Pat Rademacher and Luke Ding
Charles Winstead
June 2019
CS 586

DBMS Project Schema - Submission #3

Database Access:
Database is formally created under Pat's account.

Account: s19wdb28
Password: q4gB6xah*z
Database Schema: s19wdb28

Overview:
We'd like to provide a comprehensive source of all of Tesla's information, such that any prospective customer can find information about the product, and any manager of Tesla can quickly look up product information and issues that may plague any company.

Actors- The main type of actors using this information will be either customers or employees of Tesla (including Elon himself). Customers will be using this database for queries to access stores, service centers, decide which models they want to buy. Employees will use the database to track inventory models, sales locations, supercharger usage, etc.

Technology Stack- There are two technologies used to develop the database. The first is a web scraper. We used beautiful soup in Python to parse HTML of Tesla.com. This was able to provide us all the locations and telephone numbers of every Tesla Service Center, Store, and Supercharger in the world. A link to the code for the beautiful soup is in the appendix in Python.

We also used Python to connect with the database and create tables, define the columns, and upload entries into the table using the psycopg2, pandas, and sqlalchemy libraries. The entire script is in the appendix.

Table Schemas:

Car (Model, Subtype, Range, zto60)

Wait Times(Model, Subtype, Country, Wait Time)
Model, Subtype -> Derives uniqueness from car Model, Subtype

Purchases(Model, Subtype, CustomerID, Date, Vin)
Model, Subtype -> Car (Model, Subtype)
CustomerID -> Customer(CustomerID)

Employees (Employee ID, Name, Role, Telephone, Email)

Stores (StoreID,Name, Address, Extended Address, Locale, Country, Telephone, Review)

StoreEmployees (Store Name, EmployeeID)
Name -> Stores(Name)
EmployeeID -> Employees(EmployeeID)

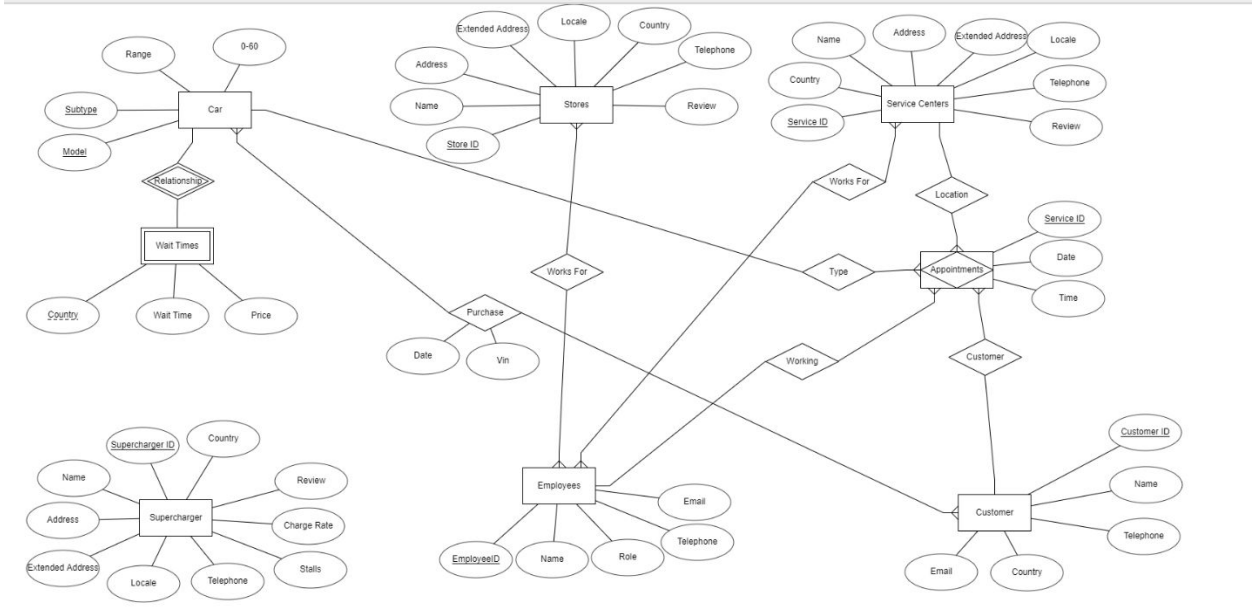
Service Centers (Service, Name ,Address, Extended Address, Locale, Country, Telephone, Review)

ServiceEmployees (Service Name, EmployeeID)
Name -> Service Centers(Name)
EmployeeID -> Employees(EmployeeID)

Supercharger (SuperchargerID,Name, Address 1, Address 2, Locale, Country, Stalls, Charge Rate, Review, Telephone,)

Appointments(ID, Date, Time, Model, Subtype, Location, EmployeeID, CustomerID)
Model, Subtype -> Car (Model, Subtype)
CustomerID -> Customer(CustomerID)
EmployeeID -> Employees(EmployeeID)
Location -> Service Centers(ServiceID)

Customers (Customer ID, Name, Telephone, Country, Email)


Range
0-60
Subtype
Car
Model
Relationship
Wait Times
Country
Wait Time
Price
Extended Address
Locale
Country
Telephone
Address
Name
Stores
Review
Store ID
Name
Address
Extended Address
Locale
Country
Service Centers
Telephone
Service ID
Review
Works For
Location
Service ID
Date
Type
Appointments
Time
Works For
Purchase
Date
Vin
Working
Customer
Supercharger ID
Country
Name
Review
Address
Supercharger
Charge Rate
Extended Address
Locale
Telephone
Stalls
Employees
Email
Telephone
EmployeeID
Name
Role
Customer ID
Name
Customer
Telephone
Email
Country

Example Queries:

1) What is the phone number of the Portland Macadam store?

- Actor: A resident of Portland interested in purchasing or checking out Tesla cars.
- English: What is the phone number for the Portland Tesla store?
- SQL Query:
    - SELECT phone
    - FROM service_centers
    - WHERE name = 'Portland-SW Macadam Avenue'
- Results:

phone
(503) 382-1685

1 row(s)

Total runtime: 2.113 ms

2) What are all the stores with five star google reviews in the United States?

- Actor: Somebody trying to find the best Tesla stores in the US. Perhaps they are doing a road trip and want to know the best locations to stop to get something fixed, or perhaps they are willing to travel to the best store within a certain radius for a purchase.
- English: “Which stores in the U.S. are the best and have a five star Google review rating?”
- SQL Query:
    - SELECT name
    - FROM stores
    - WHERE google_review_rating = 5 and country = 'US'
- Results:

name
Scottsdale-Fashion Square
Scottsdale-Kierland Commons
Buena Park
Burbank
Mission Viejo-Shops at Mission Viejo
Newport Beach-Fashion Island
Palo Alto
Santa Barbara-Hitchcock Way
Walnut Creek-Broadway Plaza
Christiana Mall
Dania Beach
Naples-Waterside Shops
Palm Beach Gardens â€" The Gardens Mall
Atlanta-Lenox Square
Chicago-Gold Coast
Chicago-Highland Park
Oak Brook-Oakbrook Center
Indianapolis
Indianapolis-The Fashion Mall at Keystone
Baltimore-Owings Mills
Dedham
Country Club Plaza
Cherry Hill
Paramus-Route 17
Springfield
Raleigh
Cincinnati
Portland-SW Macadam Avenue
King of Prussia-King of Prussia Mall
Dallas Galleria
Southlake-Southlake Town Center
Tyler-South Southwest Loop
Salt Lake City-S. State Street
McLean-Tysons Corner
Bellevue-20th Street
University Village

36 row(s)

Total runtime: 2.195 ms

3) Group all the stores in the world by stars?

- Actors: Somebody interested in seeing all Tesla stores by ranking.
- English: “In what format can I see all Tesla stores in the world by their Google review rating?”
- SQL Query:
    - SELECT name, google_review_rating
    - FROM stores
    - GROUP BY google_review_rating, name
    - ORDER BY google_review_rating
- Results:

name	google_review_rating
â °åŒ— Tesla å…§æ'–ç‡Ÿé ‹ç¸½éƒ¨	3
Aargau	3
Amsterdam-PC Hooftstraat	3
Antwerp-Aartselaar	3
Arnhem-Duiven	3
Auckland - Ponsonby	3

4) Which stores are also service centers?

- Actor: Somebody who is looking to possibly purchase Tesla and can also ask mechanical questions during the process.
- English: “Where can I find a place to buy a Tesla and also ask mechanical questions about it?”
- SQL Query:
    - SELECT stores.name
    - FROM stores, service_centers
    - WHERE stores.name = service_centers.name
- Results:

name
Buena Park
Burbank
Burlingame
Costa Mesa Service
Dublin-Amador Plaza
Los Angeles-Centinela

91 row(s)

Total runtime: 3.622 ms

5) Return the list of all stores in Texas.

- Actor: Potential customer in Texas trying to see which location is closest to him.
- English: “Give me the list of all Tesla stores located in Texas.”
- SQL Query:
    - SELECT stores.local
    - FROM stores
    - WHERE local LIKE '%TX%'
- Results:

local
Austin TX 78759
Austin TX 78758
Dallas TX 75240
Dallas TX 75225
Fort Worth TX 76109
Houston TX 77056
The Woodlands TX 77380
Plano TX 75024
San Antonio TX 78257
Southlake TX 76092
Tyler TX 75701

11 row(s)

Total runtime: 1.473 ms

6) How many superchargers are there by kwh?

- Actor: Somebody who needs to charge their Tesla but doesn't know any information about superchargers.
- English: "What are the different kinds of superchargers in regard to kilowatt per hour?"
- SQL Query:
    - SELECT charge_rate
    - FROM s19wdb28.superchargers
    - GROUP BY charge_rate
    -
- Results:

charge_rate
250 KWH
72 KWH
120 KWH

3 row(s)

7) Give me the list of 72 KWH superchargers so I can start upgrading them first.

- Actor: Elon Musk or somebody responsible for Tesla to upgrade the lowest grade superchargers.
- English: "Give me our slowest performing superchargers so we can start to make them better."
- SQL Query:
    - SELECT name, charge_rate
    - FROM s19wdb28.superchargers
    - WHERE charge_rate = '72 KWH'
    -
- Results:

name	charge_rate
Greenville Supercharger	72 KWH
Oxford AL Supercharger	72 KWH
Cordes Lakes AZ Supercharger	72 KWH
Flagstaff AZ Supercharger	72 KWH
Tucson AZ Supercharger	72 KWH

481 row(s)

Total runtime: 8.953 ms

8) Which country are most of our customers in?

- Actor: Somebody who works for Tesla and needs to figure out where majority of their efforts in advertising should go; somebody who is generally interested in Tesla and wants to know this information.
- English: "Which country has the most amount of customers for Tesla?"
- Query:
    - SELECT customers.country, count(customers.country) AS num_cust_per_country
    - FROM customers
    - GROUP BY customers.country
    - ORDER BY customers.country DESC
- Results:

country	num_cust_per_country
US	55
United Kingdom	4
United Arab Emirates	4
The Netherlands	4
Taiwan	4
Switzerland	4
Sweden	4
Spain	4
South Korea	4
Slovenia	4
Slovakia	4
Portugal	4
Poland	4
Norway	4
New Zealand	4
Mexico	5
Macao S.A.R. China	4
Luxembourg	4
Liechtenstein	4
Jordan	4
Japan	4
Italy	4
Ireland	4
Hungary	4
Hong Kong S.A.R. China	4
Germany	4
France	4
Finland	4
Denmark	4
Czech Republic	4
Croatia	5
China Mainland	4
Canada	5
Belgium	5
Austria	5
Australia	4

36 row(s)

- Total runtime: 3.104 ms

9) Give me the emails of all store managers and service center manager

- Actor: Somebody in the HR Department of Tesla or high in management who needs to send an email out to all store managers and service center managers of Tesla stores
- English: “I need all the email addresses of managers of stores and service centers.”
- SQL Query:
    - SELECT employees.email
    - FROM employees
    - WHERE role = 'Store Manager' OR role = 'Service Manager'
    -

- Results:

email
SandiBolt@tesla.com
MoraPhillips@tesla.com
TashiaOneil@tesla.com
ChaeMurillo@tesla.com
JenDaring@tesla.com
CathernMalagon@tesla.com
QuintinBaylis@tesla.com
MyrtlePrintup@tesla.com
AdellaMoultrie@tesla.com
DarnellGillins@tesla.com
KoriStancil@tesla.com
EloisHerzig@tesla.com
EmogeneSturrock@tesla.com
DagnyThacker@tesla.com
GeneAddison@tesla.com

15 row(s)

Total runtime: 1.910 ms

10)
Actor: Elon Musk is asking what the Model 3 delivery times to China are
English: How long do I have to wait to get a model 3 standard range in mainland china?
Query:

```
SELECT *
FROM "wait_time"
where country = 'China Mainland' AND model = 'Model 3'
```

Result:

model	subtype	country	wait_time	price
Model 3	Standard	China Mainland	7	76000
Model 3	Long Range	China Mainland	7	95000

11)
Actor: Curious customer who wants to know what the price range of a Tesla is
Question: What's the most and least expensive car model Tesla sells?
Result:

```
SELECT model, subtype, price, country
FROM "wait_time"
where price in (select min(price) from "wait_time" union select max(price) from "wait_time")
```

Query:

model	subtype	price	country
Model X	Long Range	169100	China Mainland
Model X	Long Range	169100	United Arab Emirates
Model 3	Standard	40000	US

12)
Actor: Elon wants to honor the first worldwide customer.
Question: Who was our first customer ever?
Query:

```
SELECT name, phone, country, email, date
from customers join (
SELECT customer_id, date
FROM "purchase_history"
where date in (select min(date) from "purchase_history")) as min on customers.customer_id =
min.customer_id
```

Result:

name	phone	country	email	date
Wen Straughter	(399) 920-8850	US	WenStraughter@gmail.com	2011-12-18

13)
Actor: Elon Musk got a call from OHSA for overworking people
Question: Show me that no employee works at both a store and a service center
Query:

```
select *
from store_employees natural join service_employees
```

Results:

No rows found.

Total runtime: 2.487 ms

SQL executed.

Edit SQL

14)
Actor: Google Machine Learning Team
Question: Judging from the appointments data, what hours are the service centers at Tesla open from?
Query:

```
select min(time) as earliest, max(time) as latest
from appointments
```

Results:

earliest	latest
8	19

15)
Actor: Elon Musk wanting to do right by customers
Question: Which customer (by ID) has had to take their car in for service the most number of times?
Query:

```
select customer_id, count(*) as frequency
from appointments
group by customer_id
having count(*) = (
(
select max(new.count)
from
(select customer_id, count(*)
from appointments
group by customer_id) as new)
)
```

Result:

customer_id	frequency
194	5

16)
Actor: Customer
Question: Which model has the slowest zero to sixty acceleration time?
Query/Result:

```
SELECT * FROM s19wdb28.cars where zto60 = (select max(zto60) from s19wdb28.cars)
```

Submit

Actions	model	subtype	range	zto60
Edit Delete	Model 3	Standard	240 Miles	5

17)
Actor: Compliance at Tesla
Question:
A customer complained that Harry is not a good sales representative, and that is all they know. Find me all the possible suspects.
Query/Result:

```
SELECT * FROM s19wdb28.employees where name like 'Harry%';
```

Submit

Actions	employee_id	name	role	phone	email
Edit Delete	16	Harry Genna	Sales	(945) 275-6213	HarryGenna@tesla.com
Edit Delete	67	Harry Swarthout	Sales	(789) 820-7751	HarrySwarthout@tesla.com

18) STORED PROCEDURE EXAMPLE
Actor: Elon
Question: Return the number of customers we have:
Stored Procedure:

```
CREATE OR REPLACE FUNCTION totalRecords ()

RETURNS integer AS $total$

declare

        total integer;

BEGIN

  SELECT count(*) into total FROM customers;

  RETURN total;

END;

$total$ LANGUAGE plpgsql;
```

select totalRecords()

Result:

totalrecords
200

Analysis & Reflection:

Overall, it was a lot harder of a project than we initially thought. There were two areas.

1) Beautiful Soup, the tool we used to collect all the location data for Tesla, was not easy to use. Web scraping is hard to learn, as every website can build their content differently. Tesla, for example, blocks active web scraping, so you have to simulate yourself as a Mozilla client. Even then, not all data is formatted perfectly.

2) Getting data into the system is hard as well. Embedded SQL takes time, and takes a lot of processing to clean.

It was a fun project though. We learned a lot and worked hard, and the data for the most part fits real world situations that people would ask. All the code for Beautiful Soup (Appendix A) and the Embedded SQL (Appendix B) is attached as well.

Appendix A (Beautiful Soup)

```
from urllib.request import Request, urlopen
from bs4 import BeautifulSoup as soup
import csv

def find_address (country,append,filename):
    url = 'https://www.tesla.com' + append
    req = Request(url, headers={'User-Agent': 'Mozilla/5.0'})

    webpage = urlopen(req).read()

    page_soup = soup(webpage, "html.parser")

    addresses = page_soup.find_all('address', class_='vcard')

    for address in addresses:
        try:
            name = address.find('a').contents[0]
        except:
            name = ''

        try:
            street = address.find('span', class_= 'street-address').text.strip()
        except:
            street = ''

        try:
            extended = address.find('span', class_='extended-address').text.strip()
        except:
            extended = ''

        try:
            locality = address.find('span', class_='locality').text.strip()
        except:
            locality = ''

        try:
            phone = address.find('span', class_='value').text.strip()
        except:
            phone = ''

        # street = address.find('span', class_= 'street-address').text.strip()
        # extended = address.find('span', class_='extended-address').text.strip()
        # locality = address.find('span', class_='locality').text.strip()
        # # city = locality.split(',')[0]
        # # state = locality.split(',')[1].strip().split(" ")[0]
        # # zipcode = locality.split(',')[1].strip().split(" ")[1]
        # phone = address.find('span', class_='value').text.strip()


        # print(name)
        # print(street)
        # print(extended)
        # print(locality)
        # print(phone)
        # print()


        with open(filename, 'a', encoding='utf8') as f:
            f.write('%s|' % country)
```

```
            f.write('%s|' % name)
            f.write('%s|' % street )
            f.write('%s|' % extended)
            # f.write('%s,' % city)
            # f.write('%s,' % state)
            # f.write('%s,' % zipcode)
            f.write('%s|' % locality)
            f.write('%s|' % phone)
            f.write('\n')



# find_address('US','/findus/list/stores/Japan','filetest1.csv')




url = 'https://www.tesla.com/findus/list'
req = Request(url , headers={'User-Agent': 'Mozilla/5.0'})

webpage = urlopen(req).read()

page_soup = soup(webpage, "html.parser")

divs = page_soup.find_all('div', class_ = 'row')
count = 0
for div in divs:
    div = div.find_all('ul')
    for elements in div:
        for tag in elements.find_all("li"):
            address = str(tag.a).split('\"')[1]
            country = tag.span.text.strip()
            find_address(country, address, 'filetest2.csv')
            print(count)
            count = count + 1
```

Appendix B (Embedded SQL)

```
#Final Database Project - Pat Rademacher and Luke Ding
#Charles Winstead
#June 4, 2019
#CS 586


import psycopg2 as p
import pandas as pd
import io
from sqlalchemy import create_engine
from sqlalchemy import Integer
from sqlalchemy import String
from sqlalchemy import Date
from sqlalchemy import ForeignKeyConstraint
from sqlalchemy import ForeignKey
from sqlalchemy import TEXT, MetaData, Table, Column, Float
import csv


#initialize engine to connect to class Database


engine =
create_engine('postgresql://s19wdb28:q4gB6xah*z@dbclass.cs.pdx.edu:5432/s19wdb28
')
conn = engine.connect()
```

```
# read in all excel and CSV sheets through Pandas


Stores = pd.read_excel('Datasheet.xlsx', Sheetname=0)

Cars = pd.read_csv('Datasheet.csv')

service_centers = pd.read_csv('service_centers.csv', encoding = 'latin-1')

superchargers = pd.read_csv('superchargers.csv', encoding = 'latin-1')

customers = pd.read_csv('customers.csv', encoding = 'latin-1')

employees = pd.read_csv('employees.csv', encoding = 'latin-1')

purchhistory = pd.read_csv('purchasehistory.csv', encoding = 'latin-1')

storeemp = pd.read_csv('storeemp.csv', encoding = 'latin-1')

serveemp = pd.read_csv('servemp.csv', encoding = 'latin-1')

appointments = pd.read_csv('REALAPPOINTMENTS.csv', encoding = 'latin-1')

waittimes = pd.read_csv('waittimes.csv', encoding = 'latin-1')



#initialize proper list sizes for transferring CSV and Excel sheets to list data
types



superchargers_array = [''] * 10

cust = [''] * 5

emp = [''] * 5

ph = [''] * 5

ste = [''] * 2

sve = [''] * 2

app = [''] * 8

                                                                    wt = [''] * 5

                                                                newemp = [''] * 5
```

```
                                                    for i in range(10):
                            superchargers_array[i] = list(superchargers.iloc[:, i])
for i in range(5):
    cust[i] = list(customers.iloc[:, i])
    #cust[i].strip("Ê")
    emp[i] = list(employees.iloc[:199, i])
    #emp[i].strip("Ê")
    ph[i] = list(purchhistory.iloc[:, i])
    wt[i] = list(waittimes.iloc[:, i])
for i in range(2):
    ste[i] = list(storeemp.iloc[:, i])
                                                sve[i] = list(serveemp.iloc[:, i])
for i in range(8):
    app[i] = list(appointments.iloc[:, i])


#had a strange instance when reading in CSV files from home through PSU's server
- had to eliminate following character


                                    cust[1] = [s.replace('Ê', ' ') for s in cust[1]]
cust[4] = [s.replace('Ê', '') for s in cust[4]]
                                      emp[1] = [s.replace('Ê', ' ') for s in emp[1]]
                                       emp[4] = [s.replace('Ê', '') for s in emp[4]]


Store_StoreID = []
Store_Country = []
Store_Name = []
Store_Address = []
Store_Extended = []
Store_Local = []
```

```
Store_Phone = []
                                                    Store_GoogleReviewRating = []


Store_StoreID.append(Stores.iloc[:, 0])
Store_Country.append(Stores.iloc[:, 1])
Store_Name.append(Stores.iloc[:, 2])
Store_Address.append(Stores.iloc[:, 3])
Store_Extended.append(Stores.iloc[:, 4])
Store_Local.append(Stores.iloc[:, 5])
Store_Phone.append(Stores.iloc[:, 6])
                                Store_GoogleReviewRating.append(Stores.iloc[:, 7])


Cars_Model = []
Cars_Subtype = []
Cars_Range = []
                                                              Cars_Zto60 = []


Cars_Model.append(Cars.iloc[:, 0])
Cars_Subtype.append(Cars.iloc[:, 1])
Cars_Range.append(Cars.iloc[:, 2])
                                                  Cars_Zto60.append(Cars.iloc[:, 3])


service_id = []
service_country = []
service_name = []
service_address = []
service_extended = []
service_local = []
```

```
service_phone = []
                                                    service_google = []


service_id.append(service_centers.iloc[:, 0])
service_country.append(service_centers.iloc[:, 1])
service_name.append(service_centers.iloc[:, 2])
service_address.append(service_centers.iloc[:, 3])
service_extended.append(service_centers.iloc[:, 4])
service_local.append(service_centers.iloc[:, 5])
service_phone.append(service_centers.iloc[:, 6])
                                service_google.append(service_centers.iloc[:, 7])


#convert lists to Pandas' DataFrame data type


df_stores = pd.DataFrame(data = {'store_id' : Store_StoreID[0], 'country' :
Store_Country[0], 'name' : Store_Name[0], 'address' : Store_Address[0],
'extended' : Store_Extended[0], 'local' : Store_Local[0], 'phone' :
Store_Phone[0], 'google_review_rating' : Store_GoogleReviewRating[0]})


df_cars = pd.DataFrame(data = {'model' : Cars_Model[0], 'subtype' :
Cars_Subtype[0], 'range' : Cars_Range[0], 'zto60' : Cars_Zto60[0]})


df_service = pd.DataFrame(data = {'service_id' : service_id[0], 'country' :
service_country[0], 'name': service_name[0], 'address': service_address[0],
'extended': service_extended[0], 'local': service_local[0], 'phone':
service_phone[0], 'google_review_rating': service_google[0]})


df_superchargers = pd.DataFrame(data = {'supercharger_id':
superchargers_array[0], 'country': superchargers_array[1], 'name':
superchargers_array[2], 'address': superchargers_array[3], 'extended':
```

```
superchargers_array[4], 'local': superchargers_array[5], 'phone':
superchargers_array[6], 'google_review_rating': superchargers_array[7],
'stalls': superchargers_array[8], 'charge_rate': superchargers_array[9]})


df_customers = pd.DataFrame(data = {'customer_id': cust[0], 'name': cust[1],
'phone': cust[2], 'country': cust[3], 'email': cust[4]})


df_employees = pd.DataFrame(data ={'employee_id': emp[0], 'name': emp[1],
'role': emp[2], 'phone': emp[3], 'email': emp[4]})


df_ph = pd.DataFrame(data = {'customer_id': ph[0], 'model': ph[1], 'subtype':
ph[2], 'date': ph[3], 'vin': ph[4]})


df_ste = pd.DataFrame(data = {'employee_id': ste[0], 'store_id': ste[1]})


df_sve = pd.DataFrame(data = {'service_id': sve[0], 'employee_id': sve[1]})


df_wt = pd.DataFrame(data = {'model': wt[0], 'subtype': wt[1], 'country': wt[2],
'wait_time': wt[3], 'price': wt[4]})


df_app = pd.DataFrame(data = {'appointment_id': app[0], 'model': app[1],
'subtype': app[2], 'service_id': app[3], 'employee_id': app[4], 'customer_id':
app[5], 'date': app[6], 'time': app[7]})



    #use the Pandas 'to_sql' function which converts DataFrames into proper data
                                                type to be process for postgres SQL
```

```
df_stores.to_sql(name='stores', con=engine, if_exists = 'replace', index=False,
dtype = {"store_id": Integer(), "country": String(), "name" : String(),
"address" : String, "extended" : String(), "local" : String(), "phone" :
String(), "google_review_rating": Integer()})
```

```
df_cars.to_sql(name='cars', con=engine, if_exists = 'replace', index=False,
dtype = {"model" : String(), "subtype" : String(), "range" : String(), "zto60" :
Float()})
```

```
df_service.to_sql(name='service_centers', con=engine, if_exists = 'replace',
index=False, dtype = {"service_id": Integer(), "country": String(), "name":
String(), "address": String(), "extended": String(), "local": String(), "phone":
String(), "google_review_rating": Integer()})
```

```
df_superchargers.to_sql(name= 'superchargers', con=engine, if_exists =
'replace', index=False, dtype = {"supercharger_id": Integer(), "country":
String(), "name" : String(), "address" : String, "extended" : String(), "local"
: String(), "phone" : String(), "google_review_rating": Integer(),
"stalls":Integer(), "charge_rate": String()})
```

```
df_customers.to_sql(name= 'customers', con=engine, if_exists = 'replace',
index=False, dtype = {"customer_id": Integer(), "name": String(), "phone" :
String(), "country": String(), "email": String()})
```

```
df_employees.to_sql(name= 'employees', con=engine, if_exists = 'replace',
index=False, dtype = {"employee_id": Integer(), "name": String(), "role" :
String(), "phone": String(), "email": String()})
```

```
df_ph.to_sql(name= 'purchase_history', con=engine, if_exists = 'replace',
index=False, dtype = {'customer_id': Integer(), 'model': String(), 'subtype':
String(), 'date': Date(), 'vin': String()})
```

```
df_ste.to_sql(name= 'store_employees', con=engine, if_exists = 'replace',
index=False, dtype = {'employee_id': Integer(), 'store_id': Integer()})
```

```
df_sve.to_sql(name= 'service_employees', con=engine, if_exists = 'replace',
index=False, dtype = {'service_id': Integer(), 'employee_id': Integer()})
```

```
df_app.to_sql(name= 'appointments', con=engine, if_exists = 'replace',
index=False, dtype = {'appointment_id': Integer(), 'model': String(), 'subtype':
String(), 'service_id': Integer(), 'employee_id': Integer(), 'customer_id':
Integer(), 'date': String(), 'time': Integer()})
```

```
df_wt.to_sql(name = 'wait_time', con=engine, if_exists = 'replace', index=False,
dtype = {'model': String(), 'subtype': String(), 'country': String(),
'wait_time': Integer(), 'price': Integer()})
```

```
conn.execute('ALTER TABLE stores ADD PRIMARY KEY (store_id);')
conn.execute('ALTER TABLE cars ADD PRIMARY KEY (model, subtype);')
conn.execute('ALTER TABLE customers ADD PRIMARY KEY (customer_id);')
conn.execute('ALTER TABLE employees ADD PRIMARY KEY (employee_id);')
conn.execute('ALTER TABLE purchase_history ADD PRIMARY KEY (vin), ADD CONSTRAINT
ph_cust FOREIGN KEY (customer_id) REFERENCES customers(customer_id), ADD
CONSTRAINT ph_car FOREIGN KEY(model, subtype) REFERENCES cars(model, subtype);')
conn.execute('ALTER TABLE service_centers ADD PRIMARY KEY (service_id);')
conn.execute('ALTER TABLE wait_time ADD PRIMARY KEY (model, subtype, country),
ADD CONSTRAINT wait_car FOREIGN KEY (model, subtype) REFERENCES cars(model,
subtype);')
conn.execute('ALTER TABLE service_employees ADD PRIMARY KEY (service_id,
employee_id), ADD CONSTRAINT serv_emp FOREIGN KEY(service_id) REFERENCES
service_centers(service_id), ADD CONSTRAINT emp_serv FOREIGN KEY (employee_id)
REFERENCES employees(employee_id);')
```

```
conn.execute('ALTER TABLE store_employees ADD PRIMARY KEY (store_id,
employee_id), ADD CONSTRAINT store_employee_id FOREIGN KEY (store_id) REFERENCES
stores(store_id), ADD CONSTRAINT employee_store_id FOREIGN KEY (employee_id)
REFERENCES employees(employee_id);')
conn.execute('ALTER TABLE superchargers ADD PRIMARY KEY (supercharger_id);')

conn.execute('ALTER TABLE appointments ADD PRIMARY KEY (appointment_id), ADD
CONSTRAINT app_empl FOREIGN KEY (employee_id) REFERENCES employees(employee_id),
ADD CONSTRAINT app_car FOREIGN KEY(model, subtype) REFERENCES cars(model,
subtype), ADD CONSTRAINT app_cust FOREIGN KEY(customer_id) REFERENCES
customers(customer_id), ADD CONSTRAINT app_serv FOREIGN KEY (service_id)
REFERENCES service_centers(service_id);')



conn.close()



#References



# https://docs.sqlalchemy.org/en/13/core/type_basics.html

#
https://robertdavidwest.com/2014/10/12/python-pandas-%E2%86%92-mysql-using-sqlal
chemy-a-k-a-sqlalchemy-for-pandas-users-who-dont-know-sql-the-brave-and-the-fool
hardy/
# https://docs.sqlalchemy.org/en/13/orm/join_conditions.html
```