Slurm Jobs, Dask, and SQL

Matthew Seguin

Preliminary Importing of Packages

```
import os
import time
import duckdb as dd
import pandas as pd
import sqlite3 as sq
import matplotlib.pyplot as plt
```

1.

First we need to start our interactive session, this is the code I used for that. (I will first navigate to the folder for this problem set so that I can write a csv there later).

```
cd ./dir
srun --job-name=na --partition=### --ntasks-per-node=4 --mem-per-cpu=5G --pty /bin/bash
```

a.

i.

Now we copy the files in /datadir to a subdirectory of /tmp

```
# Make my subdirectory in tmp
mkdir /tmp/subdir
# Copy all files in given directory to tmp
scp /datadir/* /tmp/subdir
```

ii.

Now it is time to write the python code so first I start a python session

```
python
```

Now we write our dask code.

```
import re
import os
import dask
import pandas as pd

# Set evaluation method and resources for dask
nworkers = int(os.getenv("SLURM_NTASKS"))
dask.config.set(scheduler="processes", num_workers = nworkers)

# Make function to read the files and get only
# lines that have "Barack_Obama" in them
def read_and_filter(filename):
  print(filename)
  x = pd.read_csv(filename,
                  sep = "\\s+",
                  dtype = {0: "str",
                           1: "str"
                          },
                  compression = "gzip",
                  header = None,
                  quoting = 3)
  x = x[[re.search("Barack_Obama", str(i)) is not None for i in x[3]]]
  return(x)

# Directory for the data files
data_dir = "/datadir/"
#data_dir = "/tmp/subdir"

# Make a list of the paths to the files
file_names = [os.path.join(data_dir, f) for f in os.listdir(data_dir) if f.endswith(".gz")]

# Create tasks list
tasks = [dask.delayed(read_and_filter)(file_name) for file_name in file_names]

# Compute results
results = dask.compute(tasks)[0]

# Combine all resulting data frames
combined_results = pd.concat(results, ignore_index = True)
# Change to a datetime variable
combined_results[0] = pd.to_datetime(combined_results[0], format='%Y%m%d')
# Change to an hours, minutes, seconds time variable
combined_results[1] = pd.to_datetime(combined_results[1], format='%H%M%S').dt.time

# Write to csv
combined_results.to_csv("results1.csv", index = False)
```

```
# Show results
combined_results
```

```
         0         1           2                                                3     4         5
2008-11-04  07:00:00       en.v                          Special:Search/Barack_Obama   1.0   18651.0
2008-11-04  07:00:00         en  %20List_of_Barack_Obama_presidential_campaign_...   1.0    5047.0
2008-11-04  07:00:00         en  %20List_of_Barack_Obama_presidential_campaign_...   1.0    5050.0
2008-11-04  09:00:00  commons.m                               Image:Barack_Obama.jpg   3.0   27111.0
2008-11-04  09:00:00  commons.m                          Image:Barack_Obama_2004.jpg   1.0    7951.0
       ...       ...        ...                                                ...   ...       ...
2008-11-04  15:00:00         en  User:Giftlite/List_of_Nobel_laureates_who_endo...   4.0   13431.0
2008-11-04  15:00:00         en        User:Jfire/Kenyan_relatives_of_Barack_Obama   1.0    9704.0
2008-11-04  23:00:00         en  Talk:List_of_bills_sponsored_by_Barack_Obama_i...   3.0   33556.0
2008-11-04  04:00:00       pt.q                                        Barack_Obama   1.0     464.0
2008-11-04  14:00:00         nn                                        Barack_Obama  14.0  246723.0

[8274 rows x 6 columns]
```

We can see this looks like what we want and I will do a quick check that every row contains “Barack_Obama” later in bash.

Finally I quit out of the python session.

```
quit()
```

Now here is the checking of hte .csv file, we should see 8274 lines that have “Barack_Obama” and indeed we do.

```
grep "Barack_Obama" results1.csv | wc -l
```

```
8274
```

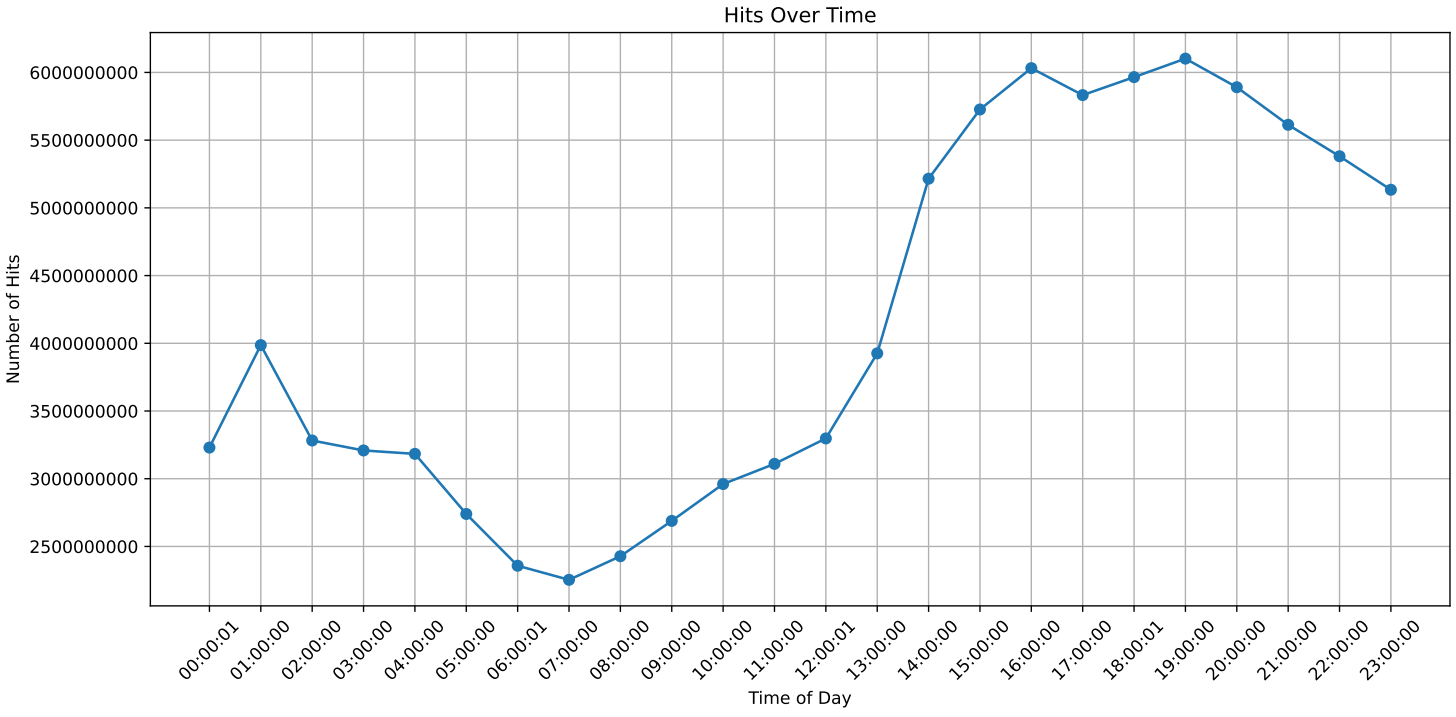
iii.

Now we read the data in and plot our results. I couldn't get a plot to show while using only the bash shell.

```
results = pd.read_csv("results1.csv")

# Concentrate results by time of day
hits_by_time = results.groupby("1")["5"].sum().reset_index()
hits_by_time.columns = ["Time", "Hits"]

# Graph results
# I assign some of these to a dummy variable because it was causing output I didn't want
_ = plt.figure(figsize=(12, 6))
_ = plt.plot(hits_by_time["Time"], hits_by_time["Hits"], marker = "o")
_ = plt.title("Hits Over Time")
_ = plt.xlabel("Time of Day")
_ = plt.ylabel("Number of Hits")
plt.ticklabel_format(style = "plain", axis = "y")
_ = plt.xticks(rotation=45)
plt.grid()
plt.tight_layout()
plt.show()
```



iv.

Now we remove all the files in the subdirectory we created in /tmp

```
rm -rf /tmp/subdir
```

b.

Now we repeat parts i and ii using sbatch. First I created a python file for the python code but I also made a .sh script for using sbatch. Here are the contents of each.

```
cat submit1.sh
```

```
#!/bin/bash

# SBATCH options

#SBATCH --job-name=na        # job name for queue (optional)
#SBATCH --partition=###      # partition (optional, default=low)
#SBATCH --error=ex.err       # file for stderr (optional)
#SBATCH --output=ex.out      # file for stdout (optional)
#SBATCH --time=00:10:00      # max runtime of job hours:minutes:seconds
#SBATCH --ntasks-per-node=4  # use 4 CPU cores
#SBATCH --mem-per-cpu=5G     # limit memory per cpu to 5GB
#SBATCH --wait               # don't continue until job finished

# Command(s) to run

# Make my subdirectory in tmp
mkdir /tmp/subdir
# Copy all files in given directory to tmp
scp /datadir/* /tmp/subdir # Run python script
python script1.py
# Remove files from tmp
rm -rf /tmp/subdir
```

```
cat script1.py
```

```
if __name__ == "__main__":
  import re
  import os
  import dask
  import pandas as pd

  # Set evaluation method and resources for dask
  nworkers = int(os.getenv("SLURM_NTASKS"))
  dask.config.set(scheduler = "processes", num_workers = nworkers)

  # Make function to read the files and get only
  # lines that have "Barack_Obama" in them
  def read_and_filter(filename):
    x = pd.read_csv(filename,
                    sep = "\\s+",
                    dtype = {0: "str",
                             1: "str"
                            },
                    compression = "gzip",
                    header = None,
```

```
                    quoting = 3)
    x = x[[re.search("Barack_Obama", str(i)) is not None for i in x[3]]]
    return(x)

  # Directory for the data files
  data_dir = "/tmp/subdir"

  # Make a list of the paths to the files
  file_names = [os.path.join(data_dir, f) for f in os.listdir(data_dir) if f.endswith(".gz")]

  # Create tasks list
  tasks = [dask.delayed(read_and_filter)(file_name) for file_name in file_names]

  # Compute results
  results = dask.compute(tasks)[0]

  # Combine all resulting data frames
  combined_results = pd.concat(results, ignore_index = True)
  # Change to a datetime variable
  combined_results[0] = pd.to_datetime(combined_results[0], format='%Y%m%d')
  # Change to an hours, minutes, seconds time variable
  combined_results[1] = pd.to_datetime(combined_results[1], format='%H%M%S').dt.time

  # Write to csv
  combined_results.to_csv("results1.csv", index = False)
```

Then even though it won't do much since we already ran everything here is the result of running our job with sbatch and a quick look at the csv file and a check that there are no errors.

```
sbatch submit1.sh
```

```
Submitted batch job ######
```

```
cat ex.err
```

```
head results1.csv
```

```
0,1,2,3,4,5
2008-11-04,11:00:00,de,Bild:Barack_Obama_in_Berlin.jpg,1.0,8191.0
2008-11-04,08:00:00,en.q,Barack_Obama,25.0,926695.0
2008-11-04,08:00:00,en.q,Barack_Obama&section=1,1.0,5427.0
2008-11-04,08:00:00,en,Electoral_history_of_Barack_Obama,2.0,23748.0
2008-11-04,08:00:00,ms,Barack_Obama,10.0,220166.0
2008-11-04,07:00:00,en.n,Former_Senator_Chafee_endorses_Barack_Obama,1.0,10400.0
2008-11-04,07:00:00,en.n,Grandmother_of_Barack_Obama_dies_at_86,138.0,1569449.0
2008-11-04,07:00:00,en,Early_life_and_career_of_Barack_Obama,186.0,7575876.0
2008-11-04,06:00:01,no,Barack_Obama,10.0,155844.0
```

```
grep "Barack_Obama" results1.csv | wc -l
```

```
8274
```

There are no errors and again we see there are 8274 rows.

2.

I will do this part with an sbatch job, first here are the scripts used.

```
cat submit2.sh
```

```
#!/bin/bash

# SBATCH options

#SBATCH --job-name=na          # job name for queue (optional)
#SBATCH --partition=###        # partition (optional, default=low)
#SBATCH --error=ex.err         # file for stderr (optional)
#SBATCH --output=ex.out        # file for stdout (optional)
#SBATCH --time=01:00:00        # max runtime of job hours:minutes:seconds
#SBATCH --ntasks-per-node=16   # use 16 CPU cores
#SBATCH --wait                 # don't continue until job finished

# Command(s) to run

# Run python script
python script2.py
```

```
cat script2.py
```

```
if __name__ == "__main__":
  import re
  import os
  import dask
  import time
  import pandas as pd
  import dask.bag as db
  import dask.multiprocessing

  # Set evaluation method and resources for dask
  nworkers = int(os.getenv("SLURM_NTASKS"))
  dask.config.set(scheduler = "processes", num_workers = nworkers)

  # Make function to get only lines whose
  # title has a match to regex in them
  def db_filter_title(line, regex = "Barack_Obama"):
    vals = line.split(" ")
    if len(vals) < 6:
        return(False)
    tmp = re.search(regex, vals[3])
    if tmp is None:
        return(False)
    return(True)

  # Make function to get only lines whose
  # language has a match to regex in them
  def db_filter_lang(line, regex = "en"):
    vals = line.split(" ")
```

```
    if len(vals) < 6:
        return(False)
    tmp = re.search(regex, vals[2])
    if tmp is None:
        return(False)
    return(True)

  # Make function to split the lines into a list
  def make_lst(line):
    return(list(line.split(" ")))

  # Directory for the data files
  data_dir = "/datadir/"

  # Start timer
  t0 = time.time()

  # Read in data
  x = db.read_text(data_dir + "part-0*gz", compression = "gzip")
  # Filter data
  dtypes = {"date": "object",
            "time": "object",
            "language": "object",
            "webpage": "object",
            "hits": "float64",
            "size": "float64"}
  filtered = x.filter(db_filter_title).filter(db_filter_lang)
  df = filtered.map(make_lst).to_dataframe(dtypes)
  results = df.compute()

  # End timer
  t1 = time.time()

  # Change to a datetime variable
  results["date"] = pd.to_datetime(results["date"], format='%Y%m%d')
  # Change to an hours, minutes, seconds time variable
  results["time"] = pd.to_datetime(results["time"], format='%H%M%S').dt.time

  # Write to csv
  results.to_csv("results2.csv", index = False)

  # Show total time for dask portion
  print("Time to run dask code:")
  print(pd.to_datetime(t1-t0, unit = "s").strftime('%H:%M:%S'))
```

Now running the job and looking at the output.

```
sbatch submit2.sh
```

```
Submitted batch job ######
```

```
cat ex.err
```

```
head results2.csv
```

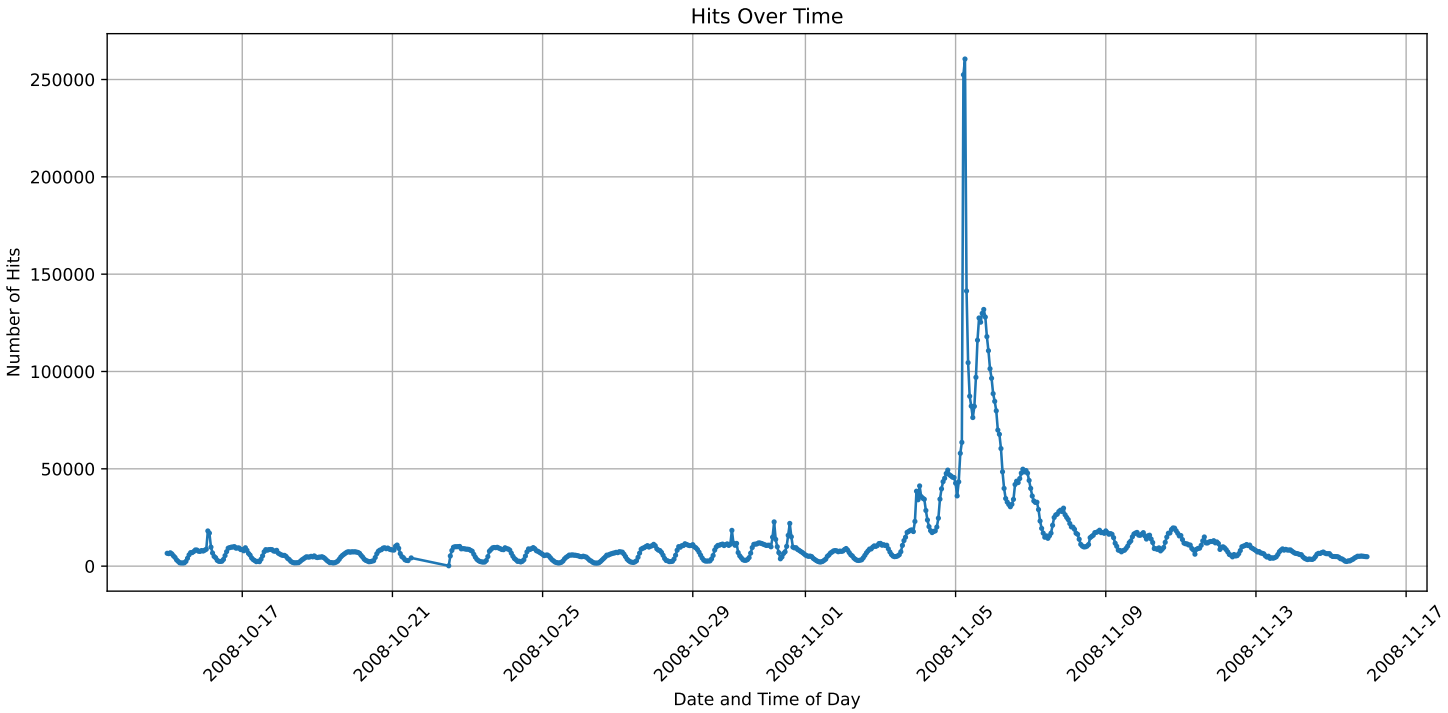
```
date,time,language,webpage,hits,size
2008-11-05,16:00:00,en,Special:RecentChangesLinked/Image:Barack_Obama.jpg,3.0,32122.0
2008-10-22,23:00:00,en,The_Case_Against_Barack_Obama,3.0,37572.0
2008-10-24,01:00:00,en,Illinois_Senate_career_of_Barack_Obama,18.0,338458.0
2008-11-04,11:00:00,en,User:Giftlite/List_of_Nobel_laureates_who_endorse_Barack_Obama,1.0,11870.0
2008-11-04,11:00:00,en,User:Jfire/Kenyan_relatives_of_Barack_Obama,1.0,9703.0
2008-11-01,01:00:00,en,"Special:Filepath/Barack_Obama_vs._John_McCain_(April_10,_2008).PNG",1.0,1169.0
2008-11-01,01:00:00,en,"Special:Filepath/Barack_Obama_vs._John_McCain_(April_12,_2008).PNG",1.0,1169.0
2008-11-01,01:00:00,en,"Special:Filepath/Barack_Obama_vs._John_McCain_(April_6,_2008).PNG",1.0,1171.0
2008-11-08,20:00:00,en,Special:PrefixIndex/Barack_Obama_pictures,1.0,4890.0
```

Now we make the time graphs with the data

```
results = pd.read_csv("results2.csv")
results["datetime"] = pd.to_datetime(results["date"] + " " + results["time"])

# Concentrate results by time of day
hits_by_time = results.groupby("datetime")["hits"].sum().reset_index()

# Graph results
# I assign some of these to a dummy variable because it was causing output I didn't want
_ = plt.figure(figsize=(12, 6))
_ = plt.plot(hits_by_time["datetime"], hits_by_time["hits"], marker = "o", markersize = 2)
_ = plt.title("Hits Over Time")
_ = plt.xlabel("Date and Time of Day")
_ = plt.ylabel("Number of Hits")
plt.ticklabel_format(style = "plain", axis = "y")
_ = plt.xticks(rotation=45)
plt.grid()
plt.tight_layout()
plt.show()
```



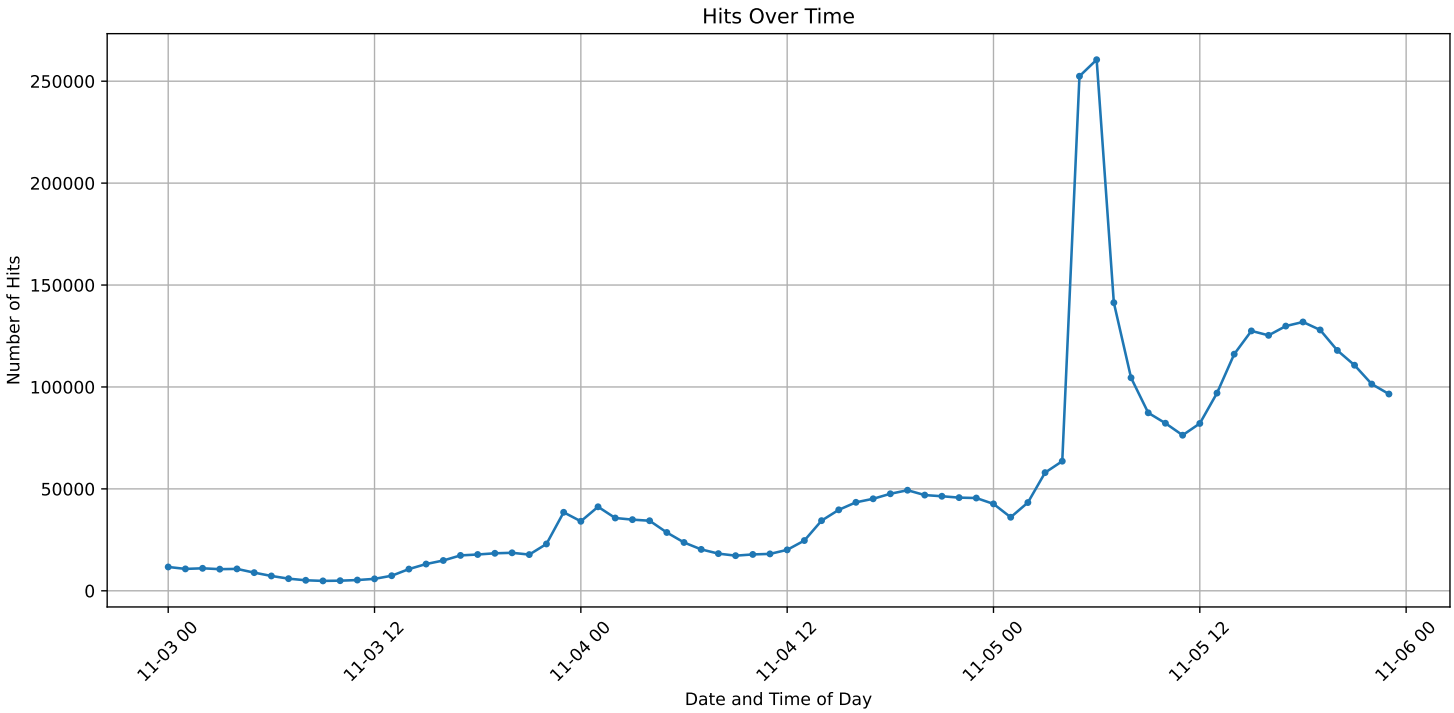
We can see there is a huge spike near 11/05 (which makes sense since Obama was elected on 11/04 at 11pm on Eastern time and this data is on GMT).

Now let us move in closer on that range

```
# Make start and end dates
start_date = pd.to_datetime("2008-11-03")
end_date = pd.to_datetime("2008-11-06")

# Keep only values that happened after start and before end
after_start = list(hits_by_time["datetime"] >= start_date)
before_end = list(hits_by_time["datetime"] <= end_date)
keep_dates = [after_start[i] and before_end[i] for i in range(len(after_start))]
election_results = hits_by_time[keep_dates]

# Graph results
# I assign some of these to a dummy variable because it was causing output I didn't want
_ = plt.figure(figsize=(12, 6))
_ = plt.plot(election_results["datetime"], election_results["hits"], marker = "o", markersize = 3)
_ = plt.title("Hits Over Time")
_ = plt.xlabel("Date and Time of Day")
_ = plt.ylabel("Number of Hits")
plt.ticklabel_format(style = "plain", axis = "y")
_ = plt.xticks(rotation=45)
plt.grid()
plt.tight_layout()
plt.show()
```


Hits Over Time
Number of Hits
250000
200000
150000
100000
50000
0
11-03 00
11-03 12
11-04 00
11-04 12
11-05 00
11-05 12
11-06 00
Date and Time of Day

3.

```
dir_path = os.getcwd()
db_filename = "stackoverflow-2021.db"

con = sq.connect(os.path.join(dir_path, db_filename))
db = con.cursor()

try:
  _ = db.execute("create view Q as \
                  select * from questions \
                  where questions.ownerid is not null")
except sq.OperationalError:
  print("View already exists")
```

```
View already exists
```

```
try:
  _ = db.execute("create view A as \
                  select * from answers \
                  where answers.ownerid is not null")
except sq.OperationalError:
  print("View already exists")
```

```
View already exists
```

```
try:
  _ = db.execute("create view U as \
                  select * from users \
                  where users.userid is not null")
except sq.OperationalError:
  print("View already exists")
```

```
View already exists
```

```
t0 = time.time()
_ = db.execute("SELECT * FROM \
                U WHERE userid IN \
                (SELECT ownerid FROM Q \
                EXCEPT \
                SELECT ownerid FROM A) \
                ")
tmp = db.fetchall()
len(tmp)
```

```
512091
```

```
t1 = time.time()
t1 - t0
```

```
11.908315896987915
```

```
t0 = time.time()
_ = db.execute("SELECT * FROM \
                (SELECT * FROM U) AS t1 \
                LEFT JOIN \
                (SELECT ownerid FROM Q \
                EXCEPT \
                SELECT ownerid FROM A) AS t2 \
                ON t1.userid = t2.ownerid \
                WHERE t2.ownerid IS NOT NULL \
                ")
tmp = db.fetchall()
len(tmp)
```

```
512091
```

```
t1 = time.time()
t1 - t0
```

```
12.893421411514282
```

```
db.close()
con.close()
```

We can see both ways are consistent with the length of the returned table.

b.

They each take very similar time to execute but the left join is slightly slower. It makes sense that both are similar in the time they take since the two are very similar in practice. The first approach uses the set operation except which is analogous in a way to the left join in the second approach then the rest consists of selecting certain rows that meet the same criteria.

Now I will try using duckdb.

```
dir_path = os.getcwd()
db_filename = "stackoverflow-2021.duckdb"

con = dd.connect(os.path.join(dir_path, db_filename))
db = con.cursor()

try:
  _ = db.execute("create view Q as \
                  select * from questions \
                  where questions.ownerid is not null")
except dd.CatalogException:
  print("View already exists")
```

```
View already exists
```

```
try:
  _ = db.execute("create view A as \
                  select * from answers \
                  where answers.ownerid is not null")
except dd.CatalogException:
  print("View already exists")
```

```
View already exists
```

```
try:
  _ = db.execute("create view U as \
                  select * from users \
                  where users.userid is not null")
except dd.CatalogException:
  print("View already exists")
```

```
View already exists
```

```
t0 = time.time()
_ = db.execute("SELECT * FROM \
                U WHERE userid IN \
                (SELECT ownerid FROM Q \
                EXCEPT \
                SELECT ownerid FROM A) \
                ")
tmp = db.fetchall()
len(tmp)
```

```
512091
```

```
t1 = time.time()
t1 - t0
```

```
2.5236964225769043
```

```
t0 = time.time()
_ = db.execute("SELECT * FROM \
                (SELECT * FROM U) AS t1 \
                LEFT JOIN \
                (SELECT ownerid FROM Q \
                EXCEPT \
                SELECT ownerid FROM A) AS t2 \
                ON t1.userid = t2.ownerid \
                WHERE t2.ownerid IS NOT NULL \
                ")
tmp = db.fetchall()
len(tmp)
```

```
512091
```

```
t1 = time.time()
t1 - t0
```

```
2.2548627853393555
```

As we can see the results are again consistent using duckdb. Also duckdb is significantly faster in both cases. I read that SQLite is an older package that is no longer getting support so this is likely why.