci.country_id = co.id;
```

FILTERING THE OUTPUT

COMPARISON OPERATORS

Fetch names of cities that have a rating above 3:

```
SELECT name
FROM city
WHERE rating > 3;
```

Fetch names of cities that are neither Berlin nor Madrid:

```
SELECT name
FROM city
WHERE name != 'Berlin'
  AND name != 'Madrid';
```

TEXT OPERATORS

Fetch names of cities that start with a 'P' or end with an 's':

```
SELECT name
FROM city
WHERE name LIKE 'P%'
   OR name LIKE '%s';
```

Fetch names of cities that start with any letter followed by 'ublin' (like Dublin in Ireland or Lublin in Poland):

```
SELECT name
FROM city
WHERE name LIKE '_ublin';
```

OTHER OPERATORS

Fetch names of cities that have a population between 500K and 5M:

```
SELECT name
FROM city
WHERE population BETWEEN 500000 AND 5000000;
```

Fetch names of cities that don't miss a rating value:

```
SELECT name
FROM city
WHERE rating IS NOT NULL;
```

Fetch names of cities that are in countries with IDs 1, 4, 7, or 8:

```
SELECT name
FROM city
WHERE country_id IN (1, 4, 7, 8);
```

QUERYING MULTIPLE TABLES

INNER JOIN

`JOIN` (or explicitly **`INNER JOIN`**) returns rows that have matching values in both tables.

```
SELECT city.name, country.name
FROM city
[INNER] JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	3	Iceland

LEFT JOIN

`LEFT JOIN` returns all rows from the left table with corresponding rows from the right table. If there's no matching row, **`NULL`**s are returned as values from the second table.

```
SELECT city.name, country.name
FROM city
LEFT JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	NULL	NULL

RIGHT JOIN

`RIGHT JOIN` returns all rows from the right table with corresponding rows from the left table. If there's no matching row, **`NULL`**s are returned as values from the left table.

```
SELECT city.name, country.name
FROM city
RIGHT JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
NULL	NULL	NULL	3	Iceland

FULL JOIN

`FULL JOIN` (or explicitly **`FULL OUTER JOIN`**) returns all rows from both tables – if there's no matching row in the second table, **`NULL`**s are returned.

```
SELECT city.name, country.name
FROM city
FULL [OUTER] JOIN country
  ON city.country_id = country.id;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
2	Berlin	2	2	Germany
3	Warsaw	4	NULL	NULL
NULL	NULL	NULL	3	Iceland

CROSS JOIN

`CROSS JOIN` returns all possible combinations of rows from both tables. There are two syntaxes available.

```
SELECT city.name, country.name
FROM city
CROSS JOIN country;

SELECT city.name, country.name
FROM city, country;
```

CITY			COUNTRY	
id	name	country_id	id	name
1	Paris	1	1	France
1	Paris	1	2	Germany
2	Berlin	2	1	France
2	Berlin	2	2	Germany

NATURAL JOIN

`NATURAL JOIN` will join tables by all columns with the same name.

```
SELECT city.name, country.name
FROM city
NATURAL JOIN country;
```

CITY			COUNTRY	
country_id	id	name	name	id
6	6	San Marino	San Marino	6
7	7	Vatican City	Vatican City	7
5	9	Greece	Greece	9
10	11	Monaco	Monaco	10

`NATURAL JOIN` used these columns to match rows:
`city.id, city.name, country.id, country.name`
`NATURAL JOIN` is very rarely used in practice.

SQL

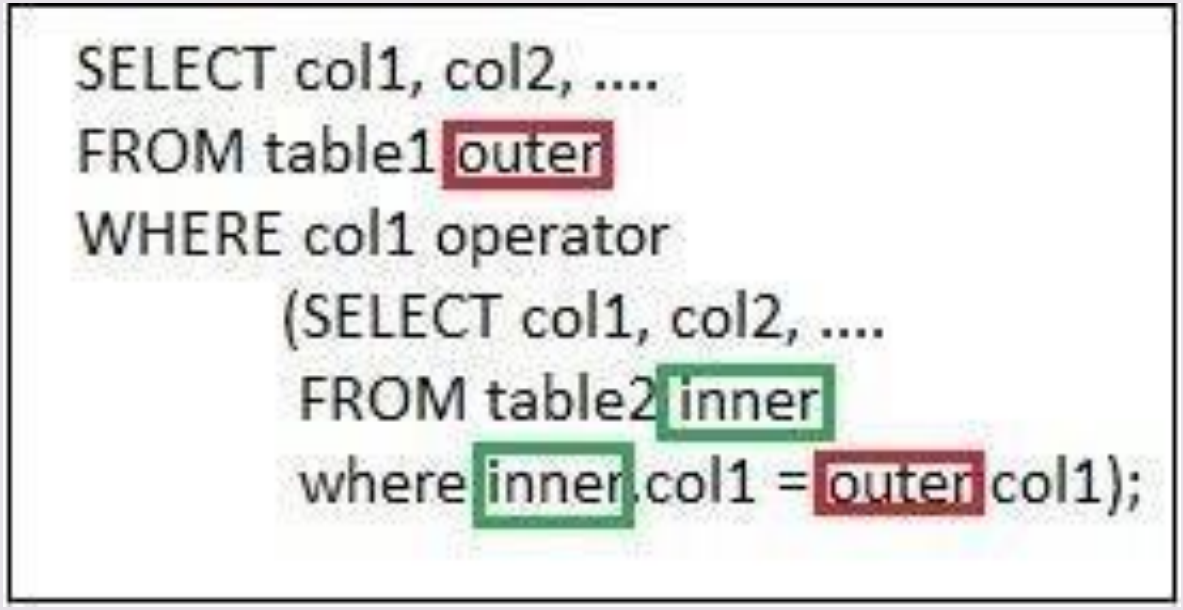
IS A STANDARD LANGUAGE FOR MANAGING AND MANIPULATING RELATIONAL DATABASES. HERE'S A SUMMARY OF KEY ASPECTS OF SQL

SQL ALLOWS USERS TO:

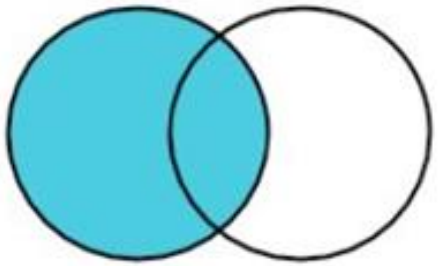
- QUERY DATA FROM DATABASES
- INSERT, UPDATE, AND DELETE RECORDS
- CREATE AND MODIFY DATABASE STRUCTURES
- SET PERMISSIONS ON DATABASE OBJECTS

SUBQUERIES

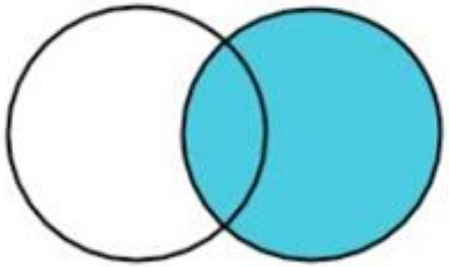
```
SCALAR SUBQUERY
select t1.id,
       t1.data_str,
       (select data_num from test_inner_2 where id = t1.id) data_num
  from test_inner_1 t1
 where t1.id = 1000;

TABLE JOIN
select t1.id, t1.data_str, t2.data_num
  from test_join_1 t1, test_join_2 t2
 where t1.id = t2.id
   and t1.id = 1000;
```



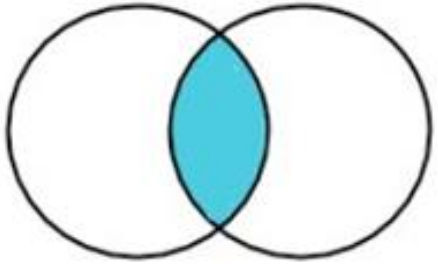
SQL Joins



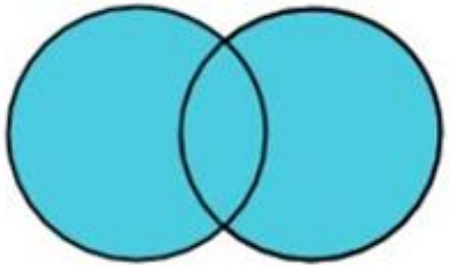
Left Join



Right Join



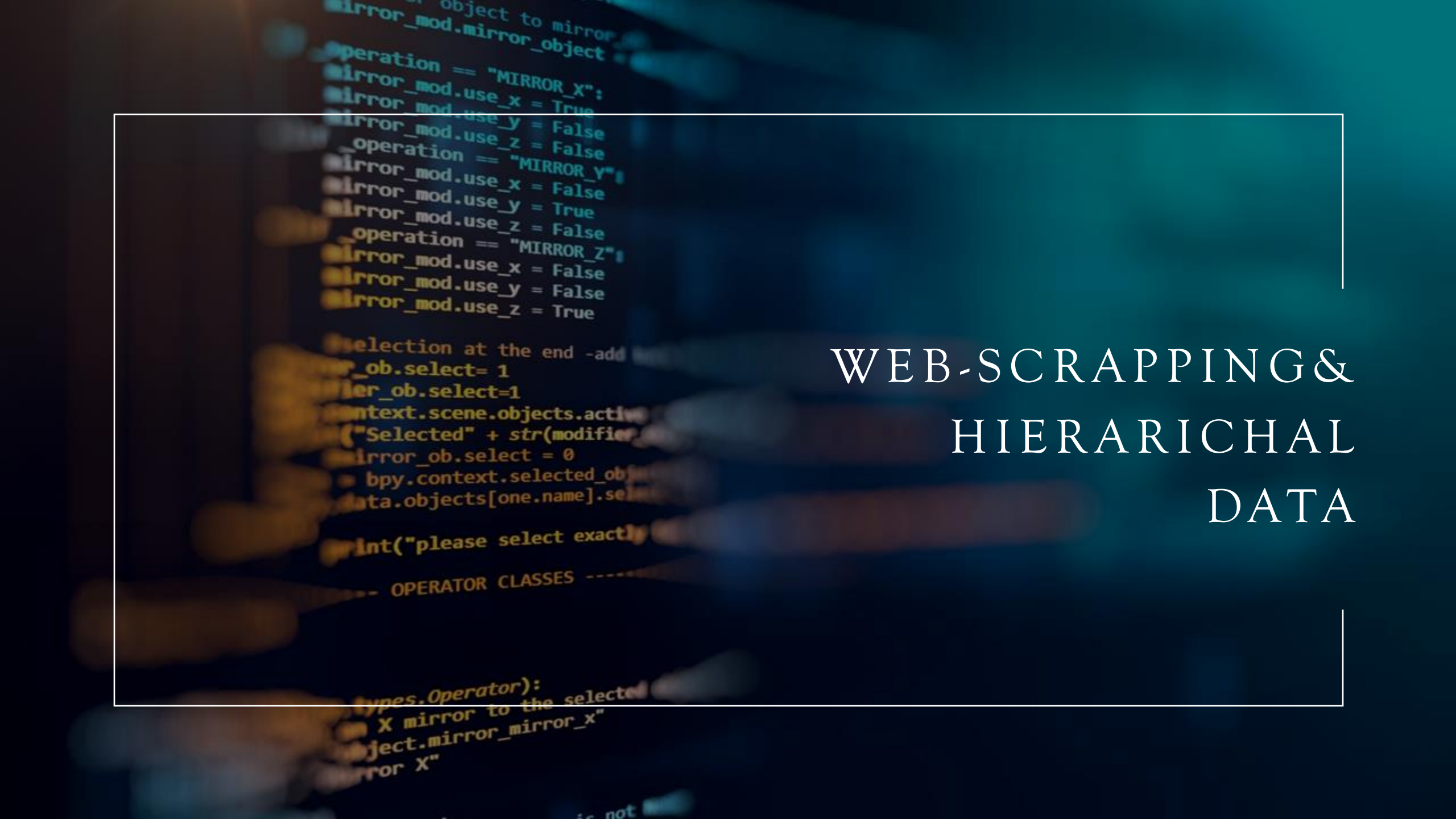
Inner Join



Full Outer Join

SQL ALCHEMY

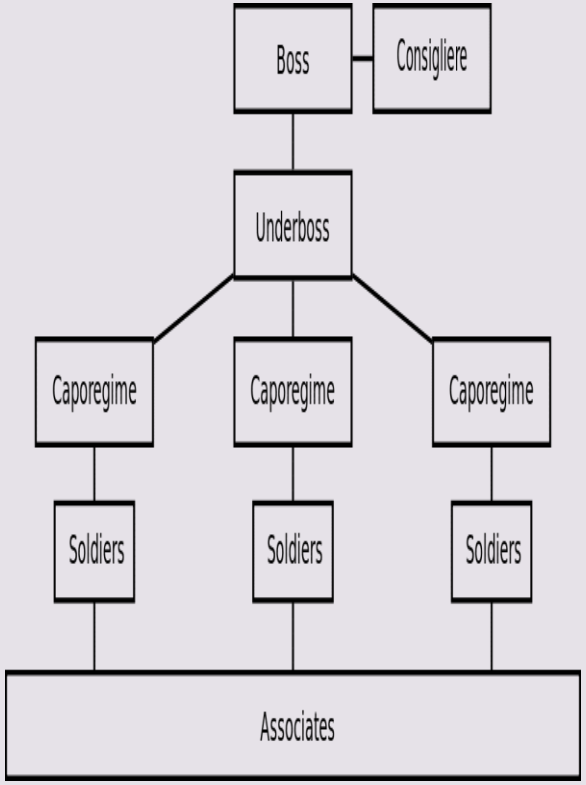
- SQLAlchemy is a powerful and flexible Python SQL toolkit and Object Relational Mapper (ORM) that provides developers with the full capabilities of SQL



WEB-SCRAPPING& HIERARICHAL DATA

HIERARICHAL DATA

- Hierarchical Database Models
  - Structure: Tree-like organization with parent-child relationships
  - Simplicity: Easy to understand and navigate
- Key Applications
  - Operating Systems: File and folder structures
  - Websites: Drop-down menus and navigation
  - Data Formats: JSON, XML, HTML
- Advantages
  - Fast data traversal and retrieval
  - Efficient for one-to-many relationships
  - Easy addition/deletion of data



- Web Data and APIs
  - Natural fit for hierarchical data from web scraping and APIs
  - XPath for XML/HTML parsing
  - Selenium for dynamic web scraping
- Limitations
  - Less flexible for complex relationships
  - Struggles with many-to-many relationships

REGEX

- Regular Expressions (RegEx)
- Definition: Powerful pattern-matching sequences for text searching and manipulation
- Key Components
- Literal Characters: Exact text to match
- Metacharacters: Special symbols (e.g., ., *, +, ?, [ ], ^, $)
- Quantifiers: Specify repetition (e.g., *, +, ?, {n}, {n,m})
- Character Classes: Define sets of characters (e.g., [a-z], [0-9])
- Anchors: Match positions (^ for start, $ for end)
- Common Applications
- Input validation (e.g., email, password)
- Text search and replace
- Data extraction and formatting
- Web scraping
- Log file analysis

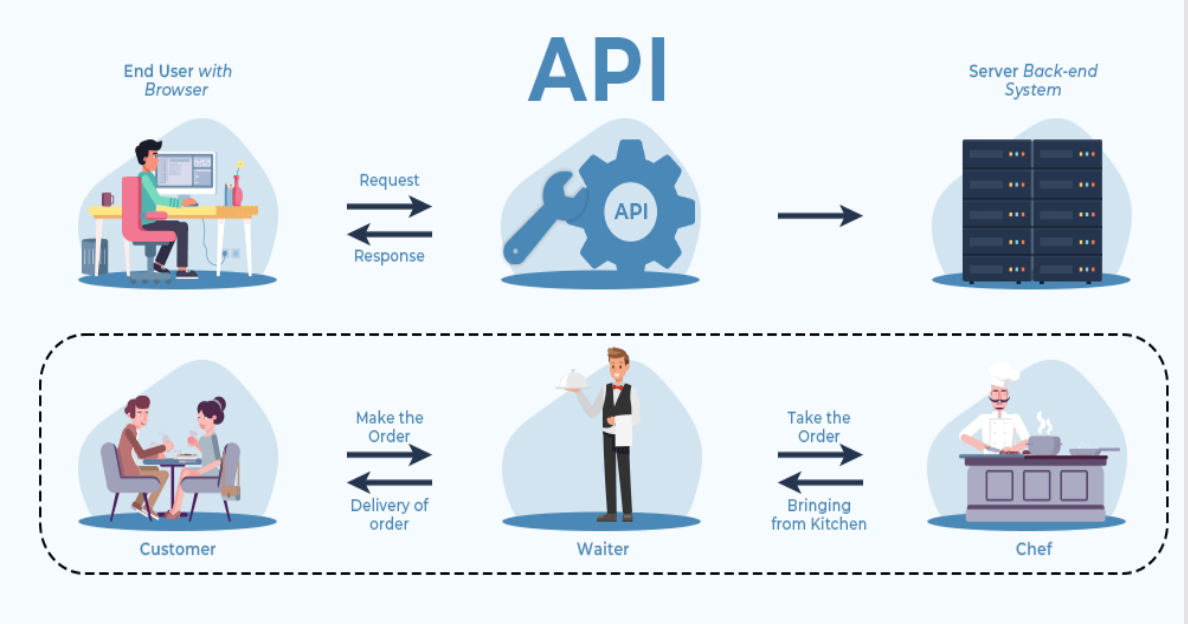
API

- What is an API?

- Definition: An API is a set of protocols that allows different software applications to communicate with each other, enabling data exchange and functionality sharing .
- Analogy: Think of an API as a waiter in a restaurant who takes your order (request), communicates with the kitchen (service), and brings back your food (response) .

- How APIs Work

- Request: A client application sends a request to the API endpoint.
- Processing: The API processes the request and interacts with the server or database.
- Response: The API sends back the requested data or confirmation of action taken



SELENIUM WEB SCRAPPING

Automate interactions with web pages, like clicking buttons and filling forms

Handle dynamic content loaded by JavaScript

Scrape data from complex, JavaScript-heavy websites

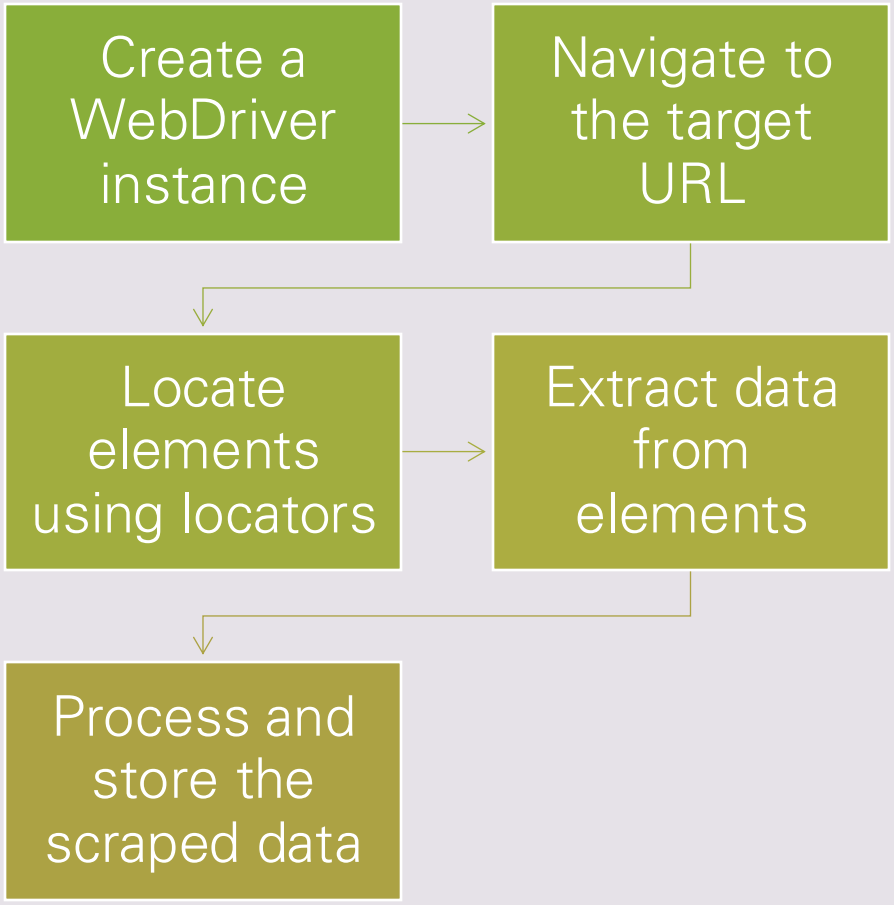
SETTING UP SELENIUM

INSTALL THE SELENIUM PACKAGE: PIP INSTALL SELENIUM

INSTALL A WEBDRIVER FOR YOUR BROWSER (E.G. CHROMEDRIVER FOR CHROME)

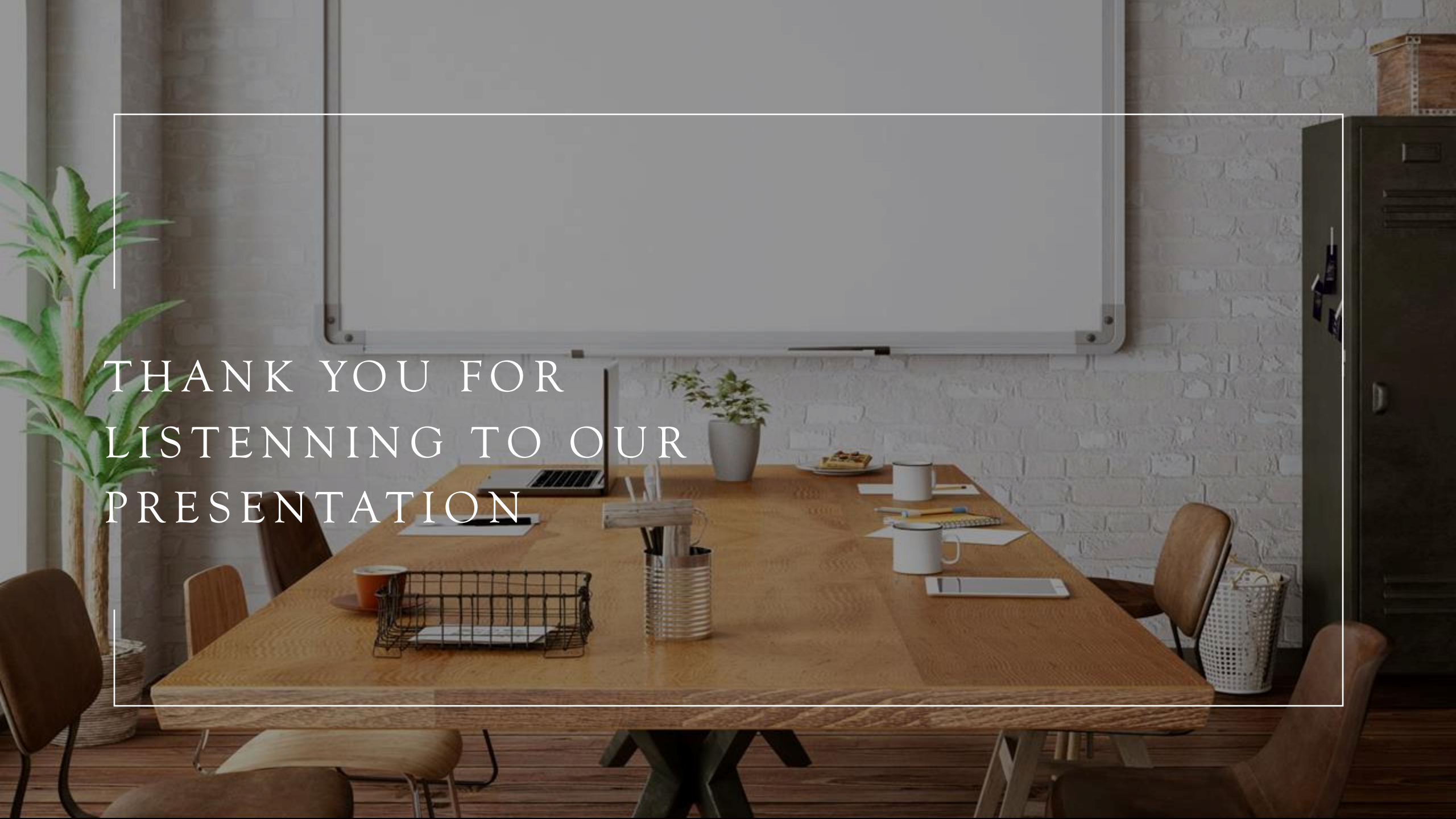
IMPORT SELENIUM MODULES IN YOUR PYTHON SCRIPT

BASIC SCRAPING STEPS



python

```python
from selenium import webdriver
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()
driver.get("https://example.com")

# Find element by ID and extract text
title = driver.find_element(By.ID, "title").text

# Find elements by CSS selector
items = driver.find_elements(By.CSS_SELECTOR, ".item")
for item in items:
    print(item.text)

driver.quit()
```


THANK YOU FOR
LISTENNING TO OUR
PRESENTATION