"DATA MIGRATION"

Prepared by- **Pranav Arora**

This Report contains:

- Introduction to Data Migration
- Application of Data Migration, how & where it is used
- Integrating Python with SQL to perform Data Migration
- Performing Data Match or Data Comparison with business logic
- Performing Data comparison through real life example with business logic
- Performing Data Migration of a sample project with business logic
- Performing Level-1 Data Migration of a sample project comprising 3 tables with business logic

What is Data Migration?

- Data migration is the process of transferring data from one storage system or computing environment to another.
- The business driver is usually an application migration or consolidation in which legacy systems are replaced or augmented by new applications that will share the same dataset.

Reasons of Data Migration:

- Replacing servers or storage devices or consolidating or decommissioning data center
- Another main reason of implementing data migration is Cloud Computing.

7- Phases of Performing Data Migration:

- **Premigration planning-** Evaluate the data being moved for stability.
- **Project initiation-** Identify and brief key stakeholders.
- **Landscape analysis-** Establish a robust data quality rules management process and brief the business on the goals of the project, including shutting down legacy systems.
- **Solution design-** Determine what data to move, and the quality of that data before and after the move.
- **Build & test-** Code the migration logic and test the migration with a mirror of the production environment.
- **Execute & validate-** Demonstrate that the migration has complied with requirements and that the data moved is viable for business use.
- **Decommission & monitor-** Shut down and dispose of old systems.

Softwares used in performing Data Migration:

1. Python installed with all libraries
2. SQL Server installed

Integrating Python with SQL Server :

To integrate Python and SQL Server, we can use libraries such as **pyodbc** or **sqlalchemy**. These libraries provide the necessary functions to connect, query, and manipulate data in SQL Server from Python.

Connecting to SQL Server:

integrate.py > ...

```python
import pypyodbc as odbc # pip install pypyodbc

DRIVER_NAME = 'SQL SERVER'
SERVER_NAME = 'PRANAV\SQLEXPRESS'
DATABASE_NAME = 'master'

# uid=<username>;
# pwd=<password>;
connection_string = f"""
    DRIVER={{{DRIVER_NAME}}};
    SERVER={SERVER_NAME};
    DATABASE={DATABASE_NAME};
    Trust_Connection=yes;

"""
conn = odbc.connect(connection_string)
print(conn)
```

By running above python code after installing sql server and installing all required libraries such as pandas and pyodbc on command prompt window, we have to give our sql server name, our database name where we are going to save tables. By performing these steps, we complete the integration between python and SQL server.

Sample Task 1:- To compare two CSV files data comprising student database. One CSV file should be created on Excel and other using SQL Server.

Then using python code and libraries match the dataframe of both csv files.

Business Logic:

Step 1:- Create one sample student database on excel and save it as csv file format

Step 2:- Create same sample student database on SQL server using sqlplus commands. Export it to csv format.

Step 3:- Open both csv files in your code editor such as VsCode. Using python code and libraries, define the path of both csv files and then match the dataframes

Step 4:- We can change some of the data in either of csv files to print the unique row then.

CSV file created on Excel:

Name	Roll_No	Section	Age
Pranav	23	A	20
Aryan	27	C	19
Arnav	12	F	20
Yash	56	D	21
Varun	34	D	18

Creating Student table on SQL Server:

```sql
create table Stu_Data(Name varchar(20),Roll_No int,Section varchar(10),Age int);
insert into Stu_Data values('Pranav',23,'A',20);
insert into Stu_Data values('Aryan',27,'C',19);
insert into Stu_Data values('Arnav',12,'F',20);
insert into Stu_Data values('Yash',56,'D',21);
insert into Stu_Data values('Varun',34,'D',18);
select * from Stu_Data;
```

200 %

Results | Messages

	Name	Roll_No	Section	Age
1	Pranav	23	A	20
2	Aryan	27	C	19
3	Arnav	12	F	20
4	Yash	56	D	21
5	Varun	34	D	18

Opening both CSV files on VsCode:

C: > VIT > stu_data.csv

```csv
Name,Roll_No,Section,Age
Pranav,23,A,20
Aryan,27,C,19
Arnav,12,F,20
Yash,56,D,21
Varun,34,D,18
```

C: > VIT > stu_data1.csv

```csv
Name,Roll_No,Section,Age
Pranav,23,A,20
Aryan,27,C,19
Arnav,12,F,20
Yash,56,D,21
Varun,34,D,18
```

Python Code:

csv_import.py > ...

```python
import pandas as pd

df1=pd.read_csv(r'C:/VIT/stu_data1.csv')
df2 = pd.read_csv(r'C:/VIT/stu_data.csv')
print("Displaying CSV file created trough Excel:")
print()
print(df1)
print()
print("Displaying CSV file exported through SQL Server")
print()
print(df2)
print()

are_equal = df1.equals(df2)
print("Are the DataFrames equal?", are_equal)

# Rows unique to the first DataFrame
unique_to_df1 = df1[~df1.isin(df2)].dropna()

# Rows unique to the second DataFrame
unique_to_df2 = df2[~df2.isin(df1)].dropna()

print("Rows unique to df1:")
print(unique_to_df1)

print("Rows unique to df2:")
print(unique_to_df2)
```

Output: Same Dataframe

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\VIT\CODING\PYTHON> python -u "c:\VIT\CODING\PYTHON\csv_import.py"
Displaying CSV file created trough Excel:

     Name  Roll_No Section  Age
0  Pranav       23       A   20
1   Aryan       27       C   19
2   Arnav       12       F   20
3    Yash       56       D   21
4   Varun       34       D   18

Displaying CSV file exported through SQL Server

     Name  Roll_No Section  Age
0  Pranav       23       A   20
1   Aryan       27       C   19
2   Arnav       12       F   20
3    Yash       56       D   21
4   Varun       34       D   18

Are the DataFrames equal? True
Rows unique to df1:
Empty DataFrame
Columns: [Name, Roll_No, Section, Age]
Index: []
Rows unique to df2:
Empty DataFrame
Columns: [Name, Roll_No, Section, Age]
Index: []
```

Changing some csv data:

C: > VIT > stu_data1.csv

```
Name,Roll_No,Section,Age
Pranav,23,A,20
Aryan,27,C,19
Arnav,12,F,20
Yash,56,D,21
Varun,34,D,18
Dhruv,21,E,20
```

C: > VIT > stu_data.csv

```
Name,Roll_No,Section,Age
Pranav,23,A,20
Aryan,27,C,19
Arnav,12,F,20
Yash,56,D,21
Varun,34,D,18
```

Output: Different Dataframes, also printed the unique row

```
PS C:\VIT\CODING\PYTHON> python -u "c:\VIT\CODING\PYTHON\csv_import.py"
Displaying CSV file created trough Excel:


     Name  Roll_No Section  Age
0  Pranav       23       A   20
1   Aryan       27       C   19
2   Arnav       12       F   20
3    Yash       56       D   21
4   Varun       34       D   18
5   Dhruv       21       E   20

Displaying CSV file exported through SQL Server

     Name  Roll_No Section  Age
0  Pranav       23       A   20
1   Aryan       27       C   19
2   Arnav       12       F   20
3    Yash       56       D   21
4   Varun       34       D   18

Are the DataFrames equal? False
Rows unique to df1:
    Name  Roll_No Section   Age
5  Dhruv     21.0       E  20.0
Rows unique to df2:
Empty DataFrame
Columns: [Name, Roll_No, Section, Age]
Index: []
PS C:\VIT\CODING\PYTHON>
```

Sample Task 2: To compare two CSV files data, one comprising student database with the subject they learn and other CSV file comprising teacher's database with the section and subject they teach.

Then using python code and libraries match the dataframe of both csv files, find if any common subject and section is found between any of the student and teacher.

Business Logic:

Step 1:- Create one sample student database on excel and save it as csv file format

Step 2:- Create another CSV file of teacher's database.

Step 3:- Open both csv files in your code editor such as VsCode. Using python code and libraries, define the path of both csv files and then match the dataframes

Step 4:- We compare both the CSV files and then print the matched dataframe, of a student who learns same subject in same section of any of the teacher.

Opening both CSV files on VsCode:

C: > VIT > Student_database.csv

```
Name,Roll_No,Section,Age,Subject
Pranav,23,A,20,Maths
Aryan,27,C,19,Physics
Arnav,12,F,20,Maths
Yash,56,D,21,Sociology
Varun,34,B,18,Accounts
```

C: > VIT > Teachers_Databse.csv

```
T_Name,T_ID,Subject,Section,Gender
Shweta,2301,Maths,A,F
Mahesh,1405,Maths,B,M
Sanjeev,5401,Physics,E,M
Pooja,1100,English,D,F
Minakshi,3324,Chemistry,C,F
Akansha,2211,Phy Edu,F,F
Shweta,2301,Hindi,B,F
Shweta,2301,Hindi,A,F
```

Python Code:

```
teacher_student.py > ...
import pandas as pd

# Read the student and teacher CSV files into dataframes
student_df = pd.read_csv(r'C:/VIT/Student_database.csv')
teacher_df = pd.read_csv(r'C:/VIT/Teachers_Databse.csv')
print(student_df)
print()
print(teacher_df)
print()

# Merge the dataframes on the 'Section' column
merged_df = student_df.merge(teacher_df, left_on=['Section', 'Subject'], right_on=['Section', 'Subject'], how='inner')
combined=pd.merge(student_df,teacher_df)
print(combined)
print()
# Select the relevant columns for output
for index, row in merged_df.iterrows():
    student_name = row['Name']
    teacher_name = row['T_Name']
    section = row['Section']
    subject = row['Subject']
    print(f"{student_name} learns under {teacher_name}  with section {section} and subject {subject}")
```

Output: Student and teacher having common subject and section is printed

```
PS C:\VIT\CODING\PYTHON> python -u "c:\VIT\CODING\PYTHON\teacher_student.py"
     Name  Roll_No Section  Age    Subject
0  Pranav       23       A   20      Maths
1   Aryan       27       C   19    Physics
2   Arnav       12       F   20      Maths
3    Yash       56       D   21  Sociology
4   Varun       34       B   18   Accounts

     T_Name  T_ID    Subject Section Gender
0    Shweta  2301      Maths       A      F
1    Mahesh  1405      Maths       B      M
2   Sanjeev  5401    Physics       E      M
3     Pooja  1100    English       D      F
4  Minakshi  3324  Chemistry       C      F
5   Akansha  2211    Phy Edu       F      F
6    Shweta  2301      Hindi       B      F
7    Shweta  2301      Hindi       A      F

     Name  Roll_No Section  Age Subject  T_Name  T_ID Gender
0  Pranav       23       A   20   Maths  Shweta  2301      F

Pranav learns under Shweta  with section A and subject Maths
PS C:\VIT\CODING\PYTHON>
```

Sample Task 3:- "Data Migration Task"

Using the above applications of comparing data between csv files, now we need to migrate data in a company's sample file.

You have been given with one sample csv file in which you have carrier code, business ro entity and network id as the attributes.

If length of business ro entity is 2 then you have to migrate data to resp_org_entity table(compare it's pk with carrier code), if it matches then append bussiness ro entity and network id in resp_org_entity table

If length of business ro entity is 5 then you have to migrate data to resp_org_unit table(compare it's pk with carrier code), if it matches then append bussiness ro entity and network id in resp_org_unit table

*Both resp_org_entity and resp_org_unit table should be created on SQL server, then using integration between python and SQL, we have to migrate sample csv file given data to SQL tables.

Business Logic:

Step 1:- One sample CSV file comprising carrier code, Business RO entity, and network ID is given to us

Step 2:- Create two tables, Resp_org_Entity and Resp_org_Unit on SQL Server. Add one primary key in both the tables, along with Business_RO_Entity and Net_ID attributes containing 'NULL' values

Step 3:- Now, develop a python code that connects python and SQL server, by taking server name, database name of your SQL tables- resp_org_entity and resp_org_unit.

Step 4:- Then, provide the SQL query, to first compare the CSV file and table data wherein, all matched PK's value of table with carrier code in CSV file are executed

Step 5:- Then, among those matched values filter out the one in which business ro entity length in csv file is of 2, to append in resp_org_entity table and the one with length of 5, is appended in resp_org_unit table

Sample CSV file:

C: > VIT > carrier.csv

```
Carrier  Code,Business RO Entity,Network ID
445,ERI,12345
678,NOKIA,67890
231,HU,23456
110,CISCO,78901
998,ZTE,34567
342,ALCATEL,89012
```

SQL tables of resp_org_entity:

a) Before appending csv file required data:

	EMP_ID	EMP_NAME	Business RO Entity	Network ID
1	231	ARYAN	NULL	NULL
2	342	ANSH	NULL	NULL
3	567	VANSH	NULL	NULL
4	676	NIHAL	NULL	NULL
5	778	DHRUV	NULL	NULL
6	876	PRANAV	NULL	NULL

b) After appending csv file required data:

Results | Messages

	EMP_ID	EMP_NAME	Business RO Entity	Network ID
1	231	ARYAN	HU	23456

Python Code:

org_Entity.py > ...

```python
import pandas as pd
import pyodbc

# Define the database connection parameters
server = 'PRANAV\SQLEXPRESS'
database = 'master'
driver = 'SQL SERVER'

# Establish a database connection
conn = pyodbc.connect(f'DRIVER={{{driver}}};SERVER={server};DATABASE={database};')

# Define the path to your CSV file
csv_file = 'C:/VIT/carrier.csv'

# Read the CSV file into a pandas DataFrame
df = pd.read_csv(csv_file)

# Iterate through the rows of the DataFrame and update the SQL Server table
cursor = conn.cursor()

for index, row in df.iterrows():
    carrier_code = row['Carrier  Code']
    business_ro_entity = row['Business RO Entity']
    network_id = row['Network ID']

    # Update the SQL Server table where EMP_ID matches the Carrier Code
    update_query = """
            UPDATE RESP_ORG_ENTY
            SET [Business RO Entity] = ?,
                [Network ID] = ?
            WHERE EMP_ID = ?
            """

    cursor.execute(update_query, (business_ro_entity, network_id, carrier_code))

# Commit the changes
conn.commit()


# Delete rows with NULL values in Business RO Entity and Network ID
delete_query = """
DELETE FROM RESP_ORG_ENTY
WHERE [Business RO Entity] IS NULL AND [Network ID] IS NULL OR LEN(ISNULL([Business RO Entity], '')) <> 2;
"""

# Execute the DELETE statement
cursor.execute(delete_query)

# Commit the changes
conn.commit()

# Close the database connection
conn.close()

print("Update completed successfully.")
```

Output: Business RO Entity and Network ID from CSV file are appended to Resp_Org_Entity table, whose carrier code matched with EMP_ID(PK) and corresponding (BROE) length was 2

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\VIT\CODING\PYTHON> python -u "c:\VIT\CODING\PYTHON\org_Entity.py"
Update completed successfully.
PS C:\VIT\CODING\PYTHON>
```

SQL tables of resp_org_entity:

a) Before appending csv file required data:

Results | Messages

	EMP_ID	EMP_NAME	BUSINESS_RO_ENTITY	NETWORK_ID
1	445	PRANAV	NULL	NULL
2	678	VANSH	NULL	NULL
3	1091	ARYAN	NULL	NULL
4	2200	DHRUV	NULL	NULL
5	3719	NIHAL	NULL	NULL

b) After appending csv file required data:

Results | Messages

	EMP_ID	EMP_NAME	BUSINESS_RO_ENTITY	NETWORK_ID
1	678	VANSH	NOKIA	67890

Python Code:

org_UNIT.py > ...

```python
import pandas as pd
import pyodbc

# Define the database connection parameters
server = 'PRANAV\SQLEXPRESS'
database = 'master'
driver = 'SQL SERVER'

# Establish a database connection
conn = pyodbc.connect(f'DRIVER={{{driver}}};SERVER={server};DATABASE={database};')

# Define the path to your CSV file
csv_file = 'C:/VIT/carrier.csv'

# Read the CSV file into a pandas DataFrame
df = pd.read_csv(csv_file)

# Iterate through the rows of the DataFrame and update the SQL Server table
cursor = conn.cursor()

for index, row in df.iterrows():
    carrier_code = row['Carrier  Code']
    business_ro_entity = row['Business RO Entity']
    network_id = row['Network ID']


    # Update the SQL Server table where EMP_ID matches the Carrier Code
    update_query = f"""
    UPDATE RESP_ORG_UNIT
    SET [BUSINESS_RO_ENTITY] = ?,
        [NETWORK_ID] = ?
    WHERE EMP_ID = ?
    """

    cursor.execute(update_query, (business_ro_entity, network_id, carrier_code))

# Commit the changes
conn.commit()

# Delete rows with NULL values in BUSINESS_RO_ENTITY and NETWORK_ID
delete_query = """
DELETE FROM RESP_ORG_UNIT
WHERE [BUSINESS_RO_ENTITY] IS NULL AND [NETWORK_ID] IS NULL OR LEN(ISNULL([BUSINESS_RO_ENTITY], '')) <> 5;
"""

# Execute the DELETE statement
cursor.execute(delete_query)

# Commit the changes
conn.commit()

# Close the database connection
conn.close()

print("Update completed successfully.")
```

Output: Business RO Entity and Network ID from CSV file are appended to Resp_Org_Unit table, whose carrier code matched with EMP_ID(PK) and corresponding (BROE) length was 5

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

PS C:\VIT\CODING\PYTHON> python -u "c:\VIT\CODING\PYTHON\org_UNIT.py"
Update completed successfully.
PS C:\VIT\CODING\PYTHON>
```

Sample Task 4:- "Linking 3 tables"

Unlike previous task, this time we need not consider csv file. Instead, two tables created on SQL- Resp_org_entity and Resp_org_Unit, are to be linked with another third table(naming it as 'link') to be created on SQL itself. LINK table should also contain business_ro_entity and network_ID as it's attributes.

Then, using Python and it's libraries, we join the both tables one by one (Entity with link table and Unit with link table), then compare the data, if any business_ro_entity in org_entity table matches with business_ro_entity in link table, Print that business_ro_entity and corresponding network_id of LINK table.

Business Logic:

Step 1:- Already two tables are created on SQL server. Create another third table, naming it as LINK as it links with another 2 tables (resp_org_entity and resp_org_Unit)

Step 2:- Develop python code to connect these two tables with LINK table, by writing sqlquery of INNER JOIN, in the code.

Step 3:- The query, compares Business_RO_Entity of both joining tables, if it matches then it displays that RO_Entity with Net_ID of LINK table

a) LINK table data in SQL server

Results | Messages

	BUSINESS_RO_ENTITY	NETWORK_ID
1	NOKIA	860
2	VODAFONE	991
3	IDEA	84
4	JIO	332
5	CISCO	990
6	HU	860

b) Resp_Org_Entity table after data migration

Results | Messages

	EMP_ID	EMP_NAME	Business RO Entity	Network ID
1	231	ARYAN	HU	23456

c) Resp_Org_Unit table after data migration

Results | Messages

	EMP_ID	EMP_NAME	BUSINESS_RO_ENTITY	NETWORK_ID
1	678	VANSH	NOKIA	67890

Python Code:

link.py > ...

```python
import pyodbc

# Define your SQL Server connection parameters
server = 'PRANAV\SQLEXPRESS'
database = 'master'
# username = 'your_username'
# password = 'your_password'

# Create a connection to the SQL Server database
connection_string = f'DRIVER=SQL Server;SERVER={server};DATABASE={database};'
# UID={username};PWD={password}
connection = pyodbc.connect(connection_string)

# Create a cursor to execute SQL queries
cursor = connection.cursor()

try:
    # Execute the SQL query to link the two tables and retrieve matching records
    sql_query = """
    SELECT l.BUSINESS_RO_ENTITY, l.NETWORK_ID
    FROM LINK AS l
    INNER JOIN RESP_ORG_ENTY AS r ON r.[Business RO Entity] = l.[BUSINESS_RO_ENTITY]
    """
    cursor.execute(sql_query)
    rows = cursor.fetchall()

    # Print the results
    for row in rows:
        print(f"BUSINESS_RO_ENTITY: {row.BUSINESS_RO_ENTITY}, NETWORK_ID: {row.NETWORK_ID}")


    sql_query1 = """
    SELECT l.BUSINESS_RO_ENTITY, l.NETWORK_ID
    FROM LINK AS l
    INNER JOIN RESP_ORG_UNIT AS r ON r.[BUSINESS_RO_ENTITY] = l.[BUSINESS_RO_ENTITY]
    """

    # Execute the query and fetch the results

    cursor.execute(sql_query1)
    rows = cursor.fetchall()

    # Print the results
    for row in rows:
        print(f"BUSINESS_RO_ENTITY: {row.BUSINESS_RO_ENTITY}, NETWORK_ID: {row.NETWORK_ID}")


except Exception as e:
    print(f"An error occurred: {str(e)}")

finally:
    # Close the cursor and the database connection
    cursor.close()
    connection.close()
```

Output: Finally after data migration in task-3, if the data in left in either of Entity and Unit table matches to LINK table, then that is printed alongwith Network ID of LINK table

```
PS C:\VIT\CODING\PYTHON> python -u "c:\VIT\CODING\PYTHON\link.py"
BUSINESS_RO_ENTITY: HU, NETWORK_ID: 860
BUSINESS_RO_ENTITY: NOKIA, NETWORK_ID: 860
PS C:\VIT\CODING\PYTHON>
```

Conclusion:

→In this report, we have discussed the process of data migration using Python and Microsoft SQL Server. We covered the integration of Python with SQL Server, data extraction, transformation, and loading steps. This integration allows for efficient and automated data migration tasks, making it a valuable tool for various data-related projects.

→This report provides a foundation for understanding the integration of Python and SQL Server for data migration.

→References:

- https://github.com/mkleehammer/pyodbc
- https://pandas.pydata.org/docs/
- https://learn.microsoft.com/en-us/sql/sql-server/?view=sql-server-ver16

→In general, through the tasks related to data migration, We have gained valuable insights and practical experience in the field.

→In the course of these data migration tasks, a comprehensive understanding of the data migration process using Python and Microsoft SQL Server has been developed. The key takeaways from these tasks include:

1) **Data Integration**: We've learned how to integrate Python, a versatile programming language, with Microsoft SQL Server, a robust relational database management system. This integration enables efficient data manipulation and migration.
2) **Data Comparison**: These tasks have equipped us with the skills to compare data from various sources, such as CSV files, Excel, and SQL

tables, using Python. This is essential for identifying commonalities and discrepancies in datasets.

3) **Data Transformation**: We' ve gained experience in data transformation, where we can clean, structure, and prepare data for migration. Python's libraries, like Pandas, provide powerful tools for data manipulation.
4) **Data Loading**: The tasks have demonstrated how to load data into SQL Server tables from various sources, creating the necessary tables if needed. This is a crucial step in data migration and database management.
5) **SQL Queries**: We've worked with SQL queries to perform operations such as data insertion, joining tables, and filtering data, enhancing our SQL skills in the context of data migration.
6) **Practical Application**: These tasks simulate real-world scenarios where We've applied data migration techniques to scenarios involving students, teachers, and organizational data. This hands-on experience is invaluable for future data-related projects.

→Overall, the combination of Python's flexibility and SQL Server's reliability offers a powerful platform for data professionals to migrate, transform, and manage data effectively. The skills acquired through these tasks are transferrable and can be applied to a wide range of data migration and analysis projects in various domains.

THANK YOU