# Physical Data Model of the Local Cassandra database

# All CQL DDL & DML statements

CREATE TYPE position (

number text,

volume text,

pages text

);

CREATE TABLE IF NOT EXISTS publication\_by\_year\_and\_category(

paper\_key text,

title text,

year int,

type text,

authors\_pids map<int,text>,

authors\_orcids map<int,text>,

authors\_names map<int,text>,

category text,

publisher text,

position position,

ee text,

url text,

crossref text,

mdate date,

primary key((year,category),paper\_key)

) WITH CLUSTER BY (paper\_key ASC);

CREATE TABLE IF NOT EXISTS publication\_count\_unique\_author\_pair(

count int,

year int,

author1 text,

author2 text,

primary key((year),author1)

);

**Query 1 CQL DML statement:**

CREATE INDEX ON publication\_by\_year\_and\_category(values(authors\_pids));

SELECT \* FROM publication\_by\_year\_and\_category WHERE year = 2020 and category = ‘con’ and authors\_pids contains ‘169/9975’;

**Query 2 CQL DML statement:**

SELECT \* FROM publication\_count\_unique\_author\_pair WHERE year = 2020 AND author1 = '56/2998-1' AND author2 = '32/5712';

**Logs**

CREATE TYPE IF NOT EXISTS author (

name text,

orcid text,

pid text,

);

Select CREATE TABLE IF NOT EXISTS volume\_update(

timestamp text,

total\_new int,

total\_unique int,

PRIMARY KEY (timestamp)

);\* from publication\_update limit 5;

# CREATE TABLE IF NOT EXISTS publication\_update(

timestamp text,

title text,

authors map<int,frozen<author>>,

ee set<text>,

primary key((timestamp,title))

);’

Select \* from volume\_update;

Select \* from publication\_update limit 5;

**Proxy Queries: Python Codes and CQL DDL and DML statemenets**

Q1

session.execute('''

CREATE TABLE IF NOT EXISTS q1(

author text,

count int,

primary key(author)

);

''')

import ast

result = session\_offline.execute(

'''

SELECT authors\_pids FROM publication\_by\_year\_and\_category WHERE year IN (2012,2013,2014,2015,2016,2017,2018,2019,2020) and category in ('book','referenc','con','homepage','journal','serie') ;

'''

)

l = 0

for row in result:

if row is None:

continue

if row.authors\_pids is None:

continue

d = dict(row.authors\_pids)

x =[]

for element in d.values():

x.append(element)

if len(x) >= 3:

if x[2] == '40/2499' :

l = l+1

s = session.prepare('''

INSERT INTO q1 (author,count) VALUES (?,?);

''')

session.execute(s,['40/2499',l])

Q2

session.execute('''

CREATE TABLE IF NOT EXISTS q2(

author text,

count int,

primary key(author)

);

''')

import ast

result = session\_offline.execute(

'''

SELECT authors\_pids FROM publication\_by\_year\_and\_category WHERE year IN (2018,2019,2020,2021,2022) and category in ('book','referenc','con','homepage','journal','serie') ;

'''

)

l = 0

for row in result:

if row is None:

continue

if row.authors\_pids is None:

continue

d = dict(row.authors\_pids)

x =[]

for element in d.values():

x.append(element)

if len(x) >= 2:

if x[1] == 'o/BengChinOoi' :

l = l+1

s = session.prepare('''

INSERT INTO q2 (author,count) VALUES (?,?);

''')

session.execute(s,['40/2499',l])

Q3

session.execute('''

CREATE TABLE IF NOT EXISTS q3(

author1 text,

author2 text,

count int,

primary key(author1,author2)

);

''')

result = session\_offline.execute(

'''

SELECT \* from publication\_count\_unique\_author\_pair WHERE year in (2010,2011,20212,2013,2014,2015,2016,2017,2018,2019,2020,2021,2022) and author1 = '40/2499'

'''

)

l = []

for row in result:

l.append([row[0],row[1],row[2],row[3]])

df = pd.DataFrame(l,columns=["year","author1","author2","count"])

df = df.groupby("author2").sum().reset\_index()

cs\_researchers = pd.read\_csv('~/Documents/is3107-2/project/cs\_researchers.csv')

l= []

for i,row in cs\_researchers.iterrows():

result = session\_offline.execute(''' SELECT \* from publication\_count\_unique\_author\_pair

WHERE year in (2010,2011,2012,2013,2014,2015,2016,2017,2018,2019,2020,2021,2022)

AND author1 = '{}'

AND author2 = '40/2499' ;

'''.format(row[0]))

for r in result:

print(r)

l.append([r[0],r[1],r[2],r[3]])

df2 = pd.DataFrame(l,columns = columns)

df2.author2 = df2.author1

df2 = df2.groupby("author2").sum().reset\_index()

df = pd.concat([df,df2])

for \_,r in df.iterrows():

s = session.prepare('''

INSERT INTO q3 (author1,author2,count) VALUES (?,?,?);

''')

print(pd.Series(['40/2499',r.author2,r['count']]))

session.execute(s,['40/2499',r.author2,r['count']])

Q4

session.execute('''

CREATE TABLE IF NOT EXISTS q4(

year int,

author1 text,

author2 text,

count int,

primary key(author1,author2)

);

''')

result = session\_offline.execute(

'''

SELECT \* from publication\_count\_unique\_author\_pair WHERE year = 2020 and author1 = 'o/BengChinOoi'

'''

)

l = []

for row in result:

l.append([row[0],row[1],row[2],row[3]])

df = pd.DataFrame(l,columns=["year","author1","author2","count"])

cs\_researchers = pd.read\_csv('~/Documents/is3107-2/project/cs\_researchers.csv')

l= []

for i,row in cs\_researchers.iterrows():

result = session\_offline.execute(''' SELECT \* from publication\_count\_unique\_author\_pair

WHERE year = 2020

AND author1 = '{}'

AND author2 = 'o/BengChinOoi' ;

'''.format(row[0]))

for r in result:

l.append([r[0],r[1],r[2],r[3]])

df2 = pd.DataFrame(l,columns = columns)

df2.author2 = df2.author1

df2 = df2.groupby("author2").sum().reset\_index()

print(df2)

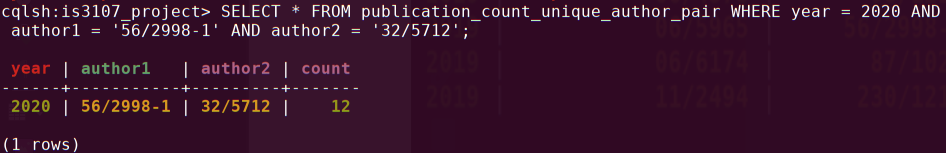
df = pd.concat([df,df2])

for \_,r in df.iterrows():

s = session.prepare('''INSERT INTO q4 (year,author1,author2,count) VALUES (?,?,?,?);’'')

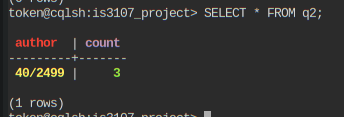
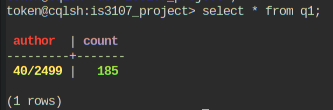
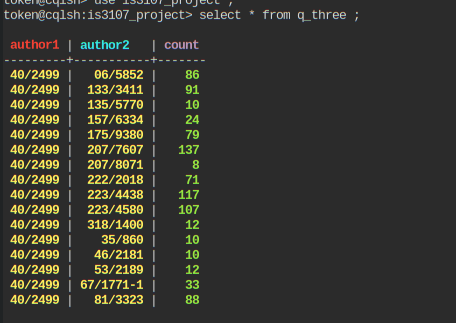
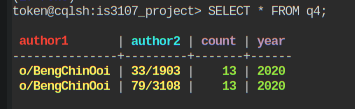
session.execute(s,[r.year,'o/BengChinOoi',r.author2,r['count']])

# Snapshots of tables with data for both local and cloud database

******

# Snapshots of proxy query outcomes stored in the cloud tables

Q1 Q2

Q3  Q4

# The rationales that you have considered for the design of the pipeline

They are separated in such a way so when the transform process fails, it will need to redo loading of data nor when the load process fails, it will not need to redo the extract task. If one a loading process to one table fails but the other pass, at least there is still some data inserted. In general when they are combined with one way or another, if a part of that crashes, the other parts will at least be saved.

**Unique data, to be deleted data and new data “caches”**

Here in the Transform task using the downloaded XML files from the previous extract task. This dag keeps a “caches” in terms of a csv file that stores what are the unique rows so if said rows are not found in the new downloaded data, it can be put into another “cache” where it is marked to be deleted from the db in the next step. if the new downloaded information if it exist in the unique rows, that new information will be marked to be inserted in another “cache” in the db in the next step.

Each table has it’s own unique set of caches so that when one branch fails(see graph), the other cache won’t be affected.

This dag runs slows on the first run as it has to download and insert all data but both mechanism insertion of specific new data and deletion of specific data will save a lot of time on subsequent runs.

#### q2\_count\_thing\_op is it’s own task as it is slower at 13 minutes and only “load\_into\_count\_table” needs so it decouples from the other branch as shown in the graph below.

The log task will only process the “new data cache” and as mentioned, will be more efficient on subsequent runs of the task.

**Limitations**

It is assumed that the publication details won’t change over time and these details are accurate so there will not be checking for changes in the publication details.

And some authors’ pids returns 404 pages instead of XMLs. This will not be rectified by trying to find the correct authors. It is assumed that they are wrong pids and name, just like nulls or something.

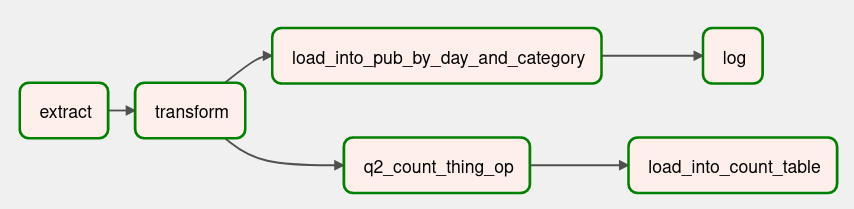
The design of the tables could be improved as In log task code, there is lot of data wrangling just to get it to the specification as mentioned in the project question paper. There could be an easier way by constructing the correct DDL and DML statements.

The ELT should have been used as the transformation part takes a long time and should be done later. The pipeline right now is in ETL like.

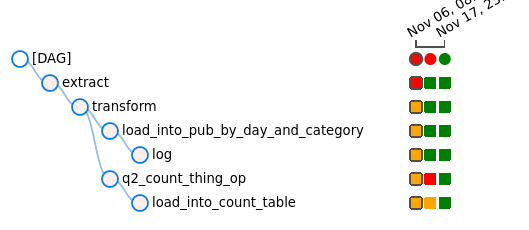
q2\_count\_thing() in between transform() and load\_into\_count\_table() and it is always taking the whole unique list and forming the rows. With the limited amount of authors to capture in the system, it will not be very slow as stated below, the current implementation took about 13 minutes only on a separate branch. That could be decent enough.

I do not have enough time or mental capacity to update only the certain row that has the new publication collaboration between author x and author y. Maybe The table attributes should be author1, author2, paper\_id, title, year and do “SELECT COUNT(\*) WHERE author1 = ‘name1’ AND author2 = ‘name2’ and year = 20XX”, as a conjecture, that may be more efficient.

# Snapshot of the graph visualization of the pipeline



# Snapshot of the tree view of the pipeline after triggering the DAG



# Run time of each step in the pipeline

|  |  |
| --- | --- |
| extract | 11 minutes and 34 seconds |
| transform | 2minutes |
| Q2 count thing op | 12 minutes and 57 seconds |
| load into count table | 4 seconds |
| load into pub by day and category | <1 seconds |
| log | 3 seconds |