ADS507 - Design Document

Team 2: Aaron Carr, Lai Ieng Chan, Kyle Esteban Dalope

GitHub Link

GitHub Repo: https://github.com/amcarr-ds/ads507_data_engineering
See the README "Getting started" for instructions on how to clone the repo.

Source System Information

1. ISO / Country List Dataset
   - Source: https://www.kaggle.com/datasets/sevgisarac/temperature-change
   - Format: .csv
   - Dimensions: 7 columns x 321 instances
2. World Population Dataset
   - Source: https://www.kaggle.com/datasets/iamsouravbanerjee/world-population-dataset
   - Format: .csv
   - Dimensions: 17 columns x 234 instances
3. Temperature Change Dataset
   - Source: https://www.kaggle.com/datasets/sevgisarac/temperature-change
   - Format: .csv
   - Dimensions: 14 columns x 242,165 instances
4. Emissions by Country Dataset
   - Source: https://www.kaggle.com/datasets/thedevastator/global-fossil-co2-emissions-by-country-2002-2022
   - Format: .csv
   - Dimensions: 11 columns x 63,104 instances

Reasons for choosing the datasets: This project is intended to bring in three datasets in relation to human population and climate related factors (emissions and temperature) by country. Using the global surface temperature change data from the 'Temperature Change' dataset, the project aims to analyze the correlations between that and two other factors – CO_2 emissions from different fuel sources extracted in the 'Emissions by Country' dataset, and historical population from the 'World Population' dataset. Ultimately, these findings and insights can be used to develop a tracking system for temperature, emissions, and population change at a country-level so that policy recommendations can be given to the responsible agencies regarding carbon emission targets and their association to global temperature trends to predict whether the agreed upon goals of not exceeding the 1.5 degrees Celsius are met.

Architectural Diagram

1. RDBMS Schema Diagram

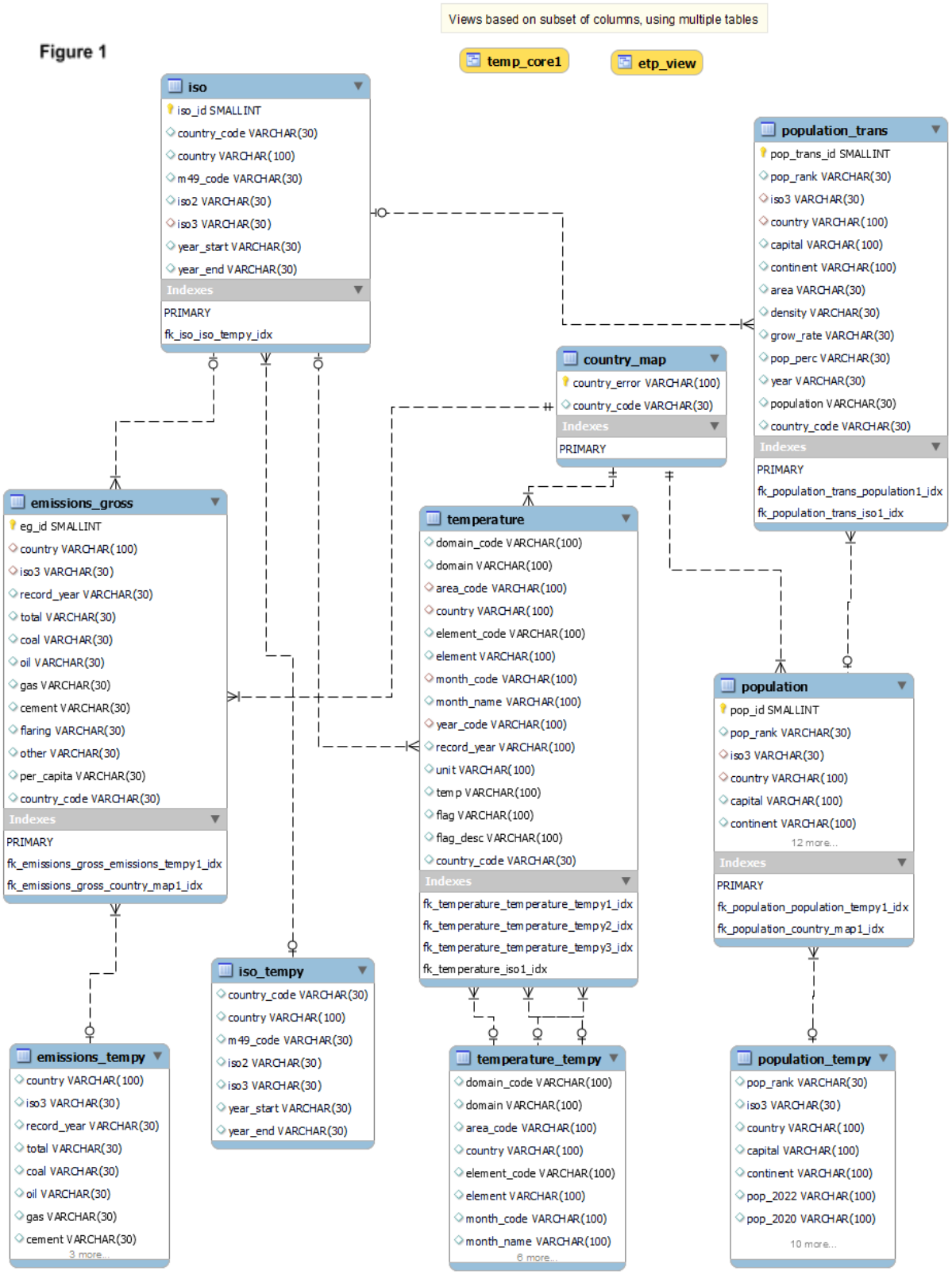
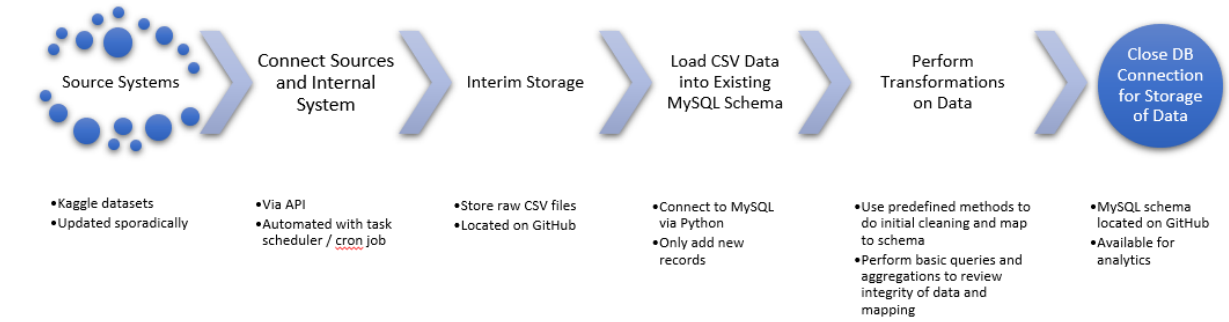


Diagram description: As seen in Figure 1, each of the four main tables (iso, emissions_gross, population, and temperature) are linked to the temporary tables in terms matching certain column values in the full temp table and existing records in the main table so that only new records are added to the latter. The iso table, which is used to standardize country column values, uses either the *iso3* or *country_code* column values to relate to the other three main tables. Lastly, the country_map table matches the *country_error* column to the main tables' *country* columns to enable updating of values prone to UTF coding errors. Also visible are two views that join several tables and select only a subset of columns.

2. Pipeline Diagram

Figure 2



STEP ONE - Establish Source System(s)

Following Figure 2 from left to right, the starting point will be the source system from where we are ingesting raw data. The source system has a data source in the form of a database or data stream. All four datasets currently used within this pipeline are sourced from Kaggle. Thus, we will work towards retrieving data from Kaggle via a direct API connection.

STEP TWO - Connect Source(s) and Internal System

In this step of the pipeline architecture, the essential data is extracted from the source system for the upcoming steps. It involves authenticating the Kaggle token and connecting to the internal system using KaggleAPI implemented in Python (see Appendix A). It will then retrieve the Kaggle datasets to the local destination of choice (e.g., GitHub desktop). Automation is also infused in the process with the use of Task Scheduler. A python script is created and runs through the Task Scheduler, which will trigger data collection every month on the 1st at 12:00 a.m. as a routine schedule. This allows the pipeline to work continuously behind the scenes without necessitating human intervention thereby increasing efficiency.

STEP THREE - Interim Storage

Following unpacking of the downloaded zip files, the resulting raw CSV files are then stored within interim storage, located on Github.

STEP FOUR - Load CSV Data into Existing MySQL Schema
The downloaded files are then loaded into an existing MySQL schema through use of Python's pymysql and pandas packages, which facilitate the embedded use of native MySQL statements. This step includes two substeps: 1) Upload the CSV into a temporary table that closely resembles the main table, but without a primary key; and 2) Only move those records considered new to the main table, which is determined by matching several related columns' values between the tables.

STEP FIVE - Perform Transformation on Data
After connection has been established, data is then transformed and modified. This includes using predefined methods to do initial cleaning and performing basic queries and aggregations to review integrity of data. After the transformation is done, the data will be mapped back to the schema. Specific transformations include:

- Melting several of the population table columns into rows to facilitate easier and more consistent joins with other tables based on year.
- Standardization of the *country* column within the emissions_gross, population, and temperature tables based on matching column values with the iso table–specifically the *iso3* or *country_code* columns depending on which column was related to the other specific tables.
- Table joins to ensure proper data updates through review and basic analytics.
- Create views limit what information is provided to users as part of adhering to the principle of least privilege (Reis & Housley, 2022). As you know, views also may increase efficiency by saving SELECT statements, which may include complex table joins.

STEP SIX - Close DB Connection for Storage of Data
Lastly, the Python-MySQL connection is closed to free up the server instance; at this stage, where the transformed data is stored in the MySQL schema on GitHub, the current pipeline is considered complete. The data is now ready for access to the next downstream/upstream users.

Pipeline Output
The output is a MySQL schema that includes multiple tables pertaining to emissions, climate temperature, country population data, and an ISO/country list. The data has undergone basic transformations to ensure correct mapping to the structures of pre-existing tables–tables were created as part of the initial pipeline creation. The transformed data can be used for general queries, analytics, business intelligence, machine learning purposes, etc.

The final output would be accessible to COP28 conference organizers and country officials to be used as regular scientific assessments on climate change. The pipeline integrates and prepares all

necessary data for quick analysis and insights, allowing decision makers to be informed of changes in factors affecting global climate timely.

Gaps in System

After conducting a code review on the pipeline created for this system, it was verified that the query syntax has been properly normalized, indexed, and designed to have limited purpose for expansion. As the system is inherently a relational database, this again limits the scalability and if to encounter issues with resources; the next steps would be transferring the system to a setup that can handle the large volume of data, such as a NoSQL database. To mitigate security issues, the team was able to limit the accessibility to this public database by hiding the credentials and token(s) (i.e., GetPass, embedding in Environment path, and Kaggle API), creating view tables, and granting SQL credentials with case-by-case needs. While granting SQL credentials, this allows another level of security and monitorization of all users that are accessing the server and to allow any permissions required by position and/or task to view or commit changes.

Monitoring

Current monitoring of the pipeline is done via Python code initiated as part of loading, inserting, and transforming of new data. After the basic configuration is set globally (Figure 3), logging triggers are instantiated along with the pymysql statements with a try/except/finally method.

Figure 3

Python Code Snippet: Log Configuration Setup

```
# Set up logging
logging.basicConfig(level=logging.INFO,
                    filename='pymysql.log',
                    filemode='a',
                    format='>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n%(asctime)s - %(levelname)s - %(message)s')
```

Figure 4 is an example of the code that dictates what message is output to a .txt logging file called pymysql.log.

Figure 4

Python Code Snippet: Log Status Output Based on Try/Except/Finally Method Criteria

```
# Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
    ist_dlt_tble_stmnt = """DELETE FROM iso_tempy"""
    cursor.execute(ist_dlt_tble_stmnt)
    logging.info(f'Successfully executed query:\n{ist_dlt_tble_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ist_dlt_tble_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f} seconds\n<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<\n\n')
```

Current outputs (as seen in Figure 5) include time of execution; whether it finished successfully, or if not, what error was produced; display of MySQL input statement; number of records scanned; and how long the script took to complete. Indicators of success or errors can be used to monitor whether the source data is mapping correctly to the existing schema. Execution time and number of scanned records can be used to monitor performance drift over time.

Figure 5

Sample Log Output

```
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2023-02-26 13:57:34,406 - INFO - Successfully executed query:

    CREATE VIEW etp_view
    AS
    SELECT
        e.country,
        e.record_year,
        e.total,
        e.coal,
        e.oil,
        e.gas,
        e.cement,
        e.flaring,
        e.other,
        e.per_capita,
        t.temp AS Temperature,
        p.density,
        p.grow_rate,
        p.pop_perc
    FROM emissions_gross e
    JOIN temperature t
        ON e.country = t.country AND e.record_year = t.record_year
    JOIN population_trans p
        ON e.country = p.country


Records scanned: 0
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2023-02-26 13:57:34,407 - INFO - Time taken: 0.006 seconds
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>


>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2023-02-26 13:57:39,733 - INFO - Successfully executed query:

    SELECT
        country,
        record_year,
        MAX(total) AS max_emission,
        MAX(Temperature) AS max_temperature
    FROM etp_view
    GROUP BY country, record_year


Records scanned: 11765
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
2023-02-26 13:57:39,733 - INFO - Time taken: 5.306 seconds
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
```

References

Reis, J., & Housley, M. (2022). *Fundamentals of data engineering: Plan and build robust data systems*. O'Reilly.

ADS507_Team2_Pipeline_Final_v2

February 27, 2023

1 Appendix A

1.1 Global setup

[1]:
```
'''Setup Kaggle API citation:
https://www.kaggle.com/docs/api'''

# Load libraries
import numpy as np
import pandas as pd
import pymysql as mysql
import matplotlib.pyplot as plt
import os
import shutil
import re
import logging
import time
import kaggle
import zipfile

# Set pandas global options
pd.options.display.max_rows = 17
```

1.2 Connect to Kaggle API

[2]:
```
# Split up-1-level path & current working folder
up1_path, curr_folder = os.path.split(os.getcwd())
print(up1_path)
print(curr_folder)
```

```
C:\Users\acarr\Documents\GitHub\ads507_data_engineering
deliverables
```

1.2.1 Setup up connection

[3]:
```
'''Setup Kaggle API connection citation: https://python.plainenglish.io/
 ↪how-to-use-the-kaggle-api-in-python-4d4c812c39c7'''
# Create Kaggle API authentication instance
from kaggle.api.kaggle_api_extended import KaggleApi
api = KaggleApi()
api.authenticate()
```

Connect to database: kaggle datasets download -d thedevastator/global-fossil-co2-emissions-by-country-2002-2022

[4]:
```
# Assignment data plaement folder
emi_place_folder = 'data\Emissions'

# Join up-1-level path to placement folder
emi_place_folder_path = os.path.join(up1_path, emi_place_folder)
print(emi_place_folder_path)
```

```
C:\Users\acarr\Documents\GitHub\ads507_data_engineering\data\Emissions
```

[5]:
```
# Assign Kaggle API link details
emi_kag_owner = 'thedevastator'
emi_kag_dataset = 'global-fossil-co2-emissions-by-country-2002-2022'
emi_kag_api_link = emi_kag_owner + '/' + emi_kag_dataset
```

[6]:
```
# Access Kaggle API and download file(s)
api.dataset_download_files(emi_kag_api_link, path=emi_place_folder_path)
```

[7]:
```
# Unzip downloaded file
emi_zip_file = emi_kag_dataset + '.zip'
emi_zip_file_path = os.path.join(emi_place_folder_path, emi_zip_file)
print(emi_zip_file_path)

with zipfile.ZipFile(emi_zip_file_path, 'r') as zipref:
    zipref.extractall(emi_place_folder_path)
```

```
C:\Users\acarr\Documents\GitHub\ads507_data_engineering\data\Emissions\global-
fossil-co2-emissions-by-country-2002-2022.zip
```

Connect to database: kaggle datasets download -d sevgisarac/temperature-change

[8]:
```
# Assignment data plaement folder
tmp_place_folder = 'data\Temperature'

# Join up-1-level path to placement folder
tmp_place_folder_path = os.path.join(up1_path, tmp_place_folder)
print(tmp_place_folder_path)
```

```
C:\Users\acarr\Documents\GitHub\ads507_data_engineering\data\Temperature
```

[9]:
```
# Assign Kaggle API link details
tmp_kag_owner = 'sevgisarac'
tmp_kag_dataset = 'temperature-change'
tmp_kag_api_link = tmp_kag_owner + '/' + tmp_kag_dataset
```

[10]:
```
# Access Kaggle API and download file(s)
api.dataset_download_files(tmp_kag_api_link, path=tmp_place_folder_path)
```

[11]:
```
# Unzip downloaded file
tmp_zip_file = tmp_kag_dataset + '.zip'
tmp_zip_file_path = os.path.join(tmp_place_folder_path, tmp_zip_file)
print(tmp_zip_file_path)

with zipfile.ZipFile(tmp_zip_file_path, 'r') as zipref:
    zipref.extractall(tmp_place_folder_path)
```

```
C:\Users\acarr\Documents\GitHub\ads507_data_engineering\data\Temperature\tempera
ture-change.zip
```

Connect to database: kaggle datasets download -d iamsouravbanerjee/world-population-dataset

[12]:
```
# Assignment data plaement folder
pop_place_folder = 'data\Population'

# Join up-1-level path to placement folder
pop_place_folder_path = os.path.join(up1_path, pop_place_folder)
print(pop_place_folder_path)
```

```
C:\Users\acarr\Documents\GitHub\ads507_data_engineering\data\Population
```

[13]:
```
# Assign Kaggle API link details
pop_kag_owner = 'iamsouravbanerjee'
pop_kag_dataset = 'world-population-dataset'
pop_kag_api_link = pop_kag_owner + '/' + pop_kag_dataset
```

[14]:
```
# Access Kaggle API and download file(s)
api.dataset_download_files(pop_kag_api_link, path=pop_place_folder_path)
```

[15]:
```
# Unzip downloaded file
pop_zip_file = pop_kag_dataset + '.zip'
pop_zip_file_path = os.path.join(pop_place_folder_path, pop_zip_file)
print(pop_zip_file_path)

with zipfile.ZipFile(pop_zip_file_path, 'r') as zipref:
    zipref.extractall(pop_place_folder_path)
```

```
C:\Users\acarr\Documents\GitHub\ads507_data_engineering\data\Population\world-
population-dataset.zip
```

1.3 Load data into MySQL tables from CSV files

1.3.1 Get credentials from local path and connect to MySQL DB

[16]:
```
'''Set local environment variables to hide user name & password citation:
https://www.geeksforgeeks.org/how-to-hide-sensitive-credentials-using-python/'''

user_name = os.environ['MySQLUSRAC']
user_pass = os.environ['MySQLPWDAC']

# Instantiate connection
db_conn = mysql.connect(host='localhost',
                        port=int(3306),
                        user=user_name,
                        passwd=user_pass,
                        db='507_final_proj')

# Create a cursor object
cursor = db_conn.cursor()
```

[17]:
```
tbl_names = pd.read_sql('SHOW TABLES', db_conn)

display(tbl_names)
print(type(tbl_names))
```

```
   Tables_in_507_final_proj
0                country_map
1            emissions_gross
2            emissions_tempy
3                   etp_view
4                        iso
5                  iso_tempy
6                 population
7           population_tempy
8           population_trans
9                 temp_core1
10               temperature
11         temperature_tempy
<class 'pandas.core.frame.DataFrame'>
```

1.3.2 Setup log parameters

[18]:
```
'''Logging citations (see additional code in following code blocks:
OpenAI. (2021). ChatGPT [Computer software]. https://openai.com/;
https://docs.python.org/3/howto/logging.html#logging-basic-example;
https://docs.python.org/3/howto/logging.html#logging-to-a-file;
https://docs.python.org/3/howto/logging-cookbook.
 ↪html#using-a-rotating-log-file-handler;
```

```
https://docs.python.org/3/howto/logging-cookbook.
 ↪html#using-a-timed-rotating-file-handler'''

# Set up logging
logging.basicConfig(level=logging.INFO,
                    filename='pymysql.log',
                    filemode='a',
                    format='>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<\n%(asctime)s -␣
 ↪%(levelname)s - %(message)s')
```

[19]:
```
'''Move files to from unpacked folder to MySQL unrestricted folder citation:
OpenAI. (2021). ChatGPT [Computer software]. https://openai.com/;
https://docs.python.org/3/library/os.html;
https://docs.python.org/3/library/shutil.html'''

import os

# Set the file paths
source_files = ['C:/Users/acarr/Documents/GitHub/ads507_data_engineering/data/
 ↪Emissions/GCB2022v27_MtCO2_flat.csv',
                'C:/Users/acarr/Documents/GitHub/ads507_data_engineering/data/
 ↪Population/world_population.csv',
                'C:/Users/acarr/Documents/GitHub/ads507_data_engineering/data/
 ↪Temperature/FAOSTAT_data_1-10-2022.csv',
                'C:/Users/acarr/Documents/GitHub/ads507_data_engineering/data/
 ↪Temperature/FAOSTAT_data_11-24-2020.csv']
destination_folder = 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads'

# Move the files to the destination folder and overwrite them if they already␣
 ↪exist
for source_file in source_files:
    file_name = os.path.basename(source_file)
    destination_file = os.path.join(destination_folder, file_name)
    os.replace(source_file, destination_file)
```

1.3.3 Update individual tables

Update iso table from CSV

[20]:
```
'''Using cursor and loading into temp file:
OpenAI. (2021). ChatGPT [Computer software]. https://openai.com/;
https://pynative.com/python-mysql-insert-data-into-database-table/'''

# Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
```

```
    ist_dlt_tble_stmnt = """DELETE FROM iso_tempy"""
    cursor.execute(ist_dlt_tble_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{ist_dlt_tble_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ist_dlt_tble_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

# Execute query and measure execution time
start_time = time.time()

# Load data from CSV file into a temporary table
try:
    ist_csv_load_stmnt = """
    LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/
 ↪FAOSTAT_data_11-24-2020.csv'
        INTO TABLE iso_tempy
    FIELDS TERMINATED BY ','
    OPTIONALLY ENCLOSED BY '"'
    LINES TERMINATED BY '\r\n'
    IGNORE 1 ROWS
    (
    country_code,
    country,
    m49_code,
    iso2,
    iso3,
    year_start,
    year_end
    )
    """
    cursor.execute(ist_csv_load_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{ist_csv_load_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ist_csv_load_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

# Execute query and measure execution time
start_time = time.time()
```

```
# Insert new records into main table
try:
    ism_load_stmtn = """
    INSERT INTO iso
    (
    country_code,
    country,
    m49_code,
    iso2,
    iso3,
    year_start,
    year_end
    )
    SELECT
        tp.country_code,
        tp.country,
        tp.m49_code,
        tp.iso2,
        tp.iso3,
        tp.year_start,
        tp.year_end
    FROM iso_tempy AS tp
    LEFT JOIN iso AS mn
        ON tp.iso3 = mn.iso3
    WHERE mn.iso3 IS NULL
    """
    cursor.execute(ism_load_stmtn)
    logging.info(f'Successfully executed query:\n{ism_load_stmtn}\n\nRecords␣
 ↪scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ism_load_stmtn}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

# Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
    cursor.execute(ist_dlt_tble_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{ist_dlt_tble_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ist_dlt_tble_stmnt}\n\n{e}')
finally:
```

```
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')
```

Update emissions_gross table from CSV

[21]:
```
# Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
    egt_dlt_tble_stmnt = """DELETE FROM emissions_tempy"""
    cursor.execute(egt_dlt_tble_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{egt_dlt_tble_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{egt_dlt_tble_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

# Execute query and measure execution time
start_time = time.time()

# Load data from CSV file into a temporary table
try:
    egt_csv_load_stmnt = """
    LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/
 ↪GCB2022v27_MtCO2_flat.csv'
        INTO TABLE emissions_tempy
    FIELDS TERMINATED BY ','
    OPTIONALLY ENCLOSED BY '"'
    LINES TERMINATED BY '\r\n'
    IGNORE 1 ROWS
    (
    country,
    iso3,
    record_year,
    total,
    coal,
    oil,
    gas,
    cement,
    flaring,
    other,
    per_capita
```

```
    )
    """
    cursor.execute(egt_csv_load_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{egt_csv_load_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{egt_csv_load_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

# Execute query and measure execution time
start_time = time.time()

# Insert new records into main table
try:
    egm_load_stmtn = """
    INSERT INTO emissions_gross
    (
    country,
    iso3,
    record_year,
    total,
    coal,
    oil,
    gas,
    cement,
    flaring,
    other,
    per_capita
    )
    SELECT
        tp.country,
        tp.iso3,
        tp.record_year,
        tp.total,
        tp.coal,
        tp.oil,
        tp.gas,
        tp.cement,
        tp.flaring,
        tp.other,
        tp.per_capita
    FROM emissions_tempy AS tp
    LEFT JOIN emissions_gross AS mn
        ON tp.iso3 = mn.iso3 AND tp.record_year = mn.record_year
```

```
    WHERE mn.iso3 IS NULL AND mn.record_year IS NULL
    """
    cursor.execute(egm_load_stmtn)
    logging.info(f'Successfully executed query:\n{egm_load_stmtn}\n\nRecords␣
 ↪scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{egm_load_stmtn}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

# Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
    egt_dlt_tble_stmnt = """DELETE FROM emissions_tempy"""
    cursor.execute(egt_dlt_tble_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{egt_dlt_tble_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{egt_dlt_tble_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')
```

Update population table from CSV

[22]:
```
# Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
    ppt_dlt_tble_stmnt = """DELETE FROM population_tempy"""
    cursor.execute(ppt_dlt_tble_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{ppt_dlt_tble_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ppt_dlt_tble_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

# Execute query and measure execution time
```

```
start_time = time.time()

# Load data from CSV file into a temporary table
try:
    ppt_csv_load_stmnt = """
    LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/
 ↪world_population.csv'
        INTO TABLE population_tempy
    FIELDS TERMINATED BY ','
    OPTIONALLY ENCLOSED BY '"'
    LINES TERMINATED BY '\r\n'
    IGNORE 1 ROWS
    (
    pop_rank,
    iso3,
    country,
    capital,
    continent,
    pop_2022,
    pop_2020,
    pop_2015,
    pop_2010,
    pop_2000,
    pop_1990,
    pop_1980,
    pop_1970,
    area,
    density,
    grow_rate,
    pop_perc
    )
    """
    cursor.execute(ppt_csv_load_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{ppt_csv_load_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ppt_csv_load_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

# Execute query and measure execution time
start_time = time.time()

# Insert new records into main table
try:
```

```
    ppm_load_stmtn = """
    INSERT INTO population
    (
    pop_rank,
    iso3,
    country,
    capital,
    continent,
    pop_2022,
    pop_2020,
    pop_2015,
    pop_2010,
    pop_2000,
    pop_1990,
    pop_1980,
    pop_1970,
    area,
    density,
    grow_rate,
    pop_perc
    )
    SELECT
        tp.pop_rank,
        tp.iso3,
        tp.country,
        tp.capital,
        tp.continent,
        tp.pop_2022,
        tp.pop_2020,
        tp.pop_2015,
        tp.pop_2010,
        tp.pop_2000,
        tp.pop_1990,
        tp.pop_1980,
        tp.pop_1970,
        tp.area,
        tp.density,
        tp.grow_rate,
        tp.pop_perc
    FROM population_tempy AS tp
    LEFT JOIN population AS mn
        ON tp.iso3 = mn.iso3
    WHERE mn.iso3 IS NULL
    """
    cursor.execute(ppm_load_stmtn)
    logging.info(f'Successfully executed query:\n{ppm_load_stmtn}\n\nRecords␣
↪scanned: {cursor.rowcount}')
```

```
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ppm_load_stmtn}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

# Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
    ppt_dlt_tble_stmnt = """DELETE FROM population_tempy"""
    cursor.execute(ppt_dlt_tble_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{ppt_dlt_tble_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ppt_dlt_tble_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')
```

Update temperature table from CSV

[23]:
```
'''Remove first row of CSV file:
OpenAI. (2021). ChatGPT [Computer software]. https://openai.com/;
https://docs.python.org/3/library/csv.html'''

import csv

# Open input and output files
input_file = 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/
 ↪FAOSTAT_data_1-10-2022.csv'
output_file = 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/
 ↪FAOSTAT_data_1-10-2022_new.csv'

with open(input_file, 'r') as csv_input_file, open(output_file, 'w',␣
 ↪newline='') as csv_output_file:
    # Create CSV reader and writer objects
    csv_reader = csv.reader(csv_input_file)
    csv_writer = csv.writer(csv_output_file)

    # Skip the first row of the input file
    next(csv_reader)

    # Write the remaining rows to the output file
```

```
    for row in csv_reader:
        csv_writer.writerow(row)
```

[24]:
```
# Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
    tpt_dlt_tble_stmnt = """DELETE FROM temperature_tempy"""
    cursor.execute(tpt_dlt_tble_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{tpt_dlt_tble_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{tpt_dlt_tble_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<\n\n')

# Execute query and measure execution time
start_time = time.time()

# Load data from CSV file into a temporary table
try:
    tpt_csv_load_stmnt = """
    LOAD DATA INFILE 'C:/ProgramData/MySQL/MySQL Server 8.0/Uploads/
 ↪FAOSTAT_data_1-10-2022_new.csv'
        INTO TABLE temperature_tempy
    FIELDS TERMINATED BY ','
    OPTIONALLY ENCLOSED BY '"'
    LINES TERMINATED BY '\r\n'
    (
    domain_code,
    domain,
    area_code,
    country,
    element_code,
    element,
    month_code,
    month_name,
    year_code,
    record_year,
    unit,
    temp,
    flag,
    flag_desc
    )
```

```
    """
    cursor.execute(tpt_csv_load_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{tpt_csv_load_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{tpt_csv_load_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

# Execute query and measure execution time
start_time = time.time()

# Insert new records into main table
try:
    tpm_load_stmtn = """
    INSERT INTO temperature
    (
    domain_code,
    domain,
    area_code,
    country,
    element_code,
    element,
    month_code,
    month_name,
    year_code,
    record_year,
    unit,
    temp,
    flag,
    flag_desc
    )
    SELECT
        tp.domain_code,
        tp.domain,
        tp.area_code,
        tp.country,
        tp.element_code,
        tp.element,
        tp.month_code,
        tp.month_name,
        tp.year_code,
        tp.record_year,
        tp.unit,
        tp.temp,
```

```
        tp.flag,
        tp.flag_desc
    FROM temperature_tempy AS tp
    LEFT JOIN temperature AS mn
        ON tp.country = mn.country AND tp.month_code = mn.month_code AND tp.
 ↪year_code = mn.year_code
    WHERE mn.country IS NULL AND  mn.month_code IS NULL AND mn.year_code IS NULL
    """
    cursor.execute(tpm_load_stmtn)
    logging.info(f'Successfully executed query:\n{tpm_load_stmtn}\n\nRecords␣
 ↪scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{tpm_load_stmtn}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<\n\n')

# Execute query and measure execution time
start_time = time.time()

# Wipe temp table
try:
    tpt_dlt_tble_stmnt = """DELETE FROM temperature_tempy"""
    cursor.execute(tpt_dlt_tble_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{tpt_dlt_tble_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{tpt_dlt_tble_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<\n\n')
```

1.3.4 Perform transformations on MySQL tables

Transform population table: Melt year cols to rows

[25]:
```
'''Convert table to pandas df, melt pop numbers form cols to rows citation:
OpenAI. (2021). ChatGPT [Computer software]. https://openai.com/;
https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.melt.html'''

ppm_slct_all_stmnt = """SELECT * FROM population"""
ppm_slct_all_df = pd.read_sql(ppm_slct_all_stmnt, db_conn)

# Melt the subset of columns
cols_to_melt = ['pop_2022',
                'pop_2020',
```

```
                'pop_2015',
                'pop_2010',
                'pop_2000',
                'pop_1990',
                'pop_1980',
                'pop_1970']
var_names = [re.sub(r'^pop_', '', col) for col in cols_to_melt]
ppm_slct_all_df_melted = pd.melt(ppm_slct_all_df, id_vars=['pop_rank',
                                                           'iso3',
                                                           'country',
                                                           'capital',
                                                           'continent',
                                                           'area',
                                                           'density',
                                                           'grow_rate',
                                                           'pop_perc'],
                                 value_vars=cols_to_melt,
                                 var_name='year',
                                 value_name='population')
#print(ppm_slct_all_df_melted.head())

# Insert the melted data into the MySQL table
insert_query = """
INSERT INTO population_trans (pop_rank, iso3, country, capital, continent,␣
 ↪area, density, grow_rate, pop_perc, year, population)
SELECT %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s FROM DUAL WHERE NOT EXISTS␣
 ↪(SELECT * FROM population_trans WHERE country = %s AND year = %s)
"""
for index, row in ppm_slct_all_df_melted.iterrows():
    variable = var_names[cols_to_melt.index(row['year'])]  # get variable name␣
 ↪based on column name
    #print(index)
    #print(row)
    #print(variable)
    values = (row['pop_rank'],
              row['iso3'],
              row['country'],
              row['capital'],
              row['continent'],
              row['area'],
              row['density'],
              row['grow_rate'],
              row['pop_perc'],
              variable,
              row['population'],
              row['country'],
              variable)
```

```
    #print(values)
    cursor.execute(insert_query, values)
```

Standardize feature values in emissions_gross table based on mapping to iso

[26]:
```
'''Update table col vals based on mapping to another table citation:
OpenAI. (2021). ChatGPT [Computer software]. https://openai.com/;
https://pynative.com/python-mysql-insert-data-into-database-table/'''

# Execute query and measure execution time
start_time = time.time()

# Update table
try:
    egm_updt_country_stmnt = """
    UPDATE emissions_gross AS t1
    INNER JOIN iso AS t2
        ON t1.iso3 = t2.iso3
    SET t1.country = t2.country
    WHERE t1.country <> t2.country AND t1.iso3 <> ''
    """
    cursor.execute(egm_updt_country_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{egm_updt_country_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{egm_updt_country_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

# Execute query and measure execution time
start_time = time.time()

# Update table
try:
    egm_add_cc_stmnt = """
    UPDATE emissions_gross AS t1
    INNER JOIN iso AS t2
        ON t1.country = t2.country
    SET t1.country_code = t2.country_code
    """
    cursor.execute(egm_add_cc_stmnt)
    logging.info(f'Successfully executed query:\n{egm_add_cc_stmnt}\n\nRecords␣
 ↪scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{egm_add_cc_stmnt}\n\n{e}')
```

```
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')
```

Standardize feature values in population_trans table based on mapping to iso

[27]:
```
# Execute query and measure execution time
start_time = time.time()

# Update table
try:
    ptm_updt_country_stmnt = """
    UPDATE population_trans AS t1
    INNER JOIN iso AS t2
        ON t1.iso3 = t2.iso3
    SET t1.country = t2.country
    WHERE t1.country <> t2.country AND t1.iso3 <> ''
    """
    cursor.execute(ptm_updt_country_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{ptm_updt_country_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ptm_updt_country_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

# Execute query and measure execution time
start_time = time.time()

# Update table
try:
    ptm_add_cc_stmnt = """
    UPDATE population_trans AS t1
    INNER JOIN iso AS t2
        ON t1.country = t2.country
    SET t1.country_code = t2.country_code
    """
    cursor.execute(ptm_add_cc_stmnt)
    logging.info(f'Successfully executed query:\n{ptm_add_cc_stmnt}\n\nRecords␣
 ↪scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{ptm_add_cc_stmnt}\n\n{e}')
finally:
    end_time = time.time()
```

```
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')
```

Standardize feature values in temperature table based on mapping to iso### Update temperature table

[28]:
```
# Execute query and measure execution time
start_time = time.time()

# Update table
try:
    tpm_updt_country_stmnt = """
    UPDATE temperature AS t1
    INNER JOIN country_map AS t2
        ON t1.country = t2.country_error
    INNER JOIN iso AS t3
        ON t2.country_code = t3.country_code
    SET t1.country = t3.country
    WHERE t1.country <> t3.country
    """
    cursor.execute(tpm_updt_country_stmnt)
    logging.info(f'Successfully executed query:
 ↪\n{tpm_updt_country_stmnt}\n\nRecords scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{tpm_updt_country_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

# Execute query and measure execution time
start_time = time.time()

# Update table
try:
    tpm_add_cc_stmnt = """
    UPDATE temperature AS t1
    INNER JOIN iso AS t2
        ON t1.country = t2.country
    SET t1.country_code = t2.country_code
    """
    cursor.execute(tpm_add_cc_stmnt)
    logging.info(f'Successfully executed query:\n{tpm_add_cc_stmnt}\n\nRecords␣
 ↪scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{tpm_add_cc_stmnt}\n\n{e}')
finally:
```

```
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')
```

[29]:
```
# Extract data from each table - emissions
start_time = time.time()

try:
    emissions_query = "SELECT * FROM emissions_gross;"
    emissions_df = pd.read_sql(emissions_query, db_conn)

    display(emissions_df)
    logging.info(f'Successfully executed query:\n{emissions_query}\n\nRecords␣
 ↪scanned: {len(emissions_df)}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{emissions_query}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')
```

```
       eg_id      country iso3 record_year         total          coal  \
0          1  Afghanistan  AFG        1750             0
1          2  Afghanistan  AFG        1751             0
2          3  Afghanistan  AFG        1752             0
3          4  Afghanistan  AFG        1753             0
4          5  Afghanistan  AFG        1754             0
…        …           …  …          …           …            …
63099  63100       Global  WLD        2017  36096.739276  14506.973805
63100  63101       Global  WLD        2018  36826.506600  14746.830688
63101  63102       Global  WLD        2019  37082.558969  14725.978025
63102  63103       Global  WLD        2020  35264.085734  14174.564010
63103  63104       Global  WLD        2021  37123.850352  14979.598083

                oil          gas       cement     flaring       other  \
0
1
2
3
4
…              …            …           …           …           …
63099  12242.627935  7144.928128  1507.923185  391.992176  302.294047
63100  12266.016285  7529.846784  1569.218392  412.115746  302.478706
63101  12345.653374  7647.528220  1617.506786  439.253991  306.638573
63102  11191.808551  7556.290283  1637.537532  407.583673  296.301685
63103  11837.159116  7921.829472  1672.592372  416.525563  296.145746

      per_capita country_code
```

```
0                          2
1                          2
2                          2
3                          2
4                          2
…          …          …
63099   4.749682         None
63100   4.792753         None
63101   4.775633         None
63102   4.497423         None
63103   4.693699         None

[63104 rows x 13 columns]
```

[30]:
```
# Extract data from each table - temperature
start_time = time.time()

try:
    temperature_query = "SELECT * FROM temperature;"
    temperature_df = pd.read_sql(temperature_query, db_conn)

    display(temperature_df)
    logging.info(f'Successfully executed query:\n{temperature_query}\n\nRecords␣
 ↪scanned: {len(temperature_df)}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{temperature_query}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')
```

```
       domain_code              domain area_code      country element_code  \
0               ET  Temperature change         2  Afghanistan         7271
1               ET  Temperature change         2  Afghanistan         7271
2               ET  Temperature change         2  Afghanistan         7271
3               ET  Temperature change         2  Afghanistan         7271
4               ET  Temperature change         2  Afghanistan         7271
…              …                 …       …           …           …
244200          ET  Temperature change       182      Réunion         7271
244201          ET  Temperature change       182      Réunion         7271
244202          ET  Temperature change       182      Réunion         7271
244203          ET  Temperature change       182      Réunion         7271
244204          ET  Temperature change       182      Réunion         7271

                   element month_code          month_name year_code  \
0       Temperature change       7001             January      1961
1       Temperature change       7001             January      1962
2       Temperature change       7001             January      1963
```

```
3       Temperature change       7001              January      1964
4       Temperature change       7001              January      1965
…                      …        …                    …         …
244200  Temperature change       7003                March      2005
244201  Temperature change       7008                August     1971
244202  Temperature change       7008                August     1999
244203  Temperature change       7011              November     1975
244204  Temperature change       7020  Meteorological year      2008

       record_year unit    temp flag        flag_desc country_code
0             1961   ?C   0.746   Fc  Calculated data            2
1             1962   ?C   0.009   Fc  Calculated data            2
2             1963   ?C   2.695   Fc  Calculated data            2
3             1964   ?C  -5.277   Fc  Calculated data            2
4             1965   ?C   1.827   Fc  Calculated data            2
…              …  …     …  …               …          …
244200        2005   ?C   1.401   Fc  Calculated data          182
244201        1971   ?C  -0.504   Fc  Calculated data          182
244202        1999   ?C    1.11   Fc  Calculated data          182
244203        1975   ?C   0.706   Fc  Calculated data          182
244204        2008   ?C   0.652   Fc  Calculated data          182

[244205 rows x 15 columns]
```

[31]:
```
# Extract data from each table - population_trans
start_time = time.time()

try:
    worldpop_query = "SELECT * FROM population_trans;"
    worldpop_df = pd.read_sql(worldpop_query, db_conn)

    display(worldpop_df)
    logging.info(f'Successfully executed query:\n{worldpop_query}\n\nRecords␣
 ↪scanned: {len(worldpop_df)}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{worldpop_query}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<\n\n')
```

```
     pop_trans_id pop_rank iso3                         country  \
0               1       36  AFG                     Afghanistan
1               2      138  ALB                         Albania
2               3       34  DZA                         Algeria
3               4      213  ASM                  American Samoa
4               5      203  AND                         Andorra
…             …       …   …                              …
```

```
3547          3548        3  USA            United States of America
3548          3549      234  VAT                                Holy See
3549          3550       51  VEN  Venezuela (Bolivarian Republic of)
3550          3551       16  VNM                                Viet Nam
3551          3552      226  WLF          Wallis and Futuna Islands

                capital      continent     area   density grow_rate pop_perc  \
0                 Kabul           Asia   652230   63.0587    1.0257     0.52
1                Tirana         Europe    28748   98.8702    0.9957     0.04
2               Algiers         Africa  2381741   18.8531    1.0164     0.56
3             Pago Pago        Oceania      199  222.4774    0.9831        0
4     Andorra la Vella         Europe      468  170.5641      1.01        0
…                   …            …       …        …        …       …
3547  Washington, D.C.  North America  9372610   36.0935    1.0038     4.24
3548      Vatican City         Europe        1       510     0.998        0
3549           Caracas  South America   916445    30.882    1.0036     0.35
3550             Hanoi           Asia   331212  296.4472    1.0074     1.23
3551          Mata-Utu        Oceania      142    81.493    0.9953        0

      year population country_code
0     2022   41128771            2
1     2022    2842321            3
2     2022   44903225            4
3     2022      44273            5
4     2022      79824            6
…    …        …           …
3547  1970  200328340          231
3548  1970        752           94
3549  1970   11355475          236
3550  1970   41928849          237
3551  1970       9377          243

[3552 rows x 13 columns]
```

[32]:
```python
# Transformation step - this is for country, year, total, temperature, and population only

# Extract data from each table - emissions
start_time = time.time()

try:
    t1_stmnt = """
    SELECT
        e.Country,
        e.Record_year,
        e.Total,
        t.temp AS Temperature,
```

```
        p.population AS Population
    FROM emissions_gross e
    JOIN temperature t
        ON e.country = t.country AND e.record_year = t.record_year
    JOIN population_trans p
        ON e.country = p.country
    ORDER BY e.country, e.record_year"""
    Transform1 = pd.read_sql(t1_stmnt, db_conn)
    display(Transform1)
    logging.info(f'Successfully executed query:\n{t1_stmnt}\n\nRecords scanned:␣
 ↪{len(Transform1)}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{t1_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')
```

	Country	Record_year	Total	Temperature	Population
0	Afghanistan	1961	0.490798	-0.121	38972230
1	Afghanistan	1961	0.490798	-0.121	33753499
2	Afghanistan	1961	0.490798	-0.121	28189672
3	Afghanistan	1961	0.490798	-0.121	19542982
4	Afghanistan	1961	0.490798	-0.121	10694796
…	…	…	…	…	…
3432227	Zimbabwe	2020	10.607897	0.568	14154937
3432228	Zimbabwe	2020	10.607897	0.568	15669666
3432229	Zimbabwe	2020	10.607897	0.568	16320537
3432230	Zimbabwe	2020	10.607897	0.746	5202918
3432231	Zimbabwe	2020	10.607897	0.746	7049926

[3432232 rows x 5 columns]

[33]:
```
# General transformation to include as much raw data to show relationships␣
 ↪between all datasets for emissions, temperature, and population.

# Extract data from each table - emissions
start_time = time.time()

try:
    t2_stmnt = """
    SELECT
        e.total AS emissions_total,
        e.country AS emissions_country,
        e.record_year AS emissions_year,
        t.country AS temperature_country,
        t.temp AS temperature_temp,
```

```
        t.record_year AS temperature_year,
        p.year AS population_year,
        p.pop_perc AS population_percentage
    FROM emissions_gross e
    INNER JOIN temperature t
        ON e.country_code = t.country_code AND e.record_year = t.record_year
    INNER JOIN population_trans p
        ON e.country_code = p.country_code AND e.record_year = p.year
    ORDER BY emissions_country, emissions_year;
    """
    Transform2 = pd.read_sql(t2_stmnt, db_conn)
    display(Transform2)
    logging.info(f'Successfully executed query:\n{t2_stmnt}\n\nRecords scanned:␣
 ↪{len(Transform2)}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{t2_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<\n\n')
```

	emissions_total	emissions_country	emissions_year	temperature_country	\
0	1.670397	Afghanistan	1970	Afghanistan	
1	1.670397	Afghanistan	1970	Afghanistan	
2	1.670397	Afghanistan	1970	Afghanistan	
3	1.670397	Afghanistan	1970	Afghanistan	
4	1.670397	Afghanistan	1970	Afghanistan	
…	…	…	…	…	
50740	10.607897	Zimbabwe	2020	Zimbabwe	
50741	10.607897	Zimbabwe	2020	Zimbabwe	
50742	10.607897	Zimbabwe	2020	Zimbabwe	
50743	10.607897	Zimbabwe	2020	Zimbabwe	
50744	10.607897	Zimbabwe	2020	Zimbabwe	

	temperature_temp	temperature_year	population_year	population_percentage
0	0.813	1970	1970	0.52
1	-0.536	1970	1970	0.52
2	-0.189	1970	1970	0.52
3	0.505	1970	1970	0.52
4	-0.907	1970	1970	0.52
…	…	…	…	…
50740	1.072	2020	2020	0.2
50741	0.502	2020	2020	0.2
50742	0.095	2020	2020	0.2
50743	0.706	2020	2020	0.2
50744	1.471	2020	2020	0.2

[50745 rows x 8 columns]

[34]:
```
# Main Transformation - depicts emission factors, temperature, and population␣
 ↪factors

# Extract data from each table - emissions
start_time = time.time()

try:
    t3_stmnt = """
    SELECT
        e.country,
        e.record_year,
        e.total,
        e.coal,
        e.oil,
        e.gas,
        e.cement,
        e.flaring,
        e.other,
        e.per_capita,
        t.temp AS Temperature,
        p.density,
        p.grow_rate,
        p.pop_perc
    FROM emissions_gross e
    JOIN temperature t
        ON e.country = t.country AND e.record_year = t.record_year
    JOIN population_trans p
        ON e.country = p.country
    """
    Transform3 = pd.read_sql(t3_stmnt, db_conn)
    display(Transform3)
    logging.info(f'Successfully executed query:\n{t3_stmnt}\n\nRecords scanned:␣
 ↪{len(Transform3)}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{t3_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>><<<<<<<<<<<<<<<<<<<<\n\n')
```

	country	record_year	total	coal	oil	gas	\
0	Afghanistan	1961	0.490798	0.175872	0.293120	0	
1	Afghanistan	1961	0.490798	0.175872	0.293120	0	
2	Afghanistan	1961	0.490798	0.175872	0.293120	0	
3	Afghanistan	1961	0.490798	0.175872	0.293120	0	
4	Afghanistan	1961	0.490798	0.175872	0.293120	0	
…	…	…	…	…	…	…	
3432227	Viet Nam	1997	44.516863	21.527760	18.398448	1.139597	

```
3432228     Viet Nam        1997  44.516863  21.527760  18.398448  1.139597
3432229     Viet Nam        1997  44.516863  21.527760  18.398448  1.139597
3432230     Viet Nam        1997  44.516863  21.527760  18.398448  1.139597
3432231     Viet Nam        1997  44.516863  21.527760  18.398448  1.139597

           cement flaring other per_capita Temperature   density grow_rate  \
0        0.021806       0         0.055835       0.746   63.0587    1.0257
1        0.021806       0         0.055835       0.746   63.0587    1.0257
2        0.021806       0         0.055835       0.746   63.0587    1.0257
3        0.021806       0         0.055835       0.746   63.0587    1.0257
4        0.021806       0         0.055835       0.746   63.0587    1.0257
…             …     …   …         …          …       …       …
3432227  3.451058       0         0.585297       0.303  296.4472    1.0074
3432228  3.451058       0         0.585297       0.303  296.4472    1.0074
3432229  3.451058       0         0.585297       0.303  296.4472    1.0074
3432230  3.451058       0         0.585297       0.303  296.4472    1.0074
3432231  3.451058       0         0.585297       0.303  296.4472    1.0074

        pop_perc
0           0.52
1           0.52
2           0.52
3           0.52
4           0.52
…           …
3432227     1.23
3432228     1.23
3432229     1.23
3432230     1.23
3432231     1.23

[3432232 rows x 14 columns]
```

[35]:
```python
# Create a view for data security purposes and hide complexity of queries

view_drp_stmnt = """DROP VIEW IF EXISTS etp_view"""
cursor.execute(view_drp_stmnt)

# Create a cursor object
cursor = db_conn.cursor()

# Execute query and measure execution time
start_time = time.time()

# Execute the CREATE VIEW query
try:
    create_view_query = """
```

```
    CREATE VIEW etp_view
    AS
    SELECT
        e.country,
        e.record_year,
        e.total,
        e.coal,
        e.oil,
        e.gas,
        e.cement,
        e.flaring,
        e.other,
        e.per_capita,
        t.temp AS Temperature,
        p.density,
        p.grow_rate,
        p.pop_perc
    FROM emissions_gross e
    JOIN temperature t
        ON e.country = t.country AND e.record_year = t.record_year
    JOIN population_trans p
        ON e.country = p.country
    """
    cursor.execute(create_view_query)
    logging.info(f'Successfully executed query:\n{create_view_query}\n\nRecords␣
 ↪scanned: {cursor.rowcount}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{create_view_query}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

# Commit the changes to the database
db_conn.commit()

#References: OpenAI. (2021). ChatGPT [Computer software]. https://openai.com/
```

[36]:
```
# Query with the View - to result highest total emissions and temperature␣
 ↪recorded for each country every year

# Extract data from each table - emissions
start_time = time.time()

try:
    vq_stmnt = """
    SELECT
```

```
        country,
        record_year,
        MAX(total) AS max_emission,
        MAX(Temperature) AS max_temperature
    FROM etp_view
    GROUP BY country, record_year
    """
    View_query = pd.read_sql(vq_stmnt, db_conn)
    display(View_query)
    logging.info(f'Successfully executed query:\n{vq_stmnt}\n\nRecords scanned:␣
 ↪{len(View_query)}')
except mysql.Error as e:
    logging.error(f'Error executing query:\n{vq_stmnt}\n\n{e}')
finally:
    end_time = time.time()
    logging.info(f'Time taken: {end_time - start_time:.3f}␣
 ↪seconds\n>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>\n\n')

#reference: Beaulieu, A. (2020). Learning SQL: Generate, manipulate, and␣
 ↪retrieve data (3rd ed.). O'Reilly.
```

	country	record_year	max_emission	max_temperature
0	Afghanistan	1961	0.490798	1.404
1	Afghanistan	1962	0.688594	2.397
2	Afghanistan	1963	0.706736	3.863
3	Afghanistan	1964	0.838551	1.608
4	Afghanistan	1965	1.006917	2.159
…	…	…	…	…
11760	Uruguay	1988	4.785973	2.343
11761	Vanuatu	1993	0.062288	0.087
11762	Vanuatu	2007	0.098928	1.692
11763	Venezuela (Bolivarian Republic of)	1985	101.026511	0.203
11764	Viet Nam	1997	44.516863	1.579

```
[11765 rows x 4 columns]
```

1.3.5 Commit changes and close cursor and connection instances

[37]:
```
# Commit the changes to the database
db_conn.commit()

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