Project Report

SQL

Xindi Sun xs2359, Rose dn2450, Leo Li sl4513, Smriti sv2577, Jingyi Zhang jz3021

Problem statement

Our client is the United States Citizenship and Immigration Services (USCIS). USCIS is responsible for the issuance and administration of H1B visa, which allows companies in the United States to temporarily employ foreign workers. Currently, USCIS operates according to immigration laws and foreign policies, yet it lacks the ability to contribute to the policy making process or to improve on its day-to-day administration of H1B. Therefore, USCIS aims to increase its business intelligence capacity in order to make evidence-driven decisions impacting H1B policies and application processes. There are two main groups of stakeholders: the analysts and executive-level managers (C-suite). The analysts' goal is to establish a database that allows them to generate insights from historical H1B application data. The C-suite will then review the key insights obtained from analyses and make informed decisions on future H1B related issues.

Proposal

Our group proposes that USCIS should establish a relational database from the H1B application dataset in PostgresQL with ETL process automated using Python and key insights reported using Metabase. The relational database will allow the analysts to perform analyses with improved speed and sophistication. The automated ETL process allows the analysts to quickly input historical data and also continue to enrich the database with future data. Finally, the dashboard will report key insights concerning trends and patterns in the H1B applications to C-suite, who can then make evidence-based decisions to improve H1B application process and make policy suggestions.

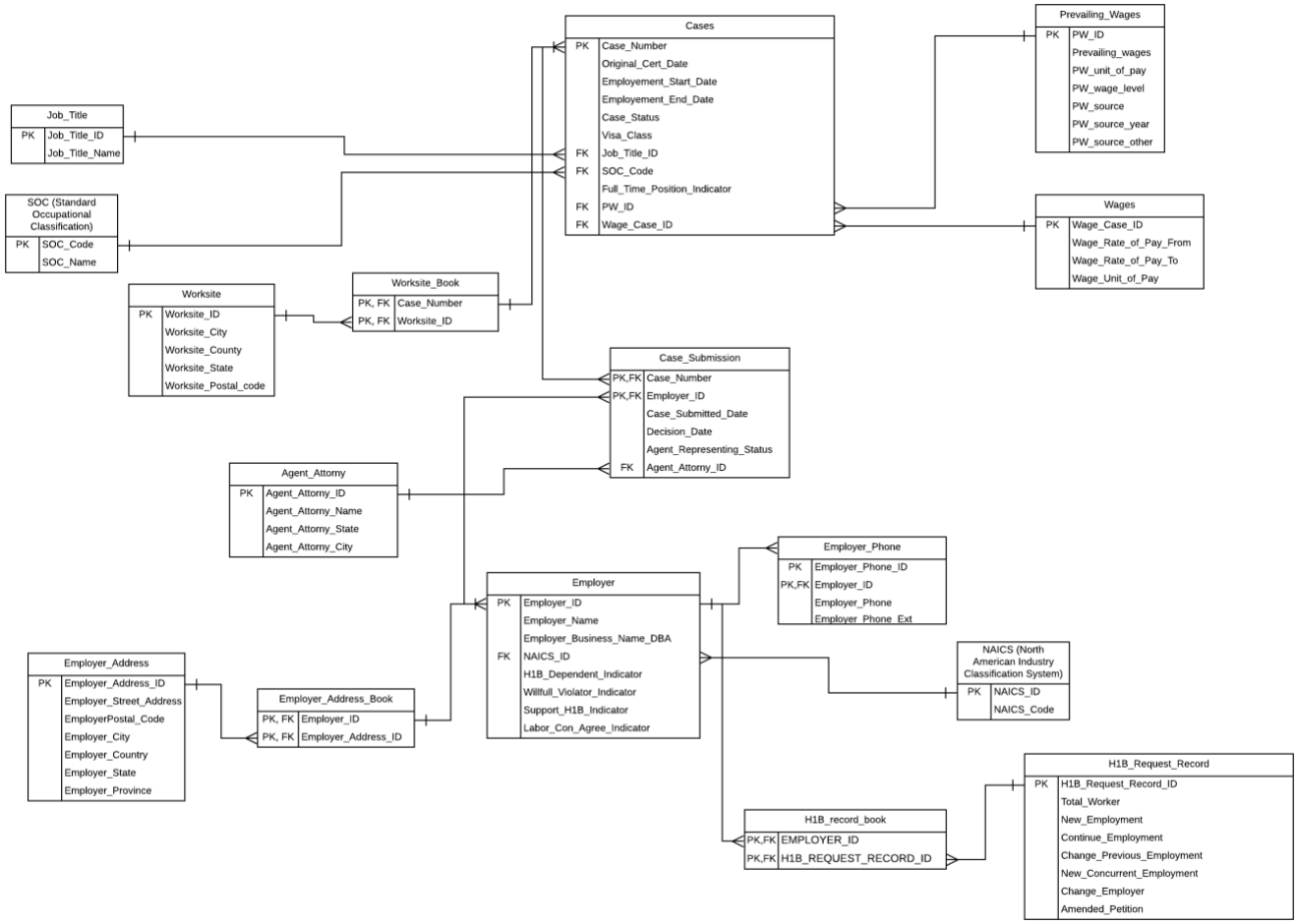
Team structure and Timeline

Agreed Weekly Meeting Time: Every Friday 3-5pm
Tasks and Responsibilities:
Checkpoint 2 - Nov 11
Task1 - Create preliminary tables from dataset: The whole team

Task2 - Construct team contract: The whole Team
Checkpoint 3 - Nov18
Task1 - database schema: The whole team
Task2 - ER diagram:Rose , Smriti
Task 3 - SQL code: Xindi Sun, Jingyi Zhang
Checkpoint 4 - Nov 25
Task1 - Transforming and entering the data to your database system: Leo Li, Jingyi Zhang, Xindi Sun
Task2 - Explanations and reasoning: Leo Li, Rose, Smriti
Checkpoint 5 - Dec 2
Task1 - plan for how customers will interact with the database system designed: The Whole team
Task2 - implement for analysts (direct querying): The Whole Team
Task3 - Report: Leo Li, Jingyi Zhang, Xindi Sun
Task4- Visualization: Smriti, Rose

Database Schema

Our dataset contains 54 columns. After initial cleaning, we eliminated two columns which contain only NAs. The ER-diagram below illustrates our relational database schema. The detailed reasonings for each table are listed after the diagram. The SQL code for generating tables is attached to the appendix.



- ***Job_Title***: In this table, we have job_title_id which is the primary key and an integer since its an ID. We also have Job_Title_name and that's varchar with a 200 character limit. The Job_title_name is NOT NULL but job_title_id is not as it's not null by default being the primary key.
- ***SOC***: This table has SOC_Code and SOC_Name. For SOC_Code we've used varchar with a character limit of 20. SOC codes have hyphens in them that cannot be categorized as integers and hence we have used varchar for them. SOC_Name is varchar(100) and not null. SOC_Code is not defined as not null as it's not null by default being the primary key.
- ***Prevailing_wages***: In this table, we have PW_ID as integer and that is the primary key, hence not defined as not null. Prevailing_wage is categorized as a float as the wages depicts money and can have multiple decimals.

PW_Unit_Of_Pay is if the pay is annual, weekly, monthly or hourly. This is depicted by varchar (10) and has a check to ensure only these units are entered. There is no not null constraint here as the dataset has null values for this column. PW_wage_level is varchar(10) and shows what is the wage level of the employee. So far we have level I, II, III, IV. PW_Source is varchar(8) and has data to see where the wage data is obtained from. PW_Source_Year is an integer and PW_Source_Other is varchar(300); this data shows if data is collected from the other source in the PW_Source. The data type is varchar and not text as the values are repeating and short in value.

- ***Wages***: In this table, we have wage_case_id which is an integer and the primary key, hence not null by default, wage_rate_of_pay_from which Is float and not null; this column depicts money which can have multiple decimals; same for the column wage_rate_of_pay_to, wage_unit_of_pay is varchar(10) and not null. There is also a check to ensure that the values entered are one of the following: 'Year','Month', 'Week', 'Hour','Bi-Weekly'.
- ***Worksite***: This table has worksite_ID as primary key and data type is an integer and it's not defined as not null as it's not null by default being the primary key. Worksite_city is varchar(50) and not null. Worksite_county is also varchar(50) but not marked as not null as the database has null values in this column. Worksite_state is char(2) and not null to ensure that only the state code is used for this entry. Worksite_postal_code is varchar(10), not null and not char(5) as expected as the dataset shows postal codes that vary from 3 digits to 10 digits.
- ***Employer_address:*** In this table employer_address_id is integer and primary key. And it's not defined as not null as it's not null by default being the primary key. The employer address is varchar(200) and not null. The data type is varchar and not text as the values are repeating and short in value. Employer city is varchar(50) and not null, employer_state is char(2) and not null to ensure that only the state code is used for this entry. Employer_postal_code is varchar(20), not null and not char(5) as expected as the dataset shows postal codes that vary from 3 digits to 20 digits. Employer_Country is varchar(50) and not null, this column so far holds only the United States of America as data. Employer_province is varchar(100) as the data in this column is very varied with integers and characters.
- ***Agent_Attorney:*** This table has agent_attorney_ID as primary key and data type is an integer, it's not defined as not null as it's not null by default being the primary key. Agent_attorney_name is varchar(100), agent_attorney_state

is char(2) to ensure that only the state code is used for this entry. Agent_attorney_City is varchar(50).

- ***NAICS:*** NAICS_ID is integer and integrity constraint is Unique to ensure that no duplication. It is also the primary key and by default not null. The ID is unique to each code that is used by each business that is classified into a six-digit NAICS code number based on the majority of activity at the business. NAICS_code is varchar(6) and not null to ensure that each code has its own ID.
- ***H1B_Request_Record:*** H1B_Request_Record_ID is primary key and data type is integer. Total_workers is an integer and not null. The dataset shows numbers from 1-40 randomly for this column. New_employment is an integer and not null and, in this column, shows numbers from 1-50 randomly. Continued_employment is an integer and not null, the data in the column range from 0-15. CHANGE_PREVIOUS_EMPLOYMENT is an integer and not null. The data in this column ranges from 0-25. NEW_CONCURRENT_EMPLOYMENT is an integer and not null. The data in this column ranges from 0-5. CHANGE_EMPLOYER is an integer and not null. The data in this column ranges from 0-15. AMENDED_PETITION is integer not null. The data in this column ranges from 0-15.
- ***Employer:*** Employer_ID is an integer and primary key. It's not defined as not null as it's not null by default being the primary key. Employer_name is varchar(200). There is no not null constraint here as the dataset has some null values. EMPLOYER_BUSINESS_DBA is varchar(100). NAICS_ID is an integer and is a foreign key in the table that references NAICS ID in the NAICS table. H1B_DEPENDENT is char(1) and depicts either Y or N in the dataset. WILLFUL_VIOLATOR is char(1) and depicts either Y or N in the dataset. SUPPORT_H1B is char(1) and depicts either Y or N in the dataset. LABOR_CON_AGREE is char(1) and depicts either Y or N in the dataset.
- ***Employer_Phone:*** Employer_Phone_ID is an integer and primary key. It's not defined as not null as it's not null by default being the primary key. EMPLOYER_PHONE is varchar(15) and not null. The 15 character limit causes the numbers in the dataset range from 9 digits to 15 digits. EMPLOYER_PHONE_EXT is varchar(8) and there is no not null here as some of the phones do not have an extension. EMPLOYER_ID is an integer and is a foreign key here. There is no employer_phone_ID referenced in the employer column as then there would be duplicate data populated and if there is a change in phone numbers we want to reflect in this column.

- ***Cases:*** CASE_NUMBER is varchar(20) and primary key. ORIGINAL_CERT_DATE is a timestamp. EMPLOYMENT_START_DATE is timestamp and not null to ensure that all employment start dates are captured. EMPLOYMENT_END_DATE is timestamp and not null to ensure that all employment end dates are captured. CASE_STATUS is varchar(20) and there is a check status implemented to ensure that only 'CERTIFIED','CERTIFIED-WITHDRAWN','DENIED', 'WITHDRAWN' can be entered for this column. VISA_CLASS is varchar(16) to accommodate entries of H1B-Singapore and H1B-Australia as well. JOB_TITLE_ID is an integer and foreign key and references Job_Title_ID in the Job_Title table. SOC_CODE is varchar(20) and foreign key and references SOC_CODE in the SOC table. FULL_TIME_POSITION is char(1) to show N or Y. PW_ID is an integer and foreign key that references PW_ID in the PREVAILING_WAGES table. WAGE_CASE_ID is an integer and foreign key that references WAGE_CASE_ID in the Wages table.
- ***WORKSITE_BOOK:*** This table has Case_Number, varchar(20), and Worksite_ID, integer, as composite key and both are foreign keys that are referenced from Cases table and worksite table respectively.
- ***EMPLOYER_ADDRESS_BOOK:*** In this table, we have Employer_ID and Employer_address_ID as composite keys. Both have data types as an integer. Employer ID is foreign key referenced from employer table and employer_address_ID is foreign key referenced from employer_address table
- ***CASE_SUBMISSION:*** Here we have case number as primary key and data type is varchar(20) to accommodate the hyphens that are part of the case number. The case number is also a foreign key that is referenced from the cases table. EMPLOYER_ID is an integer and foreign key referenced from the employer table. CASE_SUBMITTED_DATE is timestamp and it captures when the case was submitted for approval. DECISION_DATE is timestamp and it captures when the decision was made about the case. AGENT_REPRESENTING_STATUS is char(1) to show either N or Y. AGENT_ATTORNEY_ID is an integer and foreign key that references the agent_attorney table.
- ***H1B_RECORD_BOOK:*** EMPLOYER_ID and H1B_REQUEST_RECORD_ID are integers and composite keys for this table. Both are also foreign keys that are referenced from the employer table and the H1B_REQUEST_RECORD table respectively.

Extract, Transform, Load

First we create the prevailing_wages table with 6 columns getting from the original dataset: "prevailing_wage", "pw_unit_of_pay", "pw_wage_level","pw_source", "pw_source_year", "pw_source_other".Then we create id for each prevailing_wage and push data to the database.

Next, we create wages table with 3 columns: "wage_rate_of_pay_from", "wage_rate_of_pay_to", "wage_unit_of_pay" getting from the original dataset, generate id for each row under "pw_id" and push wages data to the database.

Because the soc_code are shown as date in excel so we find the soc_code from SOC website and create soc_code column that matches with SOC name. Then we create SOC tables with two columns "soc_code", "soc_name" and push data into the database. Then we create the job_title table with the job_title column from the original dataset, renamed into job_title_name and ID column and push data into the database.

In the next step, we gathered all cases relevant columns from original data frame into a dataframe called "cases". Then we merge this dataframe with "prevailing_wages" data frame on all columns to get the ID column of "prevailing_wages". We also merge "cases" data frame with "wages" dataframe on all columns in "wages" data frame to get ID column of "wages". We then merge "cases" data frame with "job_title" dataframe on all columns in "job_title" dataframe to get ID column of "job_title". Now the "cases" data frame has all foreign keys included so we get needed columns from this dataframe to create the "cases" table and push data to the database.

Moving on, we create worksite table by getting these 4 columns:"worksite_city", "worksite_county", "worksite_state", "worksite_postal_code" from the original dataset and generate ID as a new column and push data to database.

The "worksite_book" has "case_number" and "worksite_id" so we first create a dataframe that has "case_number" and all columns from the worksite tables.After that, in order to get the "worksite_id" we merge table "worksite_book" with table "worksite" on all same columns and create final worksite_book by getting only 2 columns needed, which serve as primary keys. Then push the worksite_book data to the database.

For the NAICS table, we select column 'naics_code' from the original database and generate id for each row. Then push NAICS table into the database.

In the next step we create the "h1b_request_record" table by getting needed columns 'total_workers', 'new_employment', 'continued_employment', 'change_previous_employment', 'new_concurrent_employment', 'change_employer', 'amended_petition' from the original dataframe and generate new id for each row, then push data to the database.

"Employer_address" table has all information relating to the employer's physical location so we select all relevant columns from original data frame 'employer_address',

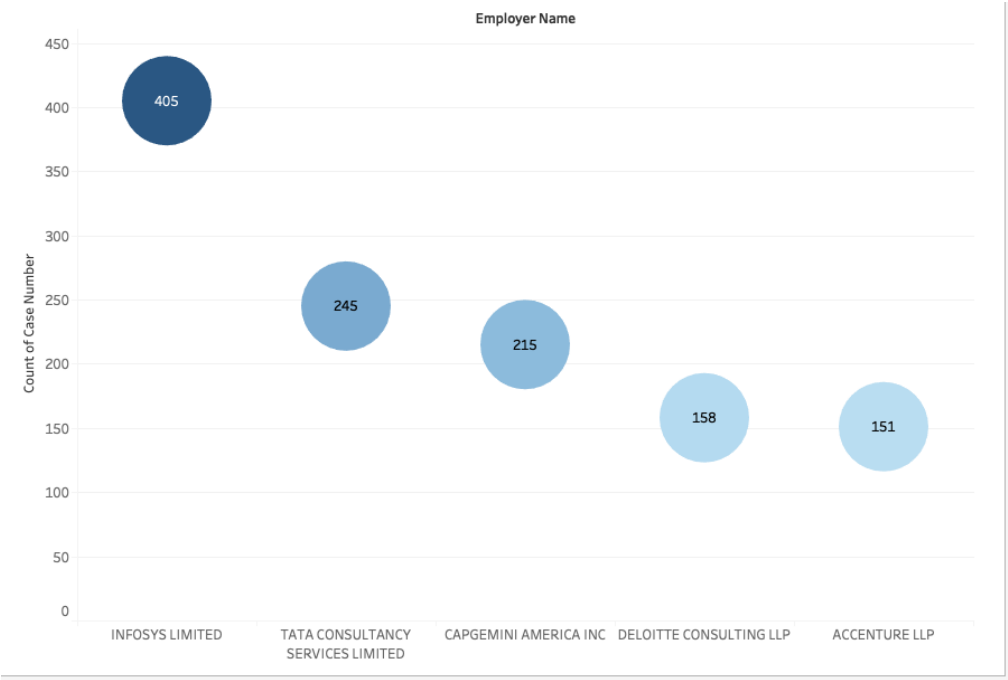
'employer_city', 'employer_state', 'employer_postal_code', 'employer_country', 'employer_province' and generate ID for each address. Then we push "Employer_address" to the database.
In the next step, we create an "employer" dataframe with these columns 'employer_name', 'employer_business_dba', 'h1b_dependent', 'willful_violator', 'support_h1b', 'labor_con_agree', 'employer_phone', 'employer_phone_ext', 'naics_code' and merge with NAICS dataframe on 'naics_code' to get the NAICS ID. Then generate id for each employer and populate the "employer" table into the database.
The "employer_phone" table has 4 columns 'employer_phone', 'employer_phone_ext', employer_phone_id and the employer_id as a foreign key.So first we create a dataframe that has 3 columns 'employer_name','employer_phone', 'employer_phone_ext' got from original dataframe, then merge with "employer" table on 'employer_name' to get 'employer_id' column. After that we generate unique ID for each phone number and gather needed columns into a table to push to the database.
To create the agent_attorney table, we gather "agent_attorney_name", 'agent_attorney_city', 'agent_attorney_state' from the original dataset and generate id for each agent, then populate data into the database.
The employer_address_book has "employer_ id" and "employer_address_id" as both composite primary key and foreign key so we create a table includes these columns 'employer_name', 'employer_address', 'employer_city', 'employer_state', 'employer_postal_code', 'employer_country', 'employer_province' from original dataframe. Then merge this table with 'employer' table on 'employer_name' to get 'employer_id' and merge with employer_address on all columns in 'employer_addrress' table to get 'employer_address_id' so that we can create final table with 2 needed columns and push data into the database.
For the case_submission table we create a dataframe with all relevant columns from the original dataset and merge with employer on shared columns to get 'employer_id' , merge with 'agent_attorney' to get 'agent_attorney_id'. After that we gather all relevant columns into "case_submission" table and push data into the database.
Lastly, we create the 'h1b_record_book' data by gathering all relevant columns into a dataframe ,merge that dataframe with 'h1_request_record' data frame to get 'h1b_request_record_id' and merge again with 'employer' to get the 'employer_id' column. Then create 'h1b_record_book' table with 'h1b_request_record_id' and 'employer_id' columns and push to the database.

Analytics Applications

We came up with 10 scenarios and provided sample queries for answering the key questions.

1. Who are the employers who submitted the most cases (list top 5)?
   Queries
   SELECT DISTINCT(employer.employer_name),
   COUNT(case_submission.employer_id)
   from employer, case_submission
   WHERE employer.employer_id = case_submission.employer_id
   GROUP BY employer.employer_id, employer_name
   ORDER BY COUNT desc
   LIMIT 5;
   Insights

Employer Name	Number of Cases
Infosys limited	405
Tata Consultancy Services	245
Capgemini America Inc	186
Accenture LLP	142
Microsoft Corporation	130

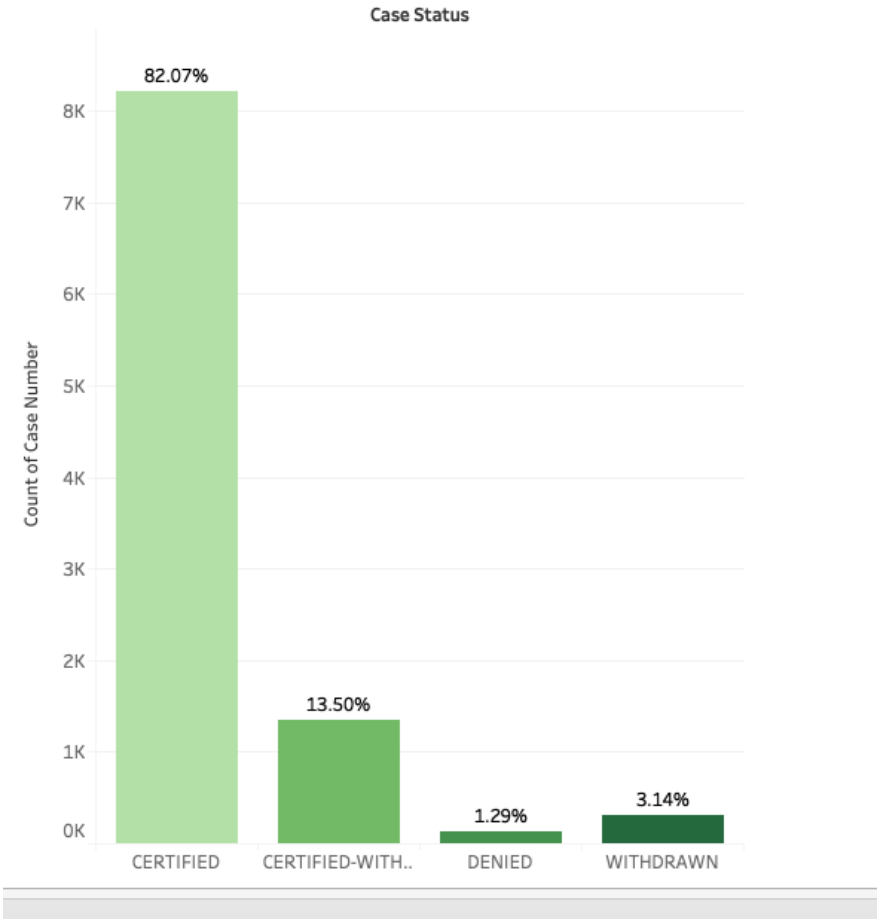


2. Percentage of cases in each case status?

   Queries

   select case_status, count(*) as cnt,
   round(count(*) * 100.0/ sum(count(*)) over (), 2) as percent
   from cases
   group by case_status;

   Insights



3. Top 5 job titles get submitted and approved

Top 5 submitted:

code:

```
select j.job_title_name, count(j.job_title_name)
from job_title j
join cases c on j.job_title_id=c.job_title_id
group by j.job_title_name
order by count desc
limit 5;
```

result:

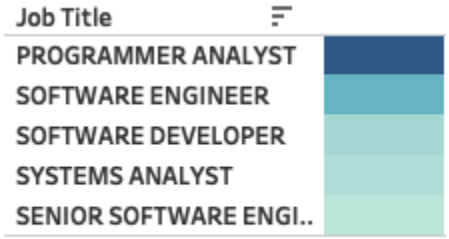
programmer analyst--793

software engineer--399

software developer--229

system analyst--197

senior software engineer--164



Top 5 approved:
code:

```
select j.job_title_name, count(j.job_title_name)
from job_title j
join cases c on j.job_title_id=c.job_title_id
where c.case_status='CERTIFIED'
group by j.job_title_name
order by count desc
limit 5;
```

result:
programmer analyst--593
software engineer--296
software developer--154
senior software engineer--143
system analyst--141

	Case Status
Job Title	CERTIFIED
PROGRAMMER ANALYST	593
SOFTWARE ENGINEER	296
SOFTWARE DEVELOPER	154
SENIOR SOFTWARE ENGI..	143
SYSTEMS ANALYST	141

4. Top 5 Job title with highest average wages

```
SELECT j.job_title_name,tmp_rk.avg_wage,tmp_rk.rk FROM
```

(SELECT job_title_id,avg_wage,RANK() OVER(ORDER BY avg_wage DESC) AS rk
FROM
(SELECT c.job_title_id,AVG(w.wage_rate_of_pay_from) avg_wage FROM cases c,
wages w
WHERE c.wage_case_id=w.wage_case_id GROUP BY c.job_title_id)tmp) as tmp_rk,
job_title AS j
WHERE tmp_rk.rk<=5
AND j.job_title_id=tmp_rk.job_title_id

5. Top 5 SOC names with highest average wages
   SELECT soc.soc_name, avg(wages.wage_rate_of_pay_from), RANK() OVER
(ORDER BY avg(wages.wage_rate_of_pay_from) DESC) AS wage_rank
FROM (soc natural join cases) join wages using (wage_case_id)
group by soc.soc_name
limit 5;
It turned out that Anesthesiologists, Surgeons, Environmental Science Teachers - Postsecondary, Chief Executives and Internists - General are the positions with the highest average wages.

6. Top 5 attorney get highest number of cases approved

```
select agent_attorney_id, agent_attorney_name, count(*) as num
from case_submission natural join agent_attorney natural join cases
where case_status = 'CERTIFIED'
group by agent_attorney_id, agent_attorney_name
order by num desc
limit 5;
```

7. Top 5 worksite city with most H1B cases submitted

```
select worksite_id,  count(*), worksite_city, worksite_state
from worksite_book natural join worksite
group by worksite_id, worksite_city, worksite_state
order by count desc limit 5;
```

8. The month that gets the most h1b applications

code:

```
select extract(month from case_submitted_date) as month, count(extract(month from
case_submitted_date))
from case_submission
group by month
order by count desc
limit 1;
```

result:
September--4861

9. Average waiting time to get the decision

```
SELECT AVG (decision_date-case_submitted_date) DAYS
FROM case_submission
```

61 days

10. Which wage level gets maximum number of certified cases?

I also looked into which wage level gets maximum number of certified cases.

```
select pw.pw_wage_level, count(c.case_status)
from prevailing_wages as pw natural join cases as c
where c.case_status = 'CERTIFIED'
group by c.case_status, pw.pw_wage_level
order by count(c.case_status) DESC
limit 1;
```

The result shows that employees with level II wage level gets max number of certified cases.

A list of agents' name and city for cases that have an agent represent employees with ranking

```
SELECT a.agent_attorney_name,a.agent_attorney_city,rk.rank_agent FROM
(SELECT *, RANK () OVER (ORDER BY ct DESC) as rank_agent
FROM
(SELECT agent_attorney_id,COUNT(agent_attorney_id) ct
FROM case_submission WHERE agent_representing_status = 'Y' GROUP BY
agent_attorney_id)as tmp) as rk, agent_attorney a
WHERE a.agent_attorney_id=rk.agent_attorney_id
```

Appendix

Python code for Extract, Transform, Load

Import necessary packages:

```
import pandas as pd
from sqlalchemy import create_engine
import matplotlib.pyplot as plt
import numpy as np
```

Connect to the database and read the csv file:

```
conn_url =
'postgresql://postgres:y6pj86qh@f19server.apan5310.com:50206/h1b_formal'
engine = create_engine(conn_url)
connection = engine.connect()
df = pd.read_csv('H-1B.csv')
df.columns = map(str.lower, df.columns)   #This is because all table names are
automatically turned into lower case in codio, so we make sure our code align with that.
```

Construct prevailing_wages table and populate the data into table:

```
prevailing_wages = df[["prevailing_wage", "pw_unit_of_pay",
"pw_wage_level","pw_source", "pw_source_year", "pw_source_other"]]
prevailing_wages = prevailing_wages.drop_duplicates()
prevailing_wages.insert(0, 'pw_id', range(1, 1 + len(prevailing_wages)))
prevailing_wages.to_sql(name='prevailing_wages',con=engine,
if_exists='append',index=False)
```

Construct wages table and populate data:

```
wages = df[["wage_rate_of_pay_from", "wage_rate_of_pay_to", "wage_unit_of_pay"]]
wages = wages.drop_duplicates()
wages.insert(0, 'wage_case_id', range(1, 1 + len(wages)))
wages.to_sql(name='wages',con=engine, if_exists='append',index=False)
```

Construct soc table:

In Excel, some soc_code is shown in the data format, the code below is to reverse those columns into the correct form:

```
df['soc_code']=np.where(df['soc_name']=='ADVERTISING AND PROMOTIONS
MANAGERS','11-2011',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='ADMINISTRATIVE SERVICES
MANAGERS','11-3012',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='MEDICAL AND HEALTH SERVICES
MANAGERS','11-9111',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='MARKETING MANAGERS','11-
2021',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='SALES MANAGERS','11-
2022',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='PUBLIC RELATIONS AND FUNDRAISING
MANAGERS','11-2030',df['soc_code'])
```

```
df['soc_code']=np.where(df['soc_name']=='COMPUTER AND INFORMATION
SYSTEMS MANAGERS','11-3021',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='FINANCIAL MANAGERS','11-
3031',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='INDUSTRIAL PRODUCTION
MANAGERS','11-3051',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='PURCHASING MANAGERS','11-
3061',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='TRANSPORTATION, STORAGE, AND
DISTRIBUTION MANAGERS','11-3071',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='HUMAN RESOURCES MANAGERS','11-
3121',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='TRAINING AND DEVELOPMENT
MANAGERS','11-3131',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='FARMERS, RANCHERS, AND OTHER
AGRICULTURAL MANAGERS','11-9013',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='CONSTRUCTION MANAGERS','11-
9021',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='EDUCATION ADMINISTRATORS,
ELEMENTARY AND SECONDARY','11-9032',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='EDUCATION ADMINISTRATORS,
POSTSECONDARY','11-9033',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='EDUCATION ADMINISTRATORS,
PRESCHOOL AND CHILDCARE','11-9031',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='EDUCATION ADMINISTRATORS, ALL
OTHER','11-9039',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='ARCHITECTURAL AND ENGINEERING
MANAGERS','11-9041',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='FOOD SERVICE MANAGERS','11-
9051',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='LODGING MANAGERS','11-
9081',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='NATURAL SCIENCES MANAGERS','11-
9121',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='PROPERTY, REAL ESTATE, AND
COMMUNITY ASSOCIATION','11-9141',df['soc_code'])
df['soc_code']=np.where(df['soc_name']=='SOCIAL AND COMMUNITY SERVICE
MANAGERS','11-9151',df['soc_code'])
```

```
df['soc_code']=np.where(df['soc_name']=='MANAGERS, ALL OTHER','11-9199',df['soc_code'])
```

Some of rows of soc_name have typo, for example, missing 's' at the end, so we find out which rows have this problem and then change their names to the correct ones:

```
df['soc_name']=np.where(df['soc_code']=='15-1121','COMPUTER SYSTEMS ANALYSTS', df['soc_name'])
df['soc_name']=np.where(df['soc_code']=='15-1131','COMPUTER PROGRAMMERS', df['soc_name'])
df['soc_name']=np.where(df['soc_code']=='15-1199','COMPUTER OCCUPATIONS, ALL OTHER', df['soc_name'])
df['soc_name']=np.where(df['soc_code']=='13-1111','MANAGEMENT ANALYSTS', df['soc_name'])
df['soc_name']=np.where(df['soc_code']=='15-1199.02','COMPUTER SYSTEMS ENGINEERS/ARCHITECTS', df['soc_name'])
df['soc_name']=np.where(df['soc_code']=='25-2021','ELEMENTARY SCHOOL TEACHERS, EXCEPT SPECIAL', df['soc_name'])
df['soc_name']=np.where(df['soc_code']=='41-9031','SALES ENGINEERS', df['soc_name'])
df['soc_name']=np.where(df['soc_code']=='17-2072','ELECTRONICS ENGINEERS, EXCEPT COMPUTER', df['soc_name'])
df['soc_name']=np.where(df['soc_code']=='15-1132','SOFTWARE DEVELOPERS, APPLICATIONS', df['soc_name'])
df['soc_name']=np.where(df['soc_code']=='15-1022','COMPUTER PROGRAMMERS, NON R&D', df['soc_name'])
df['soc_name']=np.where(df['soc_code']=='15-1142','NETWORK AND COMPUTER SYSTEMS ADMINISTRATORS', df['soc_name'])
df['soc_name']=np.where(df['soc_code']=='15-1034','SOFTWARE DEVELOPERS, APPLICATIONS, NON R&D', df['soc_name'])
df['soc_name']=np.where(df['soc_code']=='17-3023','ELECTRICAL AND ELECTRONIC ENGINEERING TECHNICIANS', df['soc_name'])
df['soc_name']=np.where(df['soc_code']=='17-2141','MECHANICAL ENGINEERS', df['soc_name'])
```

Populate data into soc table:

```
soc = df[["soc_code", "soc_name"]]
soc = soc.drop_duplicates()
soc.to_sql(name='soc',con=engine, if_exists='append',index=False)
```

Construct job_title table and populate data:

```
job_title = df[["job_title"]]
job_title = job_title.drop_duplicates()
job_title.insert(0, 'job_title_id', range(1, 1 + len(job_title)))
job_title = job_title.rename(columns = {"job_title": "job_title_name"})
job_title.to_sql(name='job_title',con=engine, if_exists='append',index=False)
```

Construct & deal with data in cases table:

In this step we first select all columns, including those in foreign key tables, so that we can join these columns in the future steps:

```
cases = df[["case_number", "original_cert_date", "employment_start_date",
"employment_end_date", "case_status", "visa_class", "soc_code", "full_time_position",
"wage_rate_of_pay_from", "wage_rate_of_pay_to", "wage_unit_of_pay",
"prevailing_wage", "pw_unit_of_pay", "pw_wage_level","pw_source",
"pw_source_year", "pw_source_other", "job_title"]]
```

Merge the needed columns to get id for different tables:

```
cases = pd.merge(cases, prevailing_wages,  how='left', on=["prevailing_wage",
"pw_unit_of_pay", "pw_wage_level","pw_source", "pw_source_year",
"pw_source_other"])
cases = pd.merge(cases, wages,  how='left', on=["wage_rate_of_pay_from",
"wage_rate_of_pay_to", "wage_unit_of_pay"])
job_title = job_title.rename(columns = {"job_title_name": "job_title"})
cases = pd.merge(cases, job_title,  how='left', on=["job_title"])
```

Rename columns to make sure they have aligned names:

```
cases = cases[["case_number", "original_cert_date", "employment_start_date",
"employment_end_date", "case_status", "visa_class", "job_title_id", "soc_code",
"full_time_position", "pw_id", "wage_case_id"]]
cases.to_sql(name='cases',con=engine, if_exists='append',index=False)
```

Construct worksite table:

```
worksite = df[["worksite_city", "worksite_county", "worksite_state",
"worksite_postal_code"]]
worksite = worksite.drop_duplicates()
worksite.insert(0, 'worksite_id', range(1, 1 + len(worksite)))
worksite.to_sql(name='worksite',con=engine, if_exists='append',index=False)
```

Construct worksite_book table:

```
worksite_book = df[["case_number", "worksite_city", "worksite_county",
"worksite_state", "worksite_postal_code"]]
worksite_book = pd.merge(worksite_book, worksite, how = "left", on = ["worksite_city",
"worksite_county", "worksite_state", "worksite_postal_code"])
worksite_book = worksite_book[["case_number", "worksite_id"]]
worksite_book.to_sql(name='worksite_book',con=engine,
if_exists='append',index=False)
```

Construct naics (North American Industry Classification System) table:

```
naics = df[['naics_code']]
naics = naics.drop_duplicates()
naics.insert(0, 'naics_id', range(1, 1 + len(naics)))
naics.to_sql(name='naics',con=engine, if_exists='append',index=False)
```

Construct h1b_request_record table:

```
h1b_request_record = df[['total_workers', 'new_employment', 'continued_employment',
'change_previous_employment', 'new_concurrent_employment', 'change_employer',
'amended_petition']]
h1b_request_record = h1b_request_record.drop_duplicates()
h1b_request_record.insert(0, 'h1b_request_record_id', range(1, 1 +
len(h1b_request_record)))
h1b_request_record.to_sql(name='h1b_request_record',con=engine,
if_exists='append',index=False)
```

Construct employer_address table:

```
employer_address = df[['employer_address', 'employer_city', 'employer_state',
'employer_postal_code', 'employer_country', 'employer_province']]
employer_address = employer_address.drop_duplicates()
employer_address.insert(0, 'employer_address_id', range(1, 1 +
len(employer_address)))
employer_address.to_sql(name='employer_address',con=engine,
if_exists='append',index=False)
```

Construct employer table, merge with nacis table to have 'naics_code' column:

```
employer = df[['employer_name', 'employer_business_dba', 'h1b_dependent',
'willful_violator', 'support_h1b', 'labor_con_agree','employer_phone',
'employer_phone_ext', 'naics_code']]
```

Merge on NAICS:

```
employer = pd.merge(employer, naics,  how='left', on=['naics_code'])
employer = employer[['employer_name', 'employer_business_dba', 'naics_id',
                'h1b_dependent', 'willful_violator', 'support_h1b', 'labor_con_agree']]
```

Populate data:

```
employer = employer.drop_duplicates()
employer.insert(0, 'employer_id', range(1, 1 + len(employer)))
employer.to_sql(name='employer',con=engine, if_exists='append',index=False)
```

Construct employer_phone and populate data:

```
employer_phone = df[['employer_name','employer_phone', 'employer_phone_ext']]
employer_phone = employer_phone.drop_duplicates()
employer_phone = pd.merge(employer_phone, employer, how='left',
on=['employer_name'])
employer_phone.insert(0, 'employer_phone_id', range(1, 1 + len(employer_phone)))
employer_phone = employer_phone[["employer_phone_id", "employer_id",
"employer_phone", "employer_phone_ext"]]
employer_phone.to_sql(name='employer_phone',con=engine,
if_exists='append',index=False)
```

Construct agent_attorney table and populate data:

```
agent_attorney = df[["agent_attorney_name", 'agent_attorney_city',
'agent_attorney_state']]
agent_attorney = agent_attorney.drop_duplicates()
agent_attorney.insert(0, 'agent_attorney_id', range(1, 1 + len(agent_attorney)))
agent_attorney.to_sql(name='agent_attorney',con=engine,
if_exists='append',index=False)
```

Construct employer_address_book table and populate data:

```
employer_address_book = df[['employer_name', 'employer_address', 'employer_city',
'employer_state', 'employer_postal_code', 'employer_country', 'employer_province']]
employer_address_book = pd.merge(employer_address_book, employer, how='left',
on=['employer_name'])
employer_address_book = pd.merge(employer_address_book, employer_address,
how='left', on=['employer_address', 'employer_city', 'employer_state',
'employer_postal_code', 'employer_country', 'employer_province'])
employer_address_book = employer_address_book.drop_duplicates()
```

```
employer_address_book = employer_address_book[["employer_id",
"employer_address_id"]]
employer_address_book.to_sql(name='employer_address_book',con=engine,
if_exists='append',index=False)
```

Construct case_submission table:
Merge with employer and agent_attorney table:

```
case_submission = df[["case_number", 'employer_name', 'employer_business_dba',
'naics_code',  'h1b_dependent', 'willful_violator', 'support_h1b', 'labor_con_agree',
"case_submitted", "decision_date", "agent_representing_employer",
"agent_attorney_name", 'agent_attorney_city', 'agent_attorney_state']]
Merge with employer table to get corresponding
case_submission = pd.merge(case_submission, employer, how='left',
on=['employer_name', 'employer_business_dba', 'h1b_dependent', 'willful_violator',
'support_h1b', 'labor_con_agree'])
case_submission = case_submission.drop_duplicates(subset = ["case_number",
"employer_name"])
case_submission = pd.merge(case_submission, agent_attorney, how='left',
on=["agent_attorney_name", 'agent_attorney_city', 'agent_attorney_state'])
case_submission = case_submission[["case_number", "employer_id",
"case_submitted", "decision_date", "agent_representing_employer",
"agent_attorney_id"]]
case_submission =
case_submission.rename(columns={"agent_representing_employer":
"agent_representing_status", "case_submitted": "case_submitted_date"})
case_submission.to_sql(name='case_submission',con=engine,
if_exists='append',index=False)
```

Construct h1b_record_book table:
Merge with h1b_request_record table to get corresponding h1b_request_record_id:

```
h1b_record_book = df[['employer_name', 'employer_business_dba',  'h1b_dependent',
'willful_violator', 'support_h1b', 'labor_con_agree', 'total_workers', 'new_employment',
'continued_employment', 'change_previous_employment',
'new_concurrent_employment', 'change_employer', 'amended_petition']]
h1b_record_book = pd.merge(h1b_record_book, h1b_request_record,  how='left',
on=['total_workers', 'new_employment', 'continued_employment',
```

```
'change_previous_employment', 'new_concurrent_employment', 'change_employer',
'amended_petition'])
h1b_record_book = pd.merge(h1b_record_book, employer,  how='left',
on=['employer_name', 'employer_business_dba', 'h1b_dependent', 'willful_violator',
'support_h1b', 'labor_con_agree'])
h1b_record_book = h1b_record_book[["employer_id", "h1b_request_record_id"]]
h1b_record_book = h1b_record_book.drop_duplicates()
h1b_record_book.to_sql(name='h1b_record_book',con=engine,
if_exists='append',index=False)
```