**Documentación detallada del proceso ETL y consolidación de tablas**

**1. Instalación de librerías necesarias**

python

CopiarEditar

!pip install pandas pymysql

Se instalan dos paquetes esenciales:

* pandas: librería para análisis y manipulación de datos estructurados.
* pymysql: conector que permite a Python comunicarse con bases de datos MySQL.

**2. Importación de módulos**

python

CopiarEditar

from google.colab import files

import pandas as pd

from sqlalchemy import create\_engine

* files permite subir archivos locales desde el entorno de Google Colab.
* pandas para cargar y procesar los archivos CSV.
* create\_engine de SQLAlchemy para establecer conexión a la base de datos.

**3. Configuración de conexión a MySQL**

python

CopiarEditar

user = "root"

password = "1234"

host = "34.10.20.53"

port = "3306"

database = "covid\_db"

engine = create\_engine(f"mysql+pymysql://{user}:{password}@{host}:{port}/{database}")

Se definen las credenciales de acceso a la base de datos desplegada en Google Cloud Platform (GCP).  
engine actúa como una interfaz que permite ejecutar consultas y cargar datos directamente a las tablas de MySQL.

**4. Carga de archivos CSV**

python

CopiarEditar

csv\_files = ['Department.csv', 'cases.csv', 'gender.csv', 'municipality.csv', 'status.csv', 'type\_contagion.csv']

uploaded = files.upload()

* Se define la lista de archivos esperados.
* Luego, se abre un diálogo en Colab para seleccionar los archivos CSV locales.

**5. Procesamiento y carga de datos a MySQL**

python

CopiarEditar

for filename in uploaded.keys():

if filename in csv\_files:

...

* **Verifica que el archivo esté en la lista permitida.**
* **Lee el CSV** con pandas, usando sep=';' (separador punto y coma).
* Limpia los nombres de columnas: quita espacios y símbolos raros para estandarizar.
* Extrae el nombre de la tabla (removiendo .csv) y carga los datos a MySQL con to\_sql(...).

python

CopiarEditar

df.to\_sql(name=table\_name, con=engine, if\_exists='replace', index=False)

Este método **crea o reemplaza** las tablas en la base covid\_db.

**6. Lectura de las tablas desde la base**

python

CopiarEditar

dfs = {}

for file in csv\_files:

table\_name = file.replace('.csv', '')

dfs[file] = pd.read\_sql\_table(table\_name, engine)

Cada tabla de MySQL es leída nuevamente desde la base y almacenada en un diccionario dfs usando como clave el nombre del archivo. Esto permite trabajar con los datos en memoria.

**7. Renombrar columnas duplicadas**

python

CopiarEditar

dfs['gender.csv'] = dfs['gender.csv'].rename(columns={'name': 'gender\_name'})

...

Muchas tablas tienen una columna name, por lo que se renombran para evitar conflictos al hacer merge.  
Esto permite que en la tabla final sepamos a qué entidad se refiere cada nombre.

**8. Consolidación de las tablas**

python

CopiarEditar

consolidated\_df = dfs['cases.csv'] \

.merge(dfs['gender.csv'], on='id\_gender', how='left') \

...

Se hace un **merge progresivo** de las tablas relacionadas con cases.csv usando las llaves foráneas:

* id\_gender → tabla de géneros
* id\_municipality → municipios
* id\_status → estado (activo, recuperado, fallecido)
* id\_type → tipo de contagio
* id\_department → departamento

El resultado es un único DataFrame completo que consolida toda la información relevante para análisis y dashboards.

**9. Carga de la tabla consolidada**

python

CopiarEditar

consolidated\_df.to\_sql(name='consolidada', con=engine, if\_exists='replace', index=False)

La tabla final es cargada a la base covid\_db bajo el nombre consolidada.  
Esto facilita su uso directo desde herramientas como Looker Studio o Power BI.

**Resultado final**

La ejecución completa de este pipeline permite:

* Automatizar la carga de datos desde archivos locales.
* Establecer relaciones entre entidades del modelo de datos.
* Consolidar la información en una sola tabla lista para análisis.
* Publicarla en un entorno cloud accesible para dashboards.