SQL Project

Problem Statement

I am going to use SQL functionality to look into the issue of prioritisation for the Know Your Customer (KYC) reviews as well as the potential use of SQL to determine their book of work.

Overview

Every one to three years all clients that interact with by UBS have to go through the appropriate due diligence checks. These are completed to makes sure there are no potential risk issues when dealing with the client. This can be related to Anti-Money Laundering (AML) risk, Politically Exposed People (PEP) or other Financial Crime. Low AML risk clients are reviewed every three years whilst Medium up to Higher risk are done on a yearly basis. This risk rating also determines the seniority of the rank required by the business in order to 'sponsor' the client so they can help with source the necessary documents required for the KYC review.

The client data for the is kept on a SQL database with a number of tables. This system known locally as "Entity Master" (EM). This is where all the necessary information for legal entities the bank is trading with is stored. Clients can have a number of different legal entities and all which might do a different type of business. One could be a Hong Kong branch of an Investment bank and another could be the American Asset Manager. These can be two completely separate entities for the same client, and both would require different standards of KYC review.

I have write access to this system as part of my role within the offboarding team as we make direct changes to this system when we offboard clients however I cannot get direct access to tables to pull data for this exercise so I will be using a work around to demonstrate how SQL can be deployed to help with creating risk packs. I have to take a sole extract from the system which I will manipulate in order to re-create the EM setup. I will split this data into two tables the two tables to demonstrate how I can this problem can be addressed in a more production environment.

This is a potential proof of concept which can be presented to the business for justification of why having read-only access to the database could benefit UBS' Operations team in the future.

Sourcing the data

- This data has been sourced the UBS Legal Entity system known as "Entity Master" (EM).
- There is Client Identifying Data (CID) within data so as soon as I know this has been imported correct I will be removing the CID to prevent any breach of UBS data policy.
- The CID data can be added back if required by a merging the original DataFrame with the final result as number are unique for all UBS clients.

Tasks

In order to complete this analysis I need to complete the following tasks:

1. Import the libraries and data.

1. Analysis and clean the data
    - I need to check the data is in a usable format make any amendments required for my analysis.

1. Create the databases
    - I need to create the databases in order to recreate the data structure found within UBS.

1. Perform the SQL queries
    - To pull the data into the workbook.

1. Conclusion and next steps
    - How can this data be used? What are the next steps?

Exploring the data

Importing the libaries

In [1]:

```
# import sqlite to use in Jupyter notebook
import sqlite3

#import pandas for data frame manipulation
import pandas as pd
```

In [2]:

```
# Importing the data
df = pd.read_csv('kyc_data.csv', encoding='ANSI')
```

In [3]:

```
# Checking the data has been imported.
df.head(0)
```

Out[3]:

CE Number	CE Name	Client risk rating	Next Review Date	UBS Relationship Status	PPOB Address Country	Capacity	KYC Vetting	Non KYC Vetted Reason

I have made sure not to show any CID with the above. Once the column has been dropped I do another check to make sure the data has been imported correctly.

In [4]:

```
# Dropping client identifying data (CID) into order to be comply with UBS' data policy
df = df.drop('CE Name', 1)
```

In [5]:

```
# Renaming the columns for ease
df.rename(columns={'CE Number':'ce_no',
                   'Client risk rating':'risk',
                   'Next Review Date':'review_date',
                   'UBS Relationship Status':'status',
                   'PPOB Address Country':'ppob',
                   'Capacity':'capacity',
                   'KYC Vetting':'kyc_vetting',
                   'Non KYC Vetted Reason':'non_kyc_reason'}, inplace = True)
```

In [6]:

```
# Checking the amendments to the data frame have been successful.
df.head()
```

Out[6]:

	ce_no	risk	review_date	status	ppob	capacity	kyc_vetting	non_k
0	1230383	Low	30/11/2021	Active	Singapore	,Principal,Underlying Principal	Yes	
1	1230384	Low	06/07/2018	Active	Switzerland	,Principal,	Yes	
2	1230394	Low	01/01/2018	Active	United States	,,Underlying Principal	Yes	
3	1230403	Medium	13/01/2021	Active	Greece	Agent,Principal,	Yes	
4	1230404	Low	06/06/2020	Active	United States	Agent,Principal,	Yes	

The data appears to have been imported correctly so I will move onto checking the data tapes as I might need to amend them for ease of use later.

In [7]:

```
# checking the data types
df.dtypes
```

Out[7]:

```
ce_no             int64
risk             object
review_date      object
status           object
ppob             object
capacity         object
kyc_vetting      object
non_kyc_reason   object
dtype: object
```

The data as imported as expected. The 'ce_no' being integers will be useful later.

I am going to change the KYC review date into date time to make sure there will be no errors later due to the data potentially being read in the American MM-DD format. The data in EM has exported as DD-MM. If I need to convert to the YYYY-MM-DD format I will later on.

In [8]:

```
# converting 'review_date' to datetime
df['review_date'] = pd.to_datetime(df['review_date'])
```

In [9]:

```
# checking conversion
df.dtypes
```

Out[9]:

```
ce_no                      int64
risk                      object
review_date       datetime64[ns]
status                    object
ppob                      object
capacity                  object
kyc_vetting               object
non_kyc_reason            object
dtype: object
```

In [10]:

```
# checking the data
df.head(5)
```

Out[10]:

	ce_no	risk	review_date	status	ppob	capacity	kyc_vetting	non_k
0	1230383	Low	2021-11-30	Active	Singapore	,Principal,Underlying Principal	Yes	
1	1230384	Low	2018-06-07	Active	Switzerland	,Principal,	Yes	
2	1230394	Low	2018-01-01	Active	United States	,,Underlying Principal	Yes	
3	1230403	Medium	2021-01-13	Active	Greece	Agent,Principal,	Yes	
4	1230404	Low	2020-06-06	Active	United States	Agent,Principal,	Yes	

SQLite

The data has imported correctly and is in a useable format. I am going to use sqlite to create my database tables and re-pull the data using SQL queries.

In [11]:

```
# Connect to the database
conn = sqlite3.connect("kyc_database")
```

In [12]:

```
#commit dataframe to database
df.to_sql('kyc_database',conn)
```

I will '#' this out if I need to rerun as it will try and create a database with the same name. If it runs it will create an error message.

In [13]:

```
# Create curser object
cur = conn.cursor()
```

In [14]:

```
# test sql connectivity
dftest = pd.read_sql_query('SELECT * FROM kyc_database LIMIT 10',conn)
```

In [15]:

```
dftest.head()
```

Out[15]:

	index	ce_no	risk	review_date	status	ppob	capacity	kyc_vetting
0	0	1230383	Low	2021-11-30 00:00:00	Active	Singapore	,Principal,Underlying Principal	Yes
1	1	1230384	Low	2018-06-07 00:00:00	Active	Switzerland	,Principal,	Yes
2	2	1230394	Low	2018-01-01 00:00:00	Active	United States	,,Underlying Principal	Yes
3	3	1230403	Medium	2021-01-13 00:00:00	Active	Greece	Agent,Principal,	Yes
4	4	1230404	Low	2020-06-06 00:00:00	Active	United States	Agent,Principal,	Yes

The data as pulled from the kyc_database correctly. I am going to run a similar query in a different manner as it will allow me to manipulate the data I need to pull better.

In [16]:

```
cur.execute('''SELECT *
               FROM kyc_database
               LIMIT 10; ''')
kyc_df = pd.DataFrame(cur.fetchall())
kyc_df.columns = [x[0] for x in cur.description]
kyc_df
```

Out[16]:

	index	ce_no	risk	review_date	status	ppob	capacity	kyc_vetting
0	0	1230383	Low	2021-11-30 00:00:00	Active	Singapore	,Principal,Underlying Principal	Yes
1	1	1230384	Low	2018-06-07 00:00:00	Active	Switzerland	,Principal,	Yes
2	2	1230394	Low	2018-01-01 00:00:00	Active	United States	,,Underlying Principal	Yes
3	3	1230403	Medium	2021-01-13 00:00:00	Active	Greece	Agent,Principal,	Yes
4	4	1230404	Low	2020-06-06 00:00:00	Active	United States	Agent,Principal,	Yes
5	5	1230407	Low	2020-03-23 00:00:00	Active	United States	Agent,Principal,	Yes
6	6	1230410	Low	2022-10-31 00:00:00	Active	Portugal	,Principal,	Yes
7	7	1230412	Low	2020-03-20 00:00:00	Active	United States	Agent,,	Yes
8	8	1230417	Low	2020-10-27 00:00:00	Active	Germany	,Principal,	Yes
9	9	1230419	Low	2020-08-13 00:00:00	Active	Denmark	,Principal,	Yes

I am now going to pull the 'Higher' AML risk clients as these required the most due-diligence.

In [17]:

```
cur.execute('''SELECT ce_no,
                      risk,
                      review_date
               FROM kyc_database
               WHERE risk = 'Higher'
               ORDER BY review_date DESC; ''')
higher_df = pd.DataFrame(cur.fetchall())
higher_df.columns = [x[0] for x in cur.description]
higher_df
```

Out[17]:

	ce_no	risk	review_date
0	7720649	Higher	2023-10-03 00:00:00
1	6246638	Higher	2023-06-04 00:00:00
2	9575671	Higher	2022-11-22 00:00:00
3	1993153	Higher	2022-11-19 00:00:00
4	7782704	Higher	2022-11-19 00:00:00
...	...	...	...
1059	9922606	Higher	None
1060	9922610	Higher	None
1061	9922611	Higher	None
1062	9922612	Higher	None
1063	9946507	Higher	None

1064 rows × 3 columns

I can see some clients have pulled back without a KYC reviews date. This might be funds for pre-approved Agents (when a fund is traded for on behalf of a client of UBS) so they're not really required for this exercise. I will change my query to pull the data without these.

In [18]:

```
cur.execute('''SELECT ce_no,
                      risk,
                      review_date,
                      kyc_vetting,
                      ppob
               FROM kyc_database
               WHERE risk = 'Higher' AND ppob = 'United Kingdom' AND kyc_vetting = 'Ye
s'
               ORDER BY review_date ASC; ''')
higher_df = pd.DataFrame(cur.fetchall())
higher_df.columns = [x[0] for x in cur.description]
higher_df
```

Out[18]:

	ce_no	risk	review_date	kyc_vetting	ppob
0	8231495	Higher	2017-12-11 00:00:00	Yes	United Kingdom
1	7159027	Higher	2017-12-15 00:00:00	Yes	United Kingdom
2	7324276	Higher	2018-01-04 00:00:00	Yes	United Kingdom
3	1478639	Higher	2018-03-31 00:00:00	Yes	United Kingdom
4	1493163	Higher	2018-06-19 00:00:00	Yes	United Kingdom
5	7860145	Higher	2018-09-29 00:00:00	Yes	United Kingdom
6	1278374	Higher	2018-10-24 00:00:00	Yes	United Kingdom
7	1230809	Higher	2018-10-27 00:00:00	Yes	United Kingdom
8	1235891	Higher	2018-11-09 00:00:00	Yes	United Kingdom
9	8762747	Higher	2019-06-20 00:00:00	Yes	United Kingdom
10	7670224	Higher	2019-09-21 00:00:00	Yes	United Kingdom
11	6744406	Higher	2019-09-28 00:00:00	Yes	United Kingdom
12	1362760	Higher	2019-12-18 00:00:00	Yes	United Kingdom
13	3341258	Higher	2019-12-30 00:00:00	Yes	United Kingdom
14	6620324	Higher	2020-01-08 00:00:00	Yes	United Kingdom
15	8259051	Higher	2020-02-25 00:00:00	Yes	United Kingdom
16	6544035	Higher	2020-03-15 00:00:00	Yes	United Kingdom
17	8703759	Higher	2020-03-26 00:00:00	Yes	United Kingdom
18	9960916	Higher	2020-03-27 00:00:00	Yes	United Kingdom
19	8416200	Higher	2020-04-21 00:00:00	Yes	United Kingdom
20	1236251	Higher	2020-04-22 00:00:00	Yes	United Kingdom
21	7861753	Higher	2020-04-28 00:00:00	Yes	United Kingdom
22	7232599	Higher	2020-05-15 00:00:00	Yes	United Kingdom
23	6568028	Higher	2020-05-28 00:00:00	Yes	United Kingdom
24	6826873	Higher	2020-05-30 00:00:00	Yes	United Kingdom
25	6953349	Higher	2020-06-05 00:00:00	Yes	United Kingdom
26	6906311	Higher	2020-06-20 00:00:00	Yes	United Kingdom
27	9795769	Higher	2020-06-21 00:00:00	Yes	United Kingdom
28	6795892	Higher	2020-07-06 00:00:00	Yes	United Kingdom
29	7435042	Higher	2020-08-04 00:00:00	Yes	United Kingdom
30	6641014	Higher	2020-09-20 00:00:00	Yes	United Kingdom
31	7744734	Higher	2020-10-12 00:00:00	Yes	United Kingdom
32	8378224	Higher	2020-10-30 00:00:00	Yes	United Kingdom
33	6658853	Higher	2020-10-31 00:00:00	Yes	United Kingdom
34	8062447	Higher	2020-11-15 00:00:00	Yes	United Kingdom
35	6581651	Higher	2020-11-25 00:00:00	Yes	United Kingdom
36	6839138	Higher	2020-11-29 00:00:00	Yes	United Kingdom

	ce_no	risk	review_date	kyc_vetting	ppob
37	8140455	Higher	2020-12-03 00:00:00	Yes	United Kingdom
38	8356020	Higher	2020-12-03 00:00:00	Yes	United Kingdom
39	6690486	Higher	2020-12-17 00:00:00	Yes	United Kingdom
40	1231454	Higher	2020-12-18 00:00:00	Yes	United Kingdom
41	7216326	Higher	2020-12-27 00:00:00	Yes	United Kingdom
42	6704450	Higher	2021-01-20 00:00:00	Yes	United Kingdom
43	1804046	Higher	2021-01-23 00:00:00	Yes	United Kingdom
44	6676294	Higher	2021-02-13 00:00:00	Yes	United Kingdom
45	7232934	Higher	2021-02-19 00:00:00	Yes	United Kingdom
46	6725664	Higher	2021-02-28 00:00:00	Yes	United Kingdom
47	9609256	Higher	2021-04-19 00:00:00	Yes	United Kingdom
48	8390460	Higher	2021-08-01 00:00:00	Yes	United Kingdom
49	7331859	Higher	2022-08-21 00:00:00	Yes	United Kingdom
50	1278359	Higher	2022-10-18 00:00:00	Yes	United Kingdom

The above query demonstrates that SQL can be used by the Operations business in the United Kingdom to find out when the Higher Risk clients are up for KYC review. I know that these clients have to be reviewed on a yearly bases. I can see two dates in 2023 which should not be there. I can take these to the KYC team and ask them to investigate as this looks like data quality issue based on their own records. I can also ask the KYC to investigate the clients which have expired review dates where they are in the pipeline. If they are not actively being reviewed we can recommend to the business that these clients should be offboarded to reduce the risk appetite for the bank.

SQLite to query multiple tables

I am now going to split my table in to two databases to demostate how this can be used in business context

In [19]:

```
# Create a copy of the data drame before dropping the columns
df2 = df.copy(deep=True)
```

In [20]:

```
# Create a copy of the data drame before dropping the columns
df3 = df.copy(deep=True)
```

As these are just copies of the original DataFrame I will drop columns from them both to create normalised tables

In [21]:

```
#drop the columns from dataframe
df2.drop(['ppob', 'capacity','kyc_vetting','non_kyc_reason'], axis = 1, inplace = True)
```

In [22]:

```
#drop coloumns from dataframe 2
df3.drop(['risk', 'review_date','status'], axis = 1, inplace=True)
```

In [23]:

```
df2.head(5)
```

Out[23]:

	ce_no	risk	review_date	status
0	1230383	Low	2021-11-30	Active
1	1230384	Low	2018-06-07	Active
2	1230394	Low	2018-01-01	Active
3	1230403	Medium	2021-01-13	Active
4	1230404	Low	2020-06-06	Active

In [24]:

```
df3.head(5)
```

Out[24]:

	ce_no	ppob	capacity	kyc_vetting	non_kyc_reason
0	1230383	Singapore	,Principal,Underlying Principal	Yes	NaN
1	1230384	Switzerland	,Principal,	Yes	NaN
2	1230394	United States	,,Underlying Principal	Yes	NaN
3	1230403	Greece	Agent,Principal,	Yes	NaN
4	1230404	United States	Agent,Principal,	Yes	NaN

My two tables are now different and much more like an RDMS system which has been normalised. The two tables I have created are:

- AML Risk Rating
- PPoB (Principle Place of Business)

I am going to use sqlite to connect to these tables, pull new data and manipulate the data to create risk views for the KYC team.

In [25]:

```
# Connect to the database
conn2 = sqlite3.connect("risk")
```

In [26]:

```
# Connect to the database
conn3 = sqlite3.connect("ppob")
```

In [27]:

```
# Commit the dataframe to the SQL dataframe
df2.to_sql('risk',conn2)
```

In [28]:

```
# Commit the dataframe to the SQL dataframe
df3.to_sql('ppob',conn3)
```

In [29]:

```
# test sql connectivity
dftest = pd.read_sql_query('SELECT * FROM risk LIMIT 10',conn2)
```

In [30]:

```
#check test data has been pulled
dftest.head()
```

Out[30]:

	index	ce_no	risk	review_date	status
0	0	1230383	Low	2021-11-30 00:00:00	Active
1	1	1230384	Low	2018-06-07 00:00:00	Active
2	2	1230394	Low	2018-01-01 00:00:00	Active
3	3	1230403	Medium	2021-01-13 00:00:00	Active
4	4	1230404	Low	2020-06-06 00:00:00	Active

In [31]:

```
# test sql connectivity
dftest = pd.read_sql_query('SELECT * FROM ppob LIMIT 10',conn3)
```

In [32]:

```
#check test data has been pulled
dftest.head()
```

Out[32]:

	index	ce_no	ppob	capacity	kyc_vetting	non_kyc_reason
0	0	1230383	Singapore	,Principal,Underlying Principal	Yes	None
1	1	1230384	Switzerland	,Principal,	Yes	None
2	2	1230394	United States	,,Underlying Principal	Yes	None
3	3	1230403	Greece	Agent,Principal,	Yes	None
4	4	1230404	United States	Agent,Principal,	Yes	None

In [33]:

```
# Create curser object
cur2 = conn2.cursor()
```

In [34]:

```
# Create curser object
cur3 = conn3.cursor()
```

The two databases have been setup so I can now use SQL queries to merge and filter these. I have done this to mirror the rational database setup this system has in prod.

In [35]:

```
cur2.execute('''SELECT ce_no,
                       risk,
                       review_date
                FROM risk
                WHERE risk = 'Higher' ''')
risk_df = pd.DataFrame(cur2.fetchall())
risk_df.columns = [x[0] for x in cur2.description]
risk_df
```

Out[35]:

	ce_no	risk	review_date
0	1230538	Higher	2019-12-20 00:00:00
1	1230663	Higher	2020-09-23 00:00:00
2	1230809	Higher	2018-10-27 00:00:00
3	1230927	Higher	2019-12-12 00:00:00
4	1230953	Higher	2019-02-21 00:00:00
...	...	...	...
1059	9977114	Higher	2020-04-22 00:00:00
1060	9980336	Higher	2020-07-19 00:00:00
1061	9987295	Higher	2021-10-01 00:00:00
1062	9987296	Higher	2021-10-01 00:00:00
1063	9989742	Higher	2019-01-18 00:00:00

1064 rows × 3 columns

All 'Higher' AML risk clients have been succesfully pulled from 'risk' table

In [36]:

```
cur3.execute('''SELECT ce_no,
                       ppob
                       FROM ppob
                WHERE ppob = 'United Kingdom' AND kyc_vetting = 'Yes'
                ORDER BY ce_no ASC; ''')
ppob_df = pd.DataFrame(cur3.fetchall())
ppob_df.columns = [x[0] for x in cur3.description]
ppob_df
```

Out[36]:

	ce_no	ppob
0	1230451	United Kingdom
1	1230460	United Kingdom
2	1230468	United Kingdom
3	1230472	United Kingdom
4	1230518	United Kingdom
...	...	...
2482	9978662	United Kingdom
2483	9982133	United Kingdom
2484	9991829	United Kingdom
2485	9992824	United Kingdom
2486	9992996	United Kingdom

2487 rows × 2 columns

I can now merge these DataFrames to create risk profiles based on country.

In [37]:

```
uk_higher = pd.merge(risk_df, ppob_df, on='ce_no', how='outer')
```

In [38]:

```
uk_higher.head(5)
```

Out[38]:

	ce_no	risk	review_date	ppob
0	1230538	Higher	2019-12-20 00:00:00	NaN
1	1230663	Higher	2020-09-23 00:00:00	NaN
2	1230809	Higher	2018-10-27 00:00:00	United Kingdom
3	1230927	Higher	2019-12-12 00:00:00	NaN
4	1230953	Higher	2019-02-21 00:00:00	NaN

I can see some of cases have pulled back without a PPoB. These were from the 'risk' table. I am going to drop these rows as they are not related to my issue in question.

In [39]:

```
uk_higher = uk_higher[uk_higher['ppob'].notna()]
```

In [40]:

```
uk_higher = uk_higher.sort_values(by=['review_date'], ascending=True)
```

In [41]:

```
uk_higher.head(5)
```

Out[41]:

	ce_no	risk	review_date	ppob
770	8231495	Higher	2017-12-11 00:00:00	United Kingdom
508	7159027	Higher	2017-12-15 00:00:00	United Kingdom
555	7324276	Higher	2018-01-04 00:00:00	United Kingdom
112	1478639	Higher	2018-03-31 00:00:00	United Kingdom
125	1493163	Higher	2018-06-19 00:00:00	United Kingdom

Conclusion

SQL can be used by teams to pull data from the required information directly from database tables required to create risk reports based on risk rating, principle place of business. This is quicker and more efficient process than having to get data teams to pull this information. This can be sent to the correct KYC team as notification of what entities need reviewing first.

There a number of business opportunities which can be applied. All client accounts in the accounts system 'MasterFiles' have a legal entity identifier so if read-only access was granted for that you can use the 'ce_no' as a foreign key to pull back all the client's active and inactive accounts for analysis.

Next Steps

My next steps for this would be looking at this on a wider scale. The business tends to view risk profile by region

- AMER (The Americas)
- EMEA (Europe, Middle East and Africa)
- APAC (Asia-Pacific).

I would enhance the data for that region each country belongs in order to create individual reports for each reason.

I could also create visualisation based risk rating and review date as a high level overview of the current KYC situation. Given more opportunities I would overlay this information on a map of the world colour coded as an effective way to see which countries have the most clients which are out of KYC review.

There would be further opportunities if I had access to production data as I would be able to pull different data characteristics such as if they entities are in the offboarding queue or if they are linked to an Politically Exposed Person (PEP).