Project Title: Data Cleaning, Integration, and Database Management

Problem Statement

""" We are tasked with cleaning, merging, and normalizing multiple datasets obtained from various sources. The goal is to create a relational database to store the integrated data and enable efficient querying and analysis. """

Data Description

""" The datasets consist of information about Australian Business Numbers (ABN), Fair Jobs Code Pre-Assessment Certificates, Motor Dealer Organisation License Lists, Australian Financial Services Licensees, and School details. Each dataset contains different attributes such as ABN numbers, certificate numbers, license numbers, etc. """

Key Actions Taken:

- Conducted exploratory data analysis (EDA) to understand data distributions and identify anomalies.
- Cleaned datasets by addressing missing values, duplicates, and inconsistencies.
- Merged datasets using common keys like ABN and license numbers to create a unified dataset, then normalized it to enhance data integrity and reduce redundancy.

Cleaning Process for df_step1 (ABN Data)

```
import re
import sys
import os
from os import listdir
from os.path import isfile, join
```

```
# Define the folder path where the XML files are uploaded in Google Colab
folder = "/content/drive/My Drive/DB_Engineer/"  # Update as per your folder structure

# limits for the number of files and records to process.
filecount = 999
recordcount = 99999999
```

Function readData() reads XML files from the specified folder, extracts data using the extractLine() function, and writes the extracted data to CSV files. It iterates through each file, extracts information line by line, and writes it to an output CSV file. The function also prints progress messages indicating the processing status of each file.

```
def extractLine(line):

    recordLastUpdated = ""
    recordReplaced = ""
    abnStatus = ""
    abnStatusDate = ""
    abnNumber = ""
    entityTypeIND = ""
    entityTypeText = ""
    nameType = ""
    nameText = ""
    state = ""
    postcode = ""
    asicType = ""
    asicNumber = ""
    gstStatus = ""
    gstStatusDate = ""
    dgrStatusDate = ""
    dgrNameType = ""
    dgrNameText = ""
    oeEntry = ""

    # Regular expressions to extract data from each line of the XML file
    p = re.compile('<ABR recordLastUpdatedDate="(\d{8})" replaced="(\w)">')
    m = p.search(line)
    if (m == None): return
    recordLastUpdated = m.groups()[0]
```

```
recordReplaced = m.groups()[1]

# Extract ABN related information
p = re.compile('<ABN status="(\w{3})" ABNStatusFromDate="(\d{8})">(\d{0,20})')
m = p.search(line)
abnStatus = m.groups()[0]
abnStatusDate = m.groups()[1]
abnNumber = m.groups()[2]

# Extract entity type information
p = re.compile('<EntityTypeInd>(\w{0,4})')
m = p.search(line)
entityTypeIND = m.groups()[0]

# Extract entity type text information
p = re.compile('<EntityTypeText>((\w|\s){0,100})')
m = p.search(line)
entityTypeText = m.groups()[0]

# Extract name type and text information
p = re.compile('<MainEntity>(.*?)</MainEntity>')
m = p.search(line)
if (m != None):
    mainEntity = m.groups()[0]
    entity = mainEntity
    p = re.compile('<NonIndividualName type="(\w{0,3})')
    m = p.search(mainEntity)
    nameType = m.groups()[0]
    p = re.compile('<NonIndividualNameText>(.{0,})</NonIndividualNameText>')
    m = p.search(mainEntity)
    nameText = m.groups()[0]
else:
    p = re.compile('<LegalEntity>(.*?)</LegalEntity>')
    m = p.search(line)
    legalEntity = m.groups()[0]
    entity = legalEntity
    p = re.compile('<IndividualName type="(\w{0,99})')
    m = p.search(legalEntity)
    nameType = m.groups()[0]
    p = re.compile('<NameTitle>([\w|\s]{0,50})')
    m = p.search(legalEntity)
    nameTitle = ""
```

```
    if (m != None): nameTitle = m.groups()[0]
    p = re.compile('<GivenName>([\w|\s]{0,100})')
    m = p.search(legalEntity)
    givenNames = ""
    if (m != None): givenNames = " ".join(m.groups())
    p = re.compile('<FamilyName>([\w|\s]{0,50})')
    m = p.search(legalEntity)
    familyNames = ""
    if (m != None): familyNames = " ".join(m.groups())
    nameText = ""
    if (nameTitle): nameText = nameText + nameTitle
    if (givenNames): nameText = nameText + " " + givenNames
    if (familyNames): nameText = nameText + " " + familyNames

# Extract state and postcode information
p = re.compile('<State>(\w{0,3})')
m = p.search(entity)
state = m.groups()[0]
p = re.compile('<Postcode>(\d{0,50})')
m = p.search(entity)
postcode = m.groups()[0]

# Extract ASIC number information
p = re.compile('<ASICNumber ASICNumberType="(\w{0,99})">(\d{9})')
m = p.search(line)
if (m != None):
    asicType = m.groups()[0]
    asicNumber = m.groups()[1]

# Extract GST related information
p = re.compile('<GST status="(\w{0,99})" GSTStatusFromDate="(\d{8})"')
m = p.search(line)
gstStatus = m.groups()[0]
gstStatusDate = m.groups()[1]

# Extract DGR related information
p = re.compile('<DGR DGRStatusFromDate="(\d{8})?">(.*?)</DGR>')
m = p.search(line)
if (m != None):
    dgrStatusDate = m.groups()[0]
    dgr = m.groups()[1]
    if (dgr != None):
```

```
        if (dgr != None):
            p = re.compile('<NonIndividualName type="(\w{0,3})')
            m = p.search(dgr)
            dgrNameType = m.groups()[0]

            p = re.compile('<NonIndividualNameText>(.{0,})</NonIndividualNameText>')
            m = p.search(dgr)
            dgrNameText = m.groups()[0]

    # Extract Other Entity related information
    p = re.compile('<OtherEntity>.*?</OtherEntity>')
    m = p.findall(line)
    oeEntry = ""
    if m:
        otherEntities = m
        for oEntity in otherEntities:
            p = re.compile('<NonIndividualName type="(\w{0,3})')
            m = p.search(oEntity)
            oeNameType = m.groups()[0]
            p = re.compile('<NonIndividualNameText>(.{0,})</NonIndividualNameText>')
            m = p.search(oEntity)
            oeNameText = m.groups()[0]
            oeEntry = oeEntry + str(otherEntities.index(oEntity)+1) + ". " + oeNameType + "-" + oeNameText + " "

    # Prepare CSV line with extracted information
    nameText = '"' + nameText.replace('"','""').strip() + '"'
    entityTypeText = '"' + entityTypeText.replace('"','""').strip() + '"'
    dgrNameText = '"' + dgrNameText.replace('"','""').strip() + '"'
    oeEntry = '"' + oeEntry.replace('"','""').strip() + '"'
    fields = [recordLastUpdated, recordReplaced,
                    abnStatus, abnStatusDate, abnNumber,
                    entityTypeIND, entityTypeText,
                    nameType, nameText, state, postcode,
                    asicType, asicNumber,
                    gstStatus, gstStatusDate,
                    dgrNameType, dgrNameText, dgrStatusDate,
                    oeEntry]
    csvLine = ','.join(fields)
    return csvLine


from google.colab import drive
```

```
drive.mount('/content/drive')
```

Start coding or generate with AI.

Function readData() reads XML files from the specified folder, extracts data using the extractLine() function, and writes the extracted data to CSV files. It iterates through each file, extracts information line by line, and writes it to an output CSV file. The function also prints progress messages indicating the processing status of each file.

```
def readData():
    dataFiles = sorted([f for f in listdir(folder) if isfile(join(folder, f))])
    fcount = 0
    print('\nProcessing files in folder "' + folder + '"\n')
    for dataFile in dataFiles:
        if (fcount == filecount): break
        outFile = open(dataFile + '-output.csv', 'w')
        outFile.write('recordLastUpdated,recordReplaced,abnStatus,abnStatusDate,abnNumber,entityTypeIND,entityTypeText,nameType,nameText,stat

        lcount = 0
        rcount = 0
        with open(join(folder, dataFile)) as fileobject:
            print("File " + str(fcount+1) + ". " + dataFile + '')
            totalRecords = ""
            for line in fileobject:
                if (rcount == recordcount): break
                if (lcount == 0):
                    p = re.compile('<RecordCount>(.*?)</RecordCount>')
                    m = p.search(line)
                    totalRecords = m.groups()[0]
                if (recordcount == 99999999 or recordCount >= totalRecords):
                    targetRecords = totalRecords
                else: targetRecords = str(recordcount)

                csvLine = extractLine(line)
                if (csvLine):
                    msg = " Entry  " + str(rcount+1) + " of " + targetRecords
                    sys.stdout.write(('\b' * len(msg)) + msg)
                    outFile.write(csvLine + '\n')
                    rcount = rcount + 1
                lcount = lcount + 1
        print('  ==> ' + dataFile + '-output.csv')
```

```
                print(                + dataFile + "output.csv")
                sys.stdout.write("\033[F")
            print("\r\n")
        fcount = fcount + 1
        outFile.close()

# Execute the script
readData()



# Import necessary libraries
from google.colab import drive
import os
import pandas as pd

# Mount Google Drive

# Path to the folder containing CSV files in Google Drive
folder_path = '/content/drive/My Drive/DB_ENGINEER/DB_step1_op'

# Get a list of all CSV files in the folder
csv_files = [file for file in os.listdir(folder_path) if file.endswith('.csv')]

# Read each CSV file and concatenate them into a single DataFrame
dfs = []
for file in csv_files:
    file_path = os.path.join(folder_path, file)
    df = pd.read_csv(file_path)
    dfs.append(df)

# Concatenate all DataFrames into one
combined_df = pd.concat(dfs, ignore_index=True)

# Path to save the combined CSV file
combined_csv_path = '/content/drive/My Drive/DB_ENGINEER/New_step_op.csv'

# Write the combined DataFrame to a new CSV file
combined_df.to_csv(combined_csv_path, index=False)

# Print confirmation message
print("CSV files have been merged and saved as", combined_csv_path)
```

```
import pandas as pd
df_step1=pd.read_csv("/content/drive/My Drive/DB_ENGINEER/New_step_op.csv")

df_step1
```

```
Start coding or generate with AI.
```

Cleaning process for step2(Fair Jobs Code Pre-Assessment Certificates)

```
df_step2=pd.read_csv("/content/drive/My Drive/DB_ENGINEER/FJCPublicRegister_Certificate_FullExport_05-04-24.csv")

inner_join_df = pd.merge(df_step1, df_step2, left_on='abnNumber', right_on=' ABN', how='inner')


print(inner_join_df)


null_values = df_step2.isnull().sum()
print(null_values)
```

```
CertificateNumber      0
EntityName             0
 TradeName           391
 ABN                   0
 Status                0
 IssueDate             0
 ExpiryDate            0
dtype: int64
```

```
df_step2[' ABN'].nunique()
```

```
4083
```

```
abn_duplicates = df_step2[df_step2[' ABN'].duplicated()]
abn_duplicates.head()
```

```
# Filter rows where the value in the 'ABN' column is equal to '57618871266'
filtered_rows = df_step2[df_step2[' ABN'] == 18068707308]
filtered_rows.head()
```

	CertificateNumber	EntityName	TradeName	ABN	Status	IssueDate	ExpiryDate
1083	FJC-230110-1099	HATZ PTY LTD	Hatz Pty Ltd	18068707308	Issued	10/01/2023	09/01/2025
1085	FJC-230110-1101	HATZ PTY LTD	Hatz Pty Ltd	18068707308	Issued	10/01/2023	09/01/2025

```
# Iterate over each column in the DataFrame
for column in df_step2.columns:
    # Count the number of unique values in the column
    unique_count = df_step2[column].nunique()

    # Calculate the ratio of unique values to total values in the column
    uniqueness_ratio = unique_count / len(df_step2)

    # Print the column name and its uniqueness ratio
    print(f"Column '{column}' has {unique_count} unique values, with a uniqueness ratio of {uniqueness_ratio:.2f}")
```

```
Column 'CertificateNumber' has 4091 unique values, with a uniqueness ratio of 1.00
Column 'EntityName' has 4083 unique values, with a uniqueness ratio of 1.00
Column ' TradeName' has 3683 unique values, with a uniqueness ratio of 0.90
Column ' ABN' has 4083 unique values, with a uniqueness ratio of 1.00
Column ' Status' has 1 unique values, with a uniqueness ratio of 0.00
Column ' IssueDate' has 297 unique values, with a uniqueness ratio of 0.07
Column ' ExpiryDate' has 296 unique values, with a uniqueness ratio of 0.07
```

- With a uniqueness ratio of 1.00, each CertificateNumber is unique within the dataset.
- This suggests that **CertificateNumber** could be a suitable candidate for a primary key in the dataset.
- Additionally, in Step 1, we joined the datasets using the **ABN** as the foreign key.

```
Start coding or generate with AI.
```

Cleaning process for step3(Motor Dealer Organisation License Lists)

```
df_step3=pd.read_excel("/content/drive/My Drive/DB_ENGINEER/motor_organisation_license_step3.xlsx")


null_values = df_step3.isnull().sum()
print(null_values)
```

```
     Licence Number       0
     Issue Date           0
     Expiry Date          0
     Licensee             0
     Address Type         9
     Address              9
     Birth Year        2404
     ACN                  0
     ABN               1443
     Classes              0
     dtype: int64
```

```
abn_duplicates = df_step3[df_step3['ACN'].duplicated()]
abn_duplicates


df_sel=df_step3[df_step3["ACN"]=='001 736 974']


df_ste13=pd.merge(df_step1,df_step3,left_on='abnNumber',right_on='ABN',how='inner')


# Iterate over each column in the DataFrame
for column in df_step3.columns:
    # Count the number of unique values in the column
    unique_count = df_step3[column].nunique()

    # Calculate the ratio of unique values to total values in the column
    uniqueness_ratio = unique_count / len(df_step3)

    # Print the column name and its uniqueness ratio
    print(f"Column '{column}' has {unique_count} unique values, with a uniqueness ratio of {uniqueness_ratio:.2f}")
```

```
     Column 'Licence Number' has 2404 unique values, with a uniqueness ratio of 1.00
     Column 'Issue Date' has 906 unique values, with a uniqueness ratio of 0.38
     Column 'Expiry Date' has 1155 unique values, with a uniqueness ratio of 0.48
     Column 'Licensee' has 2377 unique values, with a uniqueness ratio of 0.99
```

```
Column 'Address Type' has 1 unique values, with a uniqueness ratio of 0.00
Column 'Address' has 2340 unique values, with a uniqueness ratio of 0.97
Column 'Birth Year' has 0 unique values, with a uniqueness ratio of 0.00
Column 'ACN' has 2376 unique values, with a uniqueness ratio of 0.99
Column 'ABN' has 951 unique values, with a uniqueness ratio of 0.40
Column 'Classes' has 1 unique values, with a uniqueness ratio of 0.00
```

```
Start coding or generate with AI.
```

#

- With a uniqueness ratio of 1.00, each **Licence Number** is unique within the dataset.
- This suggests that Licence Number could be a suitable candidate for a primary key in the dataset.
- Additionally, in Step 1, we joined the datasets using the **ABN** as the foreign key.

Cleaning process for step4(Australian Financial Services Licensees and Advisers_List)

```
df_step_41 = pd.read_csv("/content/drive/My Drive/DB_ENGINEER/afs_lic_step_4.csv")
```

```
df_step_42=pd.read_excel("/content/drive/My Drive/DB_ENGINEER/financial_advisers_202404_step6.xlsx")
```

```
df_step_42=df_step_42.iloc[:,0:26]
```

```
df_step_42 = df_step_42[df_step_42["ADV_ROLE_STATUS"] == 'Current']
```

```
df_step_42=df_step_42.drop_duplicates()
```

```
# Iterate over each column in the DataFrame
for column in df_step_41.columns:
    # Count the number of unique values in the column
    unique_count = df_step_41[column].nunique()
```

```
    # Calculate the ratio of unique values to total values in the column
    uniqueness_ratio = unique_count / len(df_step_41)

    # Print the column name and its uniqueness ratio
    print(f"Column '{column}' has {unique_count} unique values, with a uniqueness ratio of {uniqueness_ratio:.2f}")
```

```
 Column 'REGISTER_NAME' has 1 unique values, with a uniqueness ratio of 0.00
 Column 'AFS_LIC_NUM' has 6402 unique values, with a uniqueness ratio of 1.00
 Column 'AFS_LIC_NAME' has 6402 unique values, with a uniqueness ratio of 1.00
 Column 'AFS_LIC_ABN_ACN' has 6402 unique values, with a uniqueness ratio of 1.00
 Column 'AFS_LIC_START_DT' has 3222 unique values, with a uniqueness ratio of 0.50
 Column 'AFS_LIC_PRE_FSR' has 1093 unique values, with a uniqueness ratio of 0.17
 Column 'AFS_LIC_ADD_LOCAL' has 969 unique values, with a uniqueness ratio of 0.15
 Column 'AFS_LIC_ADD_STATE' has 8 unique values, with a uniqueness ratio of 0.00
 Column 'AFS_LIC_ADD_PCODE' has 719 unique values, with a uniqueness ratio of 0.11
 Column 'AFS_LIC_ADD_COUNTRY' has 2 unique values, with a uniqueness ratio of 0.00
 Column 'AFS_LIC_LAT' has 995 unique values, with a uniqueness ratio of 0.16
 Column 'AFS_LIC_LNG' has 995 unique values, with a uniqueness ratio of 0.16
 Column 'AFS_LIC_CONDITION' has 3512 unique values, with a uniqueness ratio of 0.55
```

```
# Iterate over each column in the DataFrame
for column in df_step_42.columns:
    # Count the number of unique values in the column
    unique_count = df_step_42[column].nunique()

    # Calculate the ratio of unique values to total values in the column
    uniqueness_ratio = unique_count / len(df_step_42)

    # Print the column name and its uniqueness ratio
    print(f"Column '{column}' has {unique_count} unique values, with a uniqueness ratio of {uniqueness_ratio:.2f}")
```

```
 Column 'REGISTER_NAME' has 1 unique values, with a uniqueness ratio of 0.00
 Column 'ADV_NAME' has 15586 unique values, with a uniqueness ratio of 0.98
 Column 'ADV_NUMBER' has 15626 unique values, with a uniqueness ratio of 0.98
 Column 'ADV_ROLE' has 1 unique values, with a uniqueness ratio of 0.00
 Column 'ADV_SUB_TYPE' has 3 unique values, with a uniqueness ratio of 0.00
 Column 'ADV_ROLE_STATUS' has 1 unique values, with a uniqueness ratio of 0.00
 Column 'ADV_ABN' has 636 unique values, with a uniqueness ratio of 0.04
 Column 'ADV_FIRST_PROVIDED_ADVICE' has 56 unique values, with a uniqueness ratio of 0.00
 Column 'LICENCE_NAME' has 1877 unique values, with a uniqueness ratio of 0.12
 Column 'LICENCE_NUMBER' has 1877 unique values, with a uniqueness ratio of 0.12
```

```
 Column 'LICENCE_ABN' has 1142 unique values, with a uniqueness ratio of 0.07
 Column 'LICENCE_CONTROLLED_BY' has 944 unique values, with a uniqueness ratio of 0.06
 Column 'ADV_START_DT' has 3805 unique values, with a uniqueness ratio of 0.24
 Column 'ADV_END_DT' has 0 unique values, with a uniqueness ratio of 0.00
 Column 'OVERALL_REGISTRATION_STATUS' has 2 unique values, with a uniqueness ratio of 0.00
 Column 'REGISTRATION_STATUS_UNDER_AFSL' has 1 unique values, with a uniqueness ratio of 0.00
 Column 'REGISTRATION_START_DATE_UNDER_AFSL' has 163 unique values, with a uniqueness ratio of 0.01
 Column 'REGISTRATION_END_DATE_UNDER_AFSL' has 0 unique values, with a uniqueness ratio of 0.00
 Column 'ADV_CPD_FAILURE_YEAR' has 8 unique values, with a uniqueness ratio of 0.00
 Column 'ADV_ADD_LOCAL' has 1924 unique values, with a uniqueness ratio of 0.12
 Column 'ADV_ADD_STATE' has 8 unique values, with a uniqueness ratio of 0.00
 Column 'ADV_ADD_PCODE' has 980 unique values, with a uniqueness ratio of 0.06
 Column 'ADV_ADD_COUNTRY' has 1 unique values, with a uniqueness ratio of 0.00
 Column 'REP_APPOINTED_BY' has 3426 unique values, with a uniqueness ratio of 0.21
 Column 'REP_APPOINTED_NUM' has 3426 unique values, with a uniqueness ratio of 0.21
 Column 'REP_APPOINTED_ABN' has 2438 unique values, with a uniqueness ratio of 0.15


#ADV_NUMB has high number unique vaalues which likely to take foriegn key


df_step4=pd.merge(df_step_42,df_step_41,left_on='LICENCE_NUMBER',right_on='AFS_LIC_NUM',how='left')


df_step4


df_step4 = df_step4.drop_duplicates(subset=['ADV_NUMBER'])


# Iterate over each column in the DataFrame
for column in df_step4.columns:
    # Count the number of unique values in the column
    unique_count = df_step4[column].nunique()

    # Calculate the ratio of unique values to total values in the column
    uniqueness_ratio = unique_count / len(df_step4)

    # Print the column name and its uniqueness ratio
    print(f"Column '{column}' has {unique_count} unique values, with a uniqueness ratio of {uniqueness_ratio:.2f}")


 Column 'REGISTER_NAME_x' has 1 unique values, with a uniqueness ratio of 0.00
 Column 'ADV_NAME' has 15586 unique values, with a uniqueness ratio of 1.00
 Column 'ADV_NUMBER' has 15626 unique values, with a uniqueness ratio of 1.00
 Column 'ADV_ROLE' has 1 unique values, with a uniqueness ratio of 0.00
```

```
Column 'ADV_SUB_TYPE' has 3 unique values, with a uniqueness ratio of 0.00
Column 'ADV_ROLE_STATUS' has 1 unique values, with a uniqueness ratio of 0.00
Column 'ADV_ABN' has 636 unique values, with a uniqueness ratio of 0.04
Column 'ADV_FIRST_PROVIDED_ADVICE' has 56 unique values, with a uniqueness ratio of 0.00
Column 'LICENCE_NAME' has 1848 unique values, with a uniqueness ratio of 0.12
Column 'LICENCE_NUMBER' has 1848 unique values, with a uniqueness ratio of 0.12
Column 'LICENCE_ABN' has 1119 unique values, with a uniqueness ratio of 0.07
Column 'LICENCE_CONTROLLED_BY' has 935 unique values, with a uniqueness ratio of 0.06
Column 'ADV_START_DT' has 3783 unique values, with a uniqueness ratio of 0.24
Column 'ADV_END_DT' has 0 unique values, with a uniqueness ratio of 0.00
Column 'OVERALL_REGISTRATION_STATUS' has 2 unique values, with a uniqueness ratio of 0.00
Column 'REGISTRATION_STATUS_UNDER_AFSL' has 1 unique values, with a uniqueness ratio of 0.00
Column 'REGISTRATION_START_DATE_UNDER_AFSL' has 163 unique values, with a uniqueness ratio of 0.01
Column 'REGISTRATION_END_DATE_UNDER_AFSL' has 0 unique values, with a uniqueness ratio of 0.00
Column 'ADV_CPD_FAILURE_YEAR' has 8 unique values, with a uniqueness ratio of 0.00
Column 'ADV_ADD_LOCAL' has 1914 unique values, with a uniqueness ratio of 0.12
Column 'ADV_ADD_STATE' has 8 unique values, with a uniqueness ratio of 0.00
Column 'ADV_ADD_PCODE' has 980 unique values, with a uniqueness ratio of 0.06
Column 'ADV_ADD_COUNTRY' has 1 unique values, with a uniqueness ratio of 0.00
Column 'REP_APPOINTED_BY' has 3355 unique values, with a uniqueness ratio of 0.21
Column 'REP_APPOINTED_NUM' has 3355 unique values, with a uniqueness ratio of 0.21
Column 'REP_APPOINTED_ABN' has 2390 unique values, with a uniqueness ratio of 0.15
Column 'REGISTER_NAME_y' has 1 unique values, with a uniqueness ratio of 0.00
Column 'AFS_LIC_NUM' has 1846 unique values, with a uniqueness ratio of 0.12
Column 'AFS_LIC_NAME' has 1846 unique values, with a uniqueness ratio of 0.12
Column 'AFS_LIC_ABN_ACN' has 1846 unique values, with a uniqueness ratio of 0.12
Column 'AFS_LIC_START_DT' has 1370 unique values, with a uniqueness ratio of 0.09
Column 'AFS_LIC_PRE_FSR' has 285 unique values, with a uniqueness ratio of 0.02
Column 'AFS_LIC_ADD_LOCAL' has 592 unique values, with a uniqueness ratio of 0.04
Column 'AFS_LIC_ADD_STATE' has 7 unique values, with a uniqueness ratio of 0.00
Column 'AFS_LIC_ADD_PCODE' has 481 unique values, with a uniqueness ratio of 0.03
Column 'AFS_LIC_ADD_COUNTRY' has 2 unique values, with a uniqueness ratio of 0.00
Column 'AFS_LIC_LAT' has 606 unique values, with a uniqueness ratio of 0.04
Column 'AFS_LIC_LNG' has 606 unique values, with a uniqueness ratio of 0.04
Column 'AFS_LIC_CONDITION' has 493 unique values, with a uniqueness ratio of 0.03


#we can remove with very less unique vlaues to remove redundency like AFS_LIC_ADD_COUNTRY,REGISTER_NAME_y,ADV_ADD_COUNTRY


#
```

- With a uniqueness ratio of 1.00, each **ADV_NUMBER** is unique within the dataset.
- This suggests that ADV_NUMBER could be a suitable candidate for a primary key in the dataset.
- Additionally, in Step 1, we joined the datasets using the **ADV_ABN** as the foreign key.

- Additionally, in Step 4, we joined the datasets using the **ADV_ABN** as the foreign key.

Start coding or generate with AI.

Cleaning process for step5(Victorian Government Schools details)

```
df_step5=pd.read_excel("/content/drive/My Drive/DB_ENGINEER/schoolABNdetails_step5.xlsx")
```

```
df_step5.columns
```

```
Index(['SchoolNumber', 'SchoolName', 'Fin_Period', 'LoadDate', 'ABN'], dtype='object')
```

```
df_step5.nunique()
```

```
SchoolNumber    1540
SchoolName      1540
Fin_Period         1
LoadDate        1540
ABN             1540
dtype: int64
```

```
#we can remove the column Fin_Period to remove redundancy
```

```
df_step1.dtypes
```

```
recordLastUpdated      int64
recordReplaced        object
abnStatus             object
abnStatusDate          int64
abnNumber              int64
entityTypeIND         object
entityTypeText        object
nameType              object
nameText              object
state                 object
postcode             float64
asicType              object
asicNumber           float64
gstStatus             object
gstStatusDate          int64
```

```
dgrNameType           object
dgrNameText           object
dgrStatusDate        float64
otherEntities         object
dtype: object
```

```
df_step5.dtypes
```

```
SchoolNumber             int64
SchoolName              object
Fin_Period               int64
LoadDate        datetime64[ns]
ABN                     object
dtype: object
```

```
# Remove non-numeric characters and spaces from 'ABN' column
df_step5['ABN'] = df_step5['ABN'].str.replace(r'\D', '', regex=True)

# Convert 'ABN' column to int64
df_step5['ABN'] = df_step5['ABN'].astype('int64')
```

```
df_step15=pd.merge(df_step1,df_step5,left_on='abnNumber',right_on='ABN',how='inner')
```

```
df_step15
```

```
df_step25=pd.merge(df_step2,df_step5,left_on=' ABN',right_on='ABN',how='inner')
```

```
df_step25
```

CertificateNumber	EntityName	TradeName	ABN	Status	IssueDate	ExpiryDate	SchoolNumber	SchoolName	Fin_Period	LoadDate	ABN

Start coding or generate with AI.

Final EDA - Integration and Normalisation(if required)

Final EDA, Integration and Normalisation(if required)

```
df_1 =df_step1
df_2 =df_step2
df_3 =df_step3
df_4 =df_step4
df_5 =df_step5

# EDA for each dataset
print("Step 1 EDA:")
print(df_1.info())
print(df_1.describe())

print("Step 2 EDA:")
print(df_2.info())
print(df_2.describe())

print("Step 3 EDA:")
print(df_3.info())
print(df_3.describe())

print("Step 4 EDA:")
print(df_4.info())
print(df_4.describe())

print("Step 5 EDA:")
print(df_5.info())
print(df_5.describe())
```

```
merged_df = pd.merge(df_1, df_2, left_on='abnNumber', right_on=' ABN', how='inner')
merged_df = pd.merge(merged_df, df_3, left_on='abnNumber', right_on='ABN', how='inner')
merged_df = pd.merge(merged_df, df_4, left_on='abnNumber', right_on='LICENCE_ABN', how='inner')
merged_df = pd.merge(merged_df, df_5, left_on='abnNumber', right_on='ABN', how='inner')

merged_df
```

recordLastUpdated	recordReplaced	abnStatus	abnStatusDate	abnNumber	entityTypeIND	entityTypeText	nameType	nameText	state	...	A

0 rows × 80 columns

```
Start coding or generate with AI.
```

##

- The absence of common fields across all datasets suggests that there are no direct relationships between them based on shared attributes.
- Upon individual examination, it's observed that each of the datasets (df_2, df_3, df_4, and df_5) exhibits relationships with df_1, albeit with minimal dependencies between them.
- Specifically, df_3 and df_4 possess their own primary keys, while LICENCE_NUMBER serves as a link between them, indicating a relationship between the two.
- Given the distinct relationships between df_1 and the other datasets, and the minimal interdependency among the latter, it's deemed appropriate to maintain all five tables separately.
- Consequently, the need for extensive normalization, as previously conducted, is mitigated due to the lack of significant redundancy among the tables.

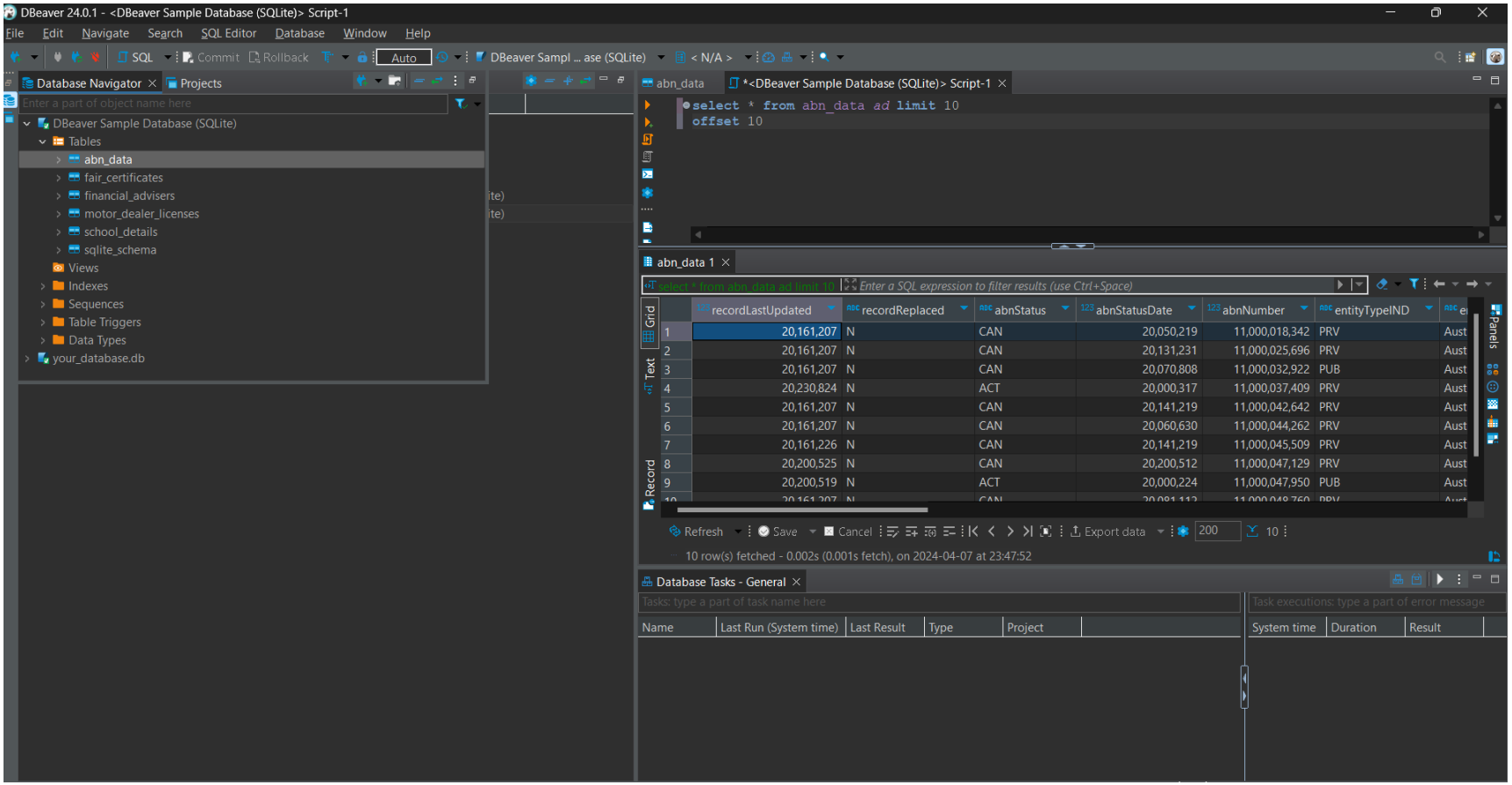
Fianl load into SQLite DataBase

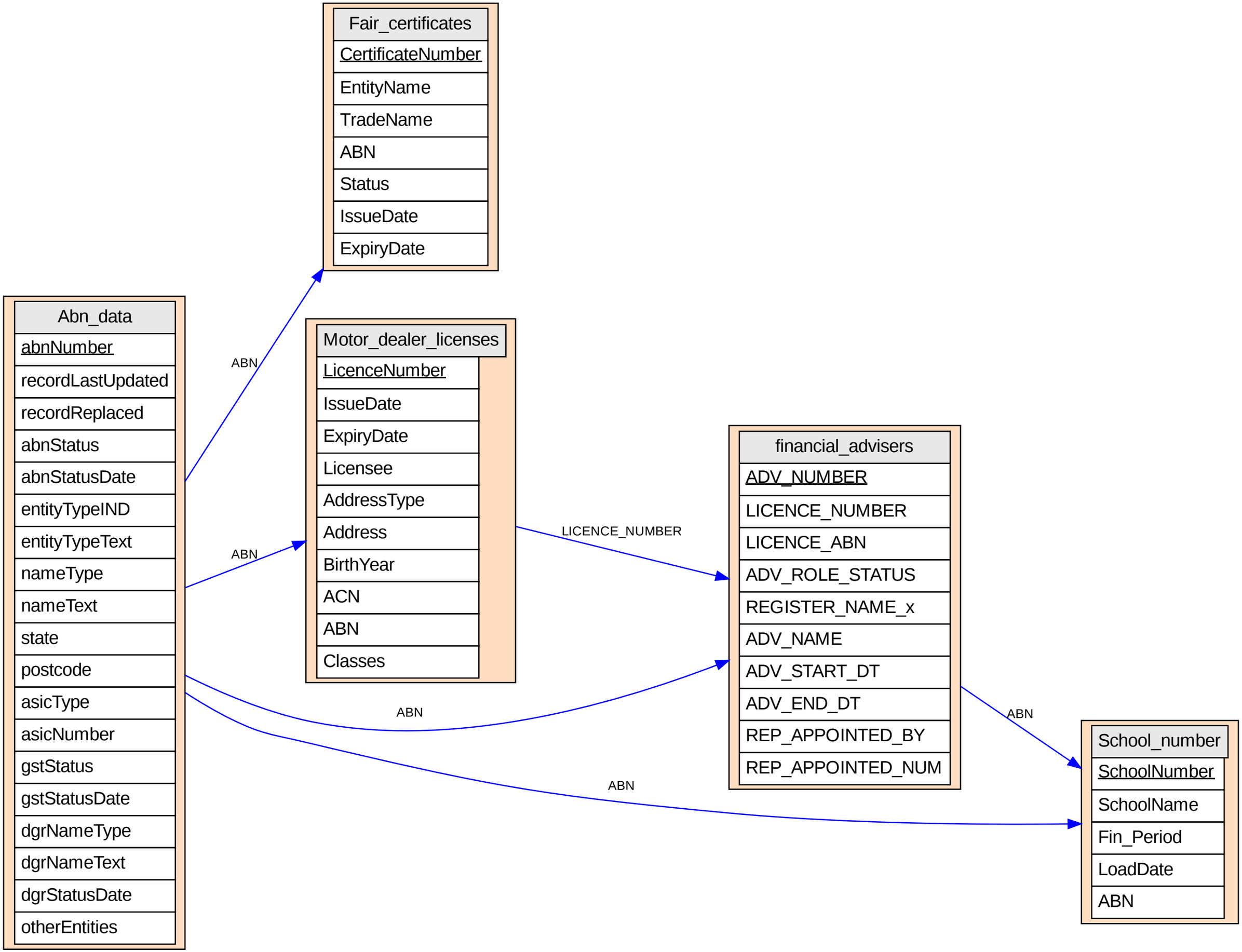
-later which we can connet to DBeaver to access them locally

```
import sqlite3

# Create a connection to the SQLite database
conn = sqlite3.connect('ABN_Database.db')

# Define the DataFrames with appropriate table names
tables = {
    'abn_data': df_1,
    'fair_certificates': df_2,
    'motor_dealer_licenses': df_3,
    'financial_advisers': df_4,
    'school_details': df_5
}
```

```
# Dump each DataFrame into the SQLite database with proper table names
for table_name, df in tables.items():
    df.to_sql(table_name, conn, if_exists='replace', index=False)

# Close the connection
conn.close()

print("Data has been successfully dumped into the database.")



import sqlite3

# Create a connection to the SQLite database
conn = sqlite3.connect('Abn_final_database.db')

# Enable foreign key constraints
conn.execute('PRAGMA foreign_keys = ON;')

# Define the DataFrames with appropriate table names
tables = {
    'abn_data': (df_1, 'abnNumber'),
    'fair_certificates': (df_2, 'CertificateNumber'),
    'motor_dealer_licenses': (df_3, 'Licence Number'),  # Commenting out df_3 for now
    'financial_advisers': (df_4, 'ADV_NUMBER'),
    'school_details': (df_5, 'SchoolNumber')
}

# Define foreign key relationships
foreign_keys = {
    'fair_certificates': ('ABN', 'abnNumber'),
    'motor_dealer_licenses': (' ABN',  'abnNumber'),  # Commenting out df_3 foreign key relationship for now
    'financial_advisers': ('LICENCE_ABN', 'abnNumber'),
    'school_details': ('ABN', 'abnNumber')
}

# Dump each DataFrame into the SQLite database with proper table names
for table_name, (df, primary_key) in tables.items():
    # Define primary key constraint
    primary_key_constraint = f'CONSTRAINT pk_{table_name} PRIMARY KEY ({primary_key})'
    # Add primary key constraint to the DataFrame
    df_with_pk = df.copy()
```

```
    df_with_pk = df_with_pk.set_index(primary_key)
    # Dump DataFrame into SQLite database
    df_with_pk.to_sql(table_name, conn, if_exists='replace', index=True)
    # Add primary key constraint to the table in the database
    conn.execute(f'CREATE UNIQUE INDEX {table_name}_idx ON {table_name} ({primary_key})')
    print(f"DataFrame dumped into SQLite table '{table_name}' successfully.")

# Define foreign key constraints
for child_table, (parent_column, parent_table, parent_primary_key) in foreign_keys.items():
    conn.execute(f'ALTER TABLE {child_table} ADD CONSTRAINT fk_{child_table}_{parent_table} '
                 f'FOREIGN KEY ({parent_column}) REFERENCES {parent_table}({parent_primary_key});')
    print(f"Foreign key constraint added for '{child_table}' referencing '{parent_table}'.")

# Close the connection
conn.close()

print("Data has been successfully dumped into the database.")
```





Entity Relationship Diagram (ERD)