Guía de Usuario
Base de datos Alejandría

Luisa Fernanda Buriticá Ruiz
Marcela Echeverri Gallego

Grupo de investigación FACom
Universidad de Antioquia
Medellín
Junio 2022

Agradecimientos

Agradecemos principalmente a nuestros tutores el profesor Alejandro Martinez y el profesor Esteban Silva por su guía y consejos durante todo el proceso de aprendizaje para la elaboración de este proyecto. A nuestros compañeros en el grupo de investigación FACom (Grupo de Física y Astrofísica Computacional) por su constante ayuda. Finalmente agradecemos la oportunidad que tuvimos de pertenecer y de aprender en el proceso, porque nos inspiró a crecer mucho más como personas y como investigadoras.

Índice general

Capítulo 1

Introducción

Este manual se ha diseñado para guiar al usuario en la creación, descarga y actualización de las observaciones de precipitación y temperatura del IDEAM. Se ha implementado el lenguaje de programación Python y el lenguaje para administrar bases de datos PostgreSQL, pgAdmin es la aplicación que gestiona y administra PostgreSQL. Para que la guía sea efectiva es necesario que el usuario tenga conocimientos básicos sobre los lenguajes antes mencionados y descargue correctamente las aplicaciones solicitadas.

El objetivo de la base de datos Alejandría es proporcionar una herramienta pública que facilite el uso de los datos observados por el IDEAM, corrigiendo errores de tipeo, omitiendo estaciones sin datos y concentrado varias variables en un solo conjunto de datos, y que además, sea una herramienta para el estudiante que desee indagar más en el estudio de las ciencias ambientales en nuestra región.

Capítulo 2

Definiciones iniciales

La base de datos en formato PostgreSQL contiene 12 tablas relacionadas entre sí por medio de llaves foráneas con la información de una serie de estaciones distribuidas por todo el territorio nacional, que recopilan en tiempo real datos de algunas variables meteorológicas. La figura 1 es el esquema completo de alejandría con sus tablas y sus respectivas relaciones.

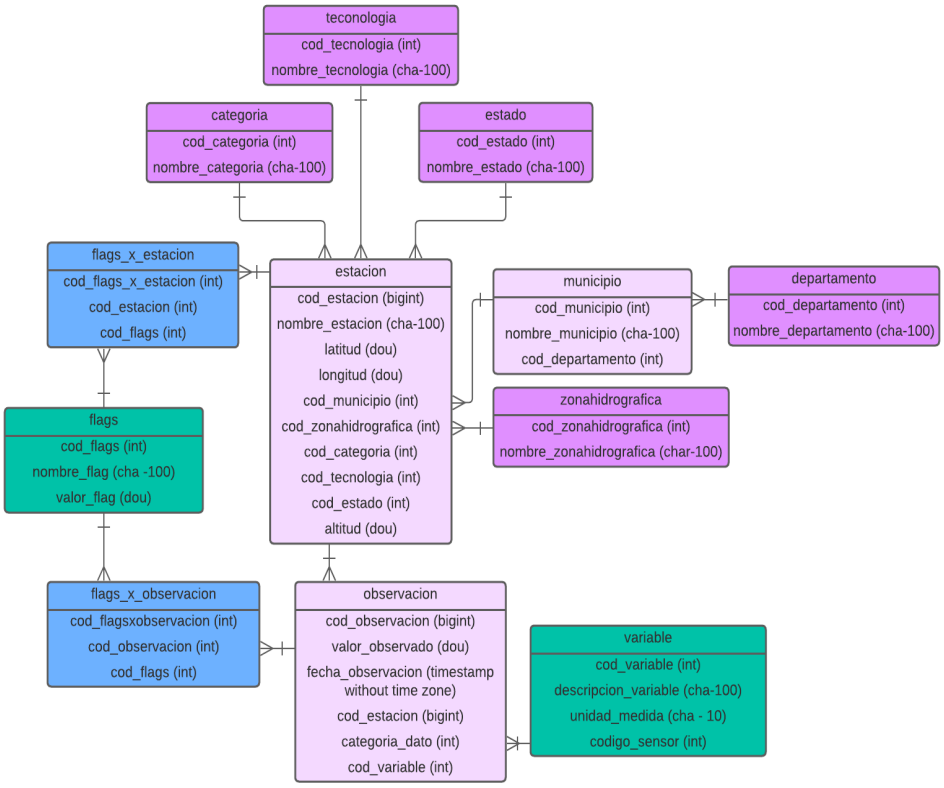

Figura 1: Esquema base de datos alejandría

A continuación se presenta una descripción de cada una de las tablas de la base de datos con sus respectivas columnas.

Tabla categoria

Columnas:

- **cod_categoria (integer):** Llave primaria autoincremental.
- **nombre_categoria (character varying(100)):** Categorías de las estaciones de acuerdo con su clase.

Tabla tecnologia

Columnas:

- **cod_tecnologia (integer):** Llave primaria autoincremental.
- **nombre_tecnologia (character varying(100)):** Tipos de estaciones de medición: convencional, automática con telemetría, automática sin telemetría.

Tabla estado

Columnas:

- **cod_estado (integer):** Llave primaria autoincremental.
- **nombre_estado (character varying(100)):** Condiciones de funcionamiento en las que se encuentra actualmente la estación.

Tabla departamento

Columnas:

- **cod_departamento (integer):** Llave primaria autoincremental.
- **nombre_departamento (character varying(100)):** Contiene los 32 departamentos de Colombia y Bogotá DC.

Tabla zonahidrografica

Columnas:

- **cod_zonahidrografica (integer):** Llave primaria autoincremental.
- **nombre_zonahidrografica (character varying(100)):** Zonas Hidrograficas en las que se encuentran las estaciones.

Tabla variable

Columnas:

- **cod_variable (integer):** Llave primaria autoincremental.
- **descripcion variable (character varying(100)):** Breve descripción de la variable meteorológica.
- **unidad_medida (character varying (100)):** Unidades de las variables en medición.
- **codigo_sensor (integer):** Código del sensor que realiza las mediciones.

Tabla municipio

Columnas:

- **cod_municipio (integer):** Llave primaria autoincremental.
- **nombre_municipio (character varying (100)):** Municipios de Colombia donde se encuentran las estaciones.
- **cod_departamento (integer):** Llave foránea que relaciona cada municipio con la tabla departamento.

Tabla estacion

Columnas:

- **cod_estacion (bigint):** Llave primaria a partir del código (identificador único) de cada estación.
- **nombre_estacion (character varying (100)):** Nombre identificador de las estaciones meteorológicas.
- **latitud (double precision):** Latitud geográfica de la estación en grados.
- **longitud (double precision):** Longitud geográfica de la estación en grados.
- **altitud (double precision):** Altura sobre el nivel del mar a la que se encuentra la estación en metros.
- **cod_municipio (double precision):** Llave foránea que relaciona cada estación con la tabla municipio.
- **cod_municipio (integer):** Llave foránea que relaciona cada estación con la tabla municipio.
- **cod_zonahidrografica (integer):** Llave foránea que relaciona cada estación con la tabla zonahidrografica.
- **cod_categoria (integer):** Llave foránea que relaciona cada estación con la tabla categoría.

- **cod_tecnologia (integer):** Llave foránea que relaciona cada estación con la tabla tecnologia.
- **cod_estado (integer):** Llave foránea que relaciona cada estación con la tabla estado.

Tabla observacion

Columnas:

- **cod_observacion (bigint):** Llave primaria autoincremental.
- **valor_observado (double precision):** Dato numérico resultante de la medición de alguna de las variables en las estaciones.
- **categoria_dato (integer):** Categorización de los valores observados, 0 = Dato dentro de la estadística general de la variable, 1 = Dato fuera de la estadística general de la variable.
- **cod_estacion (bigint):** Llave foránea que relaciona cada valor observado con la tabla estacion.
- **fecha_observacion (timestamp without time zone):** Fecha en la que se tomó el dato.
- **cod_variable (integer):** Llave foránea que relaciona cada valor observado con la tabla variable.

Información relevante

Durante el análisis de los datos se encontró que algunas de las estaciones meteorológicas cambiaron su código de estación con el paso del tiempo y por ende, el cod_estacion ya no existía en el catálogo nacional de estaciones del IDEAM. Los datos de las estaciones que cambiaron su código se almacenaron en la base de datos con el código de estación actual, con el fin de agilizar las búsquedas y unificar la información, sin embargo, a continuación se presenta un registro de los códigos modificados y las fechas respectivas a cada codigo.

- **nombre_estacion:** aeropuerto rafael nunez
  - codigo 14015020 entre 2014-09-02 16:10:00 y 2019-09-14 23:50:00
  - codigo 14015080 entre 2019-09-18 00:00:00 y actualidad
- **nombre_estacion:** aeropuerto vasquez cobo
  - codigo 48015010 entre 2014-09-02 16:10:00 y 2018-10-11 14:00:00
  - codigo 48015050 entre 2018-10-11 16:10:00 y actualidad
- **nombre_estacion:** PERENCO: TRINIDAD METEO
  - código 35237040, la mayoría de los datos que contiene son <nil>, se descarta por completo la estación.
- **nombre_estacion:** ECI JULIO GARAVITO EST. EN PRUEBAS
  - código 88112901, la mayoría de los datos que contiene son <nil>, se descarta por completo la estación.
- **nombre_estacion:** PLUVIOMETROS IDEAM BOGOTA Pruebas GPRS
  - código 21202270, tiene datos, estación en pruebas.
- **nombre_estacion:** PERENCO: LA CABANA - TERMO ELECTRICA
  - código 35217080, tiene datos, estación en pruebas.
- **nombre_estacion:** PERENCO: OROCUE PIPESCA
  - codigo 35227020, la mayoría de los datos que contiene son <nil>, se descarta por completo la estación.
- **nombre_estacion:** BARRANCABERMEJA
  - código 23157050, la mayoría de los datos que contiene son <nil>, se descarta por completo la estación.
- **nombre_estacion:** PTO NUEVO PATIA - En Siniestro
  - código 23157050, la mayoría de los datos que contiene son <nil>, se descarta por completo la estación.

Capítulo 3

¿Quién es el usuario?

Alejandría es una herramienta útil para aquellas personas que estén realizando estudios con observaciones de temperatura y/o precipitación, pero además es necesario tener conocimiento básico sobre SQL y Python; sin embargo, al final de la guía de usuario se proporcionan varios ejemplos de búsquedas tanto desde PgAdmin como desde Spyder.

Capítulo 4

Requerimientos del equipo

Para usar la base de datos alejandría es necesario tener instalados en el equipo varios programas, entre ellos, debe tener un editor de python (como Visual Studio, Spyder, Jupyter Notebook o algún otro de su preferencia) junto con el lenguaje y las siguientes librerías instaladas:

- pandas
- numpy
- datetime
- tqdm
- sqlalchemy
- re
- matplotlib

Nuestra recomendación es instalar el software Anaconda que ya contiene el lenguaje, varios editores de texto y las librerías necesarias.

Debe instalar y configurar el sistema de gestión de bases de datos relacional **postgresql** junto con la interfaz gráfica **pgAdmin** allí es donde se alojará la base de datos. En el siguiente capítulo se explica paso a paso cómo crear el usuario en postgres.

Además debe tener acceso al repositorio de github `https://github.com/luisaburu/alejandria.git` donde se encuentra este manual y varios archivos que deberá descargar más adelante para realizar la creación y automatización completa de la base de datos.

Capítulo 5

¿Cómo crear el usuario y la base de datos Alejandría?

El usuario y la base de datos son los pasos siguientes a la descarga e instalación de pgAdmin; para la creación de estos, siga los pasos mencionados a continuación. Es necesario que copie y pegue una a una las instrucciones en el orden aquí mencionado sin modificar o saltar líneas de código a menos que requiera un resultado diferente.

Nota: El usuario usado en la elaboración de esta base de datos es "facom", en caso de que se desee cambiar el usuario es necesario modificar las líneas de código de los pasos 2, 3 y 4.

Paso 1: Ingresar a PgAdmin desde la terminal de Ubuntu, para ello es necesario que desde la terminal de Ubuntu ingrese la siguiente línea de código.

```
sudo su - postgres
```

A continuación se solicitará la contraseña del usuario del equipo que esté usando. Luego, se debe abrir una linea de codigo sql escribiendo:

```
psql
```

Paso 2: Creación de usuario y/o super usuario.

```
CREATE USER facom WITH password usuario
ALTER USER mysuper WITH SUPERUSER
```

Paso 3 : Creación de la base de datos Alejandría.

Nota : El nombre de la base de datos no contiene mayúsculas ni tildes.

```
CREATE DATABASE alejandria WITH OWNER = facom ENCODING = 'UTF8' CONNECTION LIMIT = -1;
```

Paso 4 : Creación de tablas, llaves primarias y llaves foráneas.

Se debe inicialmente ingresar a la base de datos para modificarla.

```
\c alejandria
```

Ingresar a continuación las líneas de códigos completas que ejecutan las tablas y relaciones entre estas.

```
CREATE TABLE public.categoria ( cod_categoria serial PRIMARY KEY, nombre_categoria
character varying(100)); ALTER TABLE IF EXISTS public.categoria OWNER to facom; CREATE
TABLE public.departamento ( cod_departamento serial PRIMARY KEY, nombre_departamento
character varying(100)); ALTER TABLE IF EXISTS public.departamento OWNER to facom;
CREATE TABLE public.tecnologia ( cod_tecnologia serial PRIMARY KEY, nombre_tecnologia
character varying(100)); ALTER TABLE IF EXISTS public.tecnologia OWNER to facom;CREATE
TABLE public.estado ( cod_estado serial PRIMARY KEY, nombre_estado character varying
(100)); ALTER TABLE IF EXISTS public.estado OWNER to facom;CREATE TABLE public.
zonahidrografica ( cod_zonahidrografica serial PRIMARY KEY, nombre_zonahidrografica
character varying(100)); ALTER TABLE IF EXISTS public.zonahidrografica OWNER to facom;
CREATE TABLE public.variable ( cod_variable serial PRIMARY KEY, descripcion_variable
character varying(100), unidad_medida character varying(10), codigo_sensor integer);
ALTER TABLE IF EXISTS public.variable OWNER to facom;CREATE TABLE public.municipio (
cod_municipio serial PRIMARY KEY, nombre_municipio character varying(100),
cod_departamento integer); ALTER TABLE IF EXISTS public.municipio OWNER to facom; ALTER
TABLE IF EXISTS public.municipio ADD CONSTRAINT pf_departamento FOREIGN KEY (
cod_departamento) REFERENCES public.departamento (cod_departamento) MATCH SIMPLE;CREATE
TABLE public.estacion ( cod_estacion bigserial PRIMARY KEY, nombre_estacion character
varying(100), latitud double precision, longitud double precision, cod_municipio integer
, cod_zonahidrografica integer, cod_categoria integer, cod_tecnologia integer,
cod_estado integer, altitud double precision); ALTER TABLE IF EXISTS public.estacion
OWNER to facom ;ALTER TABLE IF EXISTS public.estacion ADD CONSTRAINT pf_municipio
FOREIGN KEY (cod_municipio) REFERENCES public.municipio (cod_municipio) MATCH SIMPLE;
ALTER TABLE IF EXISTS public.estacion ADD CONSTRAINT pf_zonahidrografica FOREIGN KEY (
cod_zonahidrografica) REFERENCES public.zonahidrografica (cod_zonahidrografica) MATCH
SIMPLE; ALTER TABLE IF EXISTS public.estacion ADD CONSTRAINT pf_categoria FOREIGN KEY (
cod_categoria) REFERENCES public.categoria (cod_categoria) MATCH SIMPLE; ALTER TABLE IF
EXISTS public.estacion ADD CONSTRAINT pf_tecnologia FOREIGN KEY (cod_tecnologia)
REFERENCES public.tecnologia (cod_tecnologia) MATCH SIMPLE; ALTER TABLE IF EXISTS public
.estacion ADD CONSTRAINT pf_estado FOREIGN KEY (cod_estado) REFERENCES public.estado (
cod_estado) MATCH SIMPLE;CREATE TABLE public.observacion ( cod_observacion bigserial
PRIMARY KEY, valor_observado double precision, fecha_observacion timestamp without time
zone , cod_estacion bigint, categoria_dato integer, cod_variable integer); ALTER TABLE
IF EXISTS public.observacion OWNER to facom; ALTER TABLE IF EXISTS public.observacion
ADD CONSTRAINT pf_estacion FOREIGN KEY (cod_estacion) REFERENCES public.estacion (
cod_estacion) MATCH SIMPLE; ALTER TABLE IF EXISTS public.observacion ADD CONSTRAINT
pf_variable FOREIGN KEY (cod_variable) REFERENCES public.variable (cod_variable) MATCH
SIMPLE;
```

Capítulo 6

¿Cómo ingresar la información de las tablas y los datos de temperatura y precipitación?

Este código permite ingresar la información completa de los datos de las variables de temperatura y precipitación del IDEAM, la información requerida inicialmente es descargar ambos conjuntos de datos desde `datos.gov.co`, aunque este paso se puede saltar si es modificado el archivo de actualización.py dispobible en el respositorio de github alejandria.

Links para descarga de datos de precipitación y temperatura con formato CSV.

Precipitación:
`https://www.datos.gov.co/Ambiente-y-Desarrollo-Sostenible/Precipitaci-n/s54a-sgyg/data`

Temperatura:
`https://www.datos.gov.co/Ambiente-y-Desarrollo-Sostenible/Datos-Hidrometeorol-gicos-Crudos-Red-de-Estaciones/sbwg-7ju4/data`

Ingresar información en las tablas

```
#-----------------------------------------------------------------#
#1. LIBRERÍAS
import pandas as pd
import numpy as np
from datetime import datetime
from tqdm import tqdm # libreria para saber el tiempo de ejecución
from sqlalchemy import create_engine
import re
import matplotlib.pyplot as plt #Para graficar
#-----------------------------------------------------------------#

#2. CREACIÓN DE VARIABLES DE NORMALIZACIÓN, MOTOR DE POSTGRES, DIRECCIONES, CONJUNTO DE
DATOS

#2.1 Bases de datos

#2.1.1 postgresql
eng = "postgresql://facom:usuario@localhost:5432/alejandria" #Motor
engine = create_engine(eng) #Máquina
conn=engine.connect()

#Anexo 1 para leer más sobre las bases de datos de sqlite usadas a continuación
#2.1.2 sqlite

T_sqlite= 'sqlite:////home/marcelae/Desktop/FACOM/2_db/temperatura_2.db' #lucy
P_sqlite= 'sqlite:////home/marcelae/Desktop/FACOM/2_db/precipitacion_2.db' #lucy

#2.2 Creacion de la conexión con la base de datos
Tengine_sql = create_engine(T_sqlite)  #Temperatura
Pengine_sql = create_engine(P_sqlite)  #Precipitacion
Tconn = Tengine_sql.connect()  #Temperatura
Pconn = Pengine_sql.connect()  #Precipitacion

#2.3 Variables normalizadoras
vnC=['Codigo', 'Nombre', 'Categoria', 'Tecnologia', 'Estado', 'Departamento',
       'Municipio', 'Ubicación', 'Altitud', 'Fecha_instalacion',
       'Fecha_suspension', 'Area Operativa', 'Corriente', 'Area Hidrografica',
       'Zona Hidrografica', 'Subzona hidrografica', 'Entidad','Latitud','Longitud',
       'calidad','fecha_llaveforanea']

vnCSV=["CodigoEstacion","CodigoSensor","FechaObservacion","ValorObservado",
       "NombreEstacion","Departamento","Municipio","ZonaHidrografica","Latitud",
       "Longitud","DescripcionSensor","UnidadMedida"]

vnBD=["nombre_categoria","nombre_tecnologia","nombre_estado",
      "nombre_departamento","nombre_zonahidrografica","nombre_municipio",
      "cod_departamento","cod_municipio","cod_zonahidrografica","cod_categoria",
      "cod_tecnologia","cod_estado","descripcion_variable","unidad_medida",
      "codigo_sensor", "cod_estacion","nombre_estacion","latitud","longitud",
      "altitud","fecha_observacion","cod_momento_observacion","valor_observado",
      "cod_estacion","calidad_dato","cod_variable"]

tablas=["departamento","municipio","zonahidrografica","categoria","tecnologia",
        "estado","momento_observacion","estacion","observacion","variable","flags",
        "flags_X_observacion","flags_x_estacion"]
```

```
#2.4 Direcciones
#2.4.1 lucy
d1=r"/home/marcelae/Desktop/FACOM/5_Documentos/Cat_logo_Nacional_de_Estaciones_del_IDEAM
.csv"
temp = r"/home/marcelae/Desktop/FACOM/3_csv/Datos_Hidrometeorol_gicos_Crudos_-
_Red_de_Estaciones_IDEAM___Temperatura.csv"
pre = r"/home/marcelae/Desktop/FACOM/3_csv/Precipitaci_n.csv"
pres= r"/home/marcelae/Desktop/FACOM/3_csv/Presi_n_Atmosf_rica.csv"
coor= r"/home/marcelae/Desktop/FACOM/1_Proyectos/2_Estaciones/CSV/coordenadas_estaciones
.csv"


#conjunto de datos
datos = pd.read_csv(d1)
#-----------------------------------------------------------------#
# 3. FUNCIONES
def lower(df):
    df_s=df.str.lower().str.replace('á','a').str.replace('é','e').str.replace('í','i').
str.replace('ó','o').str.replace('ú','u').str.replace('ñ','n')

    return(df_s)

def llave(eng,tabla_FK,cod_FK,data,data_FK,nombredb_FK):
    q = '''
    SELECT * FROM {};
    '''.format(tabla_FK)
    b = pd.read_sql(q,con=eng)
    #b = SQL_PD(q,eng)
    b.set_index(cod_FK,inplace = True)

    cod = []

    for i in data[data_FK]:
        for j in b[nombredb_FK]:
            if i == j:
                cod.append(b.index[b[nombredb_FK] == j][0])
                break
    return cod
#-----------------------------------------------------------------#
# 4. CORRECCIONES

#4.1 MAYÚSCULAS, MINÚSCULAS Y TILDES

datos[vnC[1]] = lower(datos[vnC[1]] ) #remover mayúsculas, vocales y ñ
datos[vnC[2]] = lower(datos[vnC[2]] ) #remover mayúsculas, vocales y ñ
datos[vnC[3]] = lower(datos[vnC[3]] ) #remover mayúsculas, vocales y ñ
datos[vnC[4]] = lower(datos[vnC[4]] ) #remover mayúsculas, vocales y ñ
datos[vnC[5]] = lower(datos[vnC[5]] ) #remover mayúsculas, vocales y ñ
datos[vnC[6]] = lower(datos[vnC[6]] ) #remover mayúsculas, vocales y ñ
datos[vnC[14]] = lower(datos[vnC[14]]) #remover mayúsculas, vocales y ñ

#4.2 COLUMNA NOMBRE ESTACIÓN
def nombres_cat(df):
    df=df.str.split('-',expand=True).drop([1,2], axis=1)
    df=pd.DataFrame(df[0].str.split('[',expand=True).drop([1], axis=1))
    return df

datos[vnC[1]] = nombres_cat(datos[vnC[1]])
```

```
#4.3 UBICACIÓN
def ubicacion(datos):
    ubicacion = datos[vnC[7]].str.replace('(','').str.replace(')','').str.split(',',
expand=True)
    c = pd.read_csv(coor,sep=";")
    # Reemplaza las coordenadas de las estaciones que están en los csv del IDEAM
    for i in tqdm(range(len(datos))):
        for j in range(len(c)):
            if datos[vnC[0]][i] == c.Codigo[j]:
                ubicacion[0][i] = c.Latitud[j]
                ubicacion[1][i] = c.Longitud[j]
    return ubicacion[0].astype(float),ubicacion[1].astype(float)

lat, lon = ubicacion(datos)
datos.insert(17,"Latitud",lat)
datos.insert(18,"Longitud",lon)
datos =  datos.drop(columns=["Ubicación"])

#4.4 ALTITUD
def Altitud(datos):
    datos[vnC[8]]=pd.DataFrame(datos[vnC[8]].str.replace(',','')).astype(float)

Altitud(datos)

#-------------------------------------------------------------------#
# 5.PROCESOS POR TABLA

#5.1 TABLA CATEGORÍA
def categoria(datos):
    ca = pd.DataFrame(datos[vnC[2]].unique(),columns = [vnBD[0]])
    ca = ca.sort_values(vnBD[0])
    ca.to_sql(tablas[3], engine, if_exists= "append",index=False)

#5.2 TABLA DEPARTAMENTO
def departamento(datos):
    dep = pd.DataFrame(datos[vnC[5]].unique(),columns = [vnBD[3]])
    dep = dep.sort_values(vnBD[3])
    dep.to_sql(tablas[0], engine, if_exists= "append",index=False)

#5.3 TABLA ESTADO
def estado(datos):
    es = pd.DataFrame(datos[vnC[4]].unique(),columns = [vnBD[2]])
    es = es.sort_values(vnBD[2])
    es.to_sql(tablas[5], engine, if_exists= "append",index=False)

#5.4 TABLA FLAGS

#5.5 TABLA MOMENTO OBSERVACION
def momento_observacion():
    mo=pd.DataFrame(pd.date_range(start="2000-01-01 00:00:00", end="2031-01-01 00:00:00"
, freq='min'),columns=["fecha_observacion"])
    mo.to_sql(tablas[6], con=engine, index=False, if_exists='append', chunksize=10000)

#5.6 TABLA TECNOLOGIA
def tecnologia(datos):
    tec = pd.DataFrame(datos[vnC[3]].unique(),columns = [vnBD[1]])
    tec = tec.sort_values(vnBD[1])
    tec.to_sql(tablas[4], engine, if_exists= "append",index=False)
```

```
#5.7 TABLA VARIABLE
def variable():
    temperatura = pd.read_csv(temp,nrows=1)
    precipitacion = pd.read_csv(pre,nrows=1)

    variable = pd.DataFrame(columns = [vnBD[12],vnBD[13],vnBD[14]])

    variable.descripcion_variable = [temperatura.DescripcionSensor[0],
                                     precipitacion.DescripcionSensor[0],
                                     "Presión Atmosferica (1h)"]

    variable.unidad_medida = [temperatura.UnidadMedida[0],
                              precipitacion.UnidadMedida[0],"HPa"]

    variable.codigo_sensor = [temperatura.CodigoSensor[0],
                              precipitacion.CodigoSensor[0],255]

    # Se añade el df a la tabla variable
    variable.to_sql(tablas[9], engine, if_exists= "append",index=False)

#5.8 TABLA ZONA HIDROGRÁFICA
def zonahidrografica(datos):
    zh = pd.DataFrame(datos[vnC[14]].unique(),columns = [vnBD[4]])
    zh = zh.sort_values(vnBD[4])
    zh.to_sql(tablas[2], engine, if_exists= "append",index=False)

#5.9 TABLA MUNICIPIO
def municipio(datos,eng):

    mun_cat = datos[[vnC[5],vnC[6]]]
    mun_cat = mun_cat.drop_duplicates(subset = vnC[6])
    mun_cat = mun_cat.sort_values(vnC[6])

    cod_mun = llave(eng,tablas[0],vnBD[6],mun_cat,vnC[5],vnBD[3])

    mun = pd.DataFrame(columns = [vnBD[6],vnBD[5]])
    mun.nombre_municipio = mun_cat[vnC[6]]
    mun.cod_departamento = cod_mun

    mun.to_sql(tablas[1], engine, if_exists= "append",index=False)

#5.10 TABLA ESTACION
def estacion(datos,eng):
    # Busqueda de Codigos

    cod_mun = llave(eng,tablas[1],vnBD[7],datos,vnC[6],vnBD[5])
    cod_zh = llave(eng,tablas[2],vnBD[8],datos,vnC[14],vnBD[4])
    cod_tec = llave(eng,tablas[4],vnBD[10],datos,vnC[3],vnBD[1])
    cod_est = llave(eng,tablas[5],vnBD[11],datos,vnC[4],vnBD[2])
    cod_cat = llave(eng,tablas[3],vnBD[9],datos,vnC[2],vnBD[0])

    estacion = pd.DataFrame(columns = [vnBD[15],vnBD[16],vnBD[17],vnBD[18],vnBD[7],
                                       vnBD[8],vnBD[9],vnBD[10],vnBD[11],vnBD[19]])
```

```
        estacion.cod_estacion = datos[vnC[0]]
        estacion.nombre_estacion = datos[vnC[1]]
        estacion.latitud = datos[vnC[17]]
        estacion.longitud = datos[vnC[18]]
        estacion.cod_municipio = cod_mun
        estacion.cod_zonahidrografica = cod_zh
        estacion.cod_categoria = cod_cat
        estacion.cod_tecnologia = cod_tec
        estacion.cod_estado = cod_est
        estacion.altitud = datos[vnC[8]]

        # Se añade el df a la tabla estacion
        estacion.to_sql(tablas[7], engine, if_exists= "append",index=False)

    #-------------------------------------------------------------------#
    # EJECUCION DE LAS FUNCIONES PARA AGREGAR TABLAS
    def añadirdb(catalogo,eng):
        departamento(catalogo)
        print("Se añadieron los datos a la tabla departamento")
        municipio(catalogo,eng)
        print("Se añadieron los datos a la tabla municipio")
        zonahidrografica(catalogo)
        print("Se añadieron los datos a la tabla zonahidrografica")
        categoria(catalogo)
        print("Se añadieron los datos a la tabla categoria")
        tecnologia(catalogo)
        print("Se añadieron los datos a la tabla tecnologia")
        estado(catalogo)
        print("Se añadieron los datos a la tabla estado")
        estacion(catalogo,eng)
        print("Se añadieron los datos a la tabla estacion")
        variable()
        print("Se añadieron los datos a la tabla variable")
    añadirdb(datos,eng)
```

Ingresar información de precipitación

```
    #-------------------------------------------------------------------#
    #5.11.1 PRECIPITACIÓN

    #longitud del archivo de entrada
    lp=pd.read_csv(pre,usecols=[0])
    n_p=len(lp)
    del lp

    step=math.ceil(n_p*0.05)  #el número es el porcentaje que se va a tomar "dx"
    cont=0 # el contador inicia desde 0, pero si es necesario se puede asignar uno diferente
    dx=0
    print("Longitud del archivo de entrada= ",n_p)
    print("Los pasos de tiempo son de= ",step)
    print("Inicia desde= ",cont)

    while tqdm(cont <= (n_p-1)):
        start= time.time()
        # La siguiente fila de código lo que carga es el dx, se toma una porción y
        # solo se carga.

        # El porcentaje que se desea cargar que inicialmente se asigno en cada paso.
        df=pd.read_csv(pre,nrows=int(step),skiprows=range(1,int(cont)),usecols=[0,2,3])
        print("#------#-------#")
        print("contador ",cont,"paso=",dx)
        print("#------#-------#")
        dx=dx+1
```

```
    df[vnCSV[2]]=pd.to_datetime(df[vnCSV[2]],format='%m/%d/%Y %I:%M:%S %p')
    df[vnCSV[2]] = df[vnCSV[2]].dt.floor('Min')
    df=df.sort_values(by=vnCSV[2]).reset_index(drop=True,inplace=False)
    df[vnC[19]]= np.zeros(len(df))
    df[vnC[20]]=np.zeros(len(df))
    n_df=len(df)

    # Categoría del dato
    for index, row in tqdm(df.iterrows()):

        if row[vnCSV[3]] < 0.0 or row[vnCSV[3]] >0.8:
            df[vnC[19]][index] = 1.0

    V=[]
    p=0

    for i in tqdm(range(p,n_df)):
        ab=df["CodigoEstacion"][i]
        if (ab==88112901 or ab==35237040 or ab==21202270
            or ab==35217080 or ab==35227020 or ab==23157050 or ab==52017020):
            continue

        if ab ==14015020:
            df[vnCSV[0]][i] = 14015080
        if ab==48015010:
            df[vnCSV[0]][i] = 48015050

        v =[df[vnCSV[3]][i],df[vnCSV[2]][i],df[vnCSV[0]][i],df[vnC[19]][i],2]
        V.append(v)

    V=pd.DataFrame(V)
    vnBD[25]
    V.columns=[vnBD[22], "fecha_observacion",vnBD[23],"categoria_dato",vnBD[25]]
    V.to_sql(tablas[8], con=engine, index=False, if_exists='append',chunksize=100000)
    cont=cont+step
    final= time.time()
    print("Tiempo de ejecución",final-start)
```

Ingresar información de temperatura

```
#5.11.2 TEMPERATURA

#longitud del archivo de entrada
lt =pd.read_csv(temp,usecols=[0])
n_t=len(lt)
del lt

step=math.ceil(n_t*0.05)  #el número es el porcentaje que se va a tomar "dx"
cont=0 # el contador inicia desde 0, pero si es necesario se puede asignar uno diferente
dx=0
print("Longitud del archivo de entrada= ",n_t)
print("Los pasos de tiempo son de= ",step)
print("Inicia desde= ",cont)
while tqdm(cont <= (n_t-1)):
    start= time.time()
    #La siguiente fila de código lo que carga es el dx, se toma una porción y solo se
carga
    #el porcentaje que se desea cargar que inicialmente se asignó en cada paso.
    df=pd.read_csv(temp,nrows=int(step),skiprows=range(1,int(cont)),usecols=[0,2,3])
    print("########################")
    print("#------#-------#")
    print("contador",cont,"paso=",dx)
    print("#------#-------#")
    df[vnCSV[2]]=pd.to_datetime(df[vnCSV[2]],format='%m/%d/%Y %I:%M:%S %p')
```

```
        df[vnCSV[2]] = df[vnCSV[2]].dt.floor('Min')
        df=df.sort_values(by=vnCSV[2]).reset_index(drop=True,inplace=False)
        df[vnC[19]]= np.zeros(len(df))
        df[vnC[20]]=np.zeros(len(df))
        n_df=len(df)
        print("Ingresa a calidad-",dx)
        print("#------#-------#")
        #Calidad del dato
        for index, row in tqdm(df.iterrows()):

            if row[vnCSV[3]] < 1.3 or row[vnCSV[3]] > 32.90:
                df[vnC[19]][index] = 1


        print("Ingresa a ingresar información-",dx)
        print("#------#-------#")
        #Ingreso de la información
        V=[]
        p=0
        for i in tqdm(range(p,n_df)):
            ab=df["CodigoEstacion"][i]
            if (ab==88112901 or ab==35237040 or ab==21202270
                or ab==35217080 or ab==35227020 or ab==23157050 or ab==52017020):
                continue

            if ab ==14015020:
                df[vnCSV[0]][i] = 14015080
            if ab==48015010:
                df[vnCSV[0]][i] = 48015050

            v =[df[vnCSV[3]][i],df[vnCSV[2]][i],df[vnCSV[0]][i],df[vnC[19]][i],2]
            V.append(v)

        V=pd.DataFrame(V)
        vnBD[25]
        V.columns=[vnBD[22], "fecha_observacion",vnBD[23],"categoria_dato",vnBD[25]]
        V.to_sql(tablas[8], con=engine, index=False, if_exists='append',chunksize=100000)
        cont=cont+step
        final= time.time()
        print("Termina-",dx)
        print("#------#-------#")

        dx=dx+1
        print("Tiempo de ejecución",final-start)
        print("#------#-------#")
        print("######################")
```

Las siguientes tablas no contienen datos, sin embargo, el ejercicio queda abierto a incluir información en el futuro.

- TABLA FLAGS X ESTACION
- TABLA FLAGS X OBSERVACION
- TABLA FLAGS

Capítulo 7

¿Cómo actualizar la base de datos con crontab?

Para mantener la base de datos actualizada se usa la herramienta contab -e.

Paso 1:
Debe decidir cada cuánto tiempo desea actualizar la base de datos. Para ello se utliza una secuencia de 5 símbolos y/o números separados por un espacio (en este orden: minuto - hora - día del mes - mes - dia de la semana). En esta página puede encontrar más información acerca de los símbolos y sus significados, así como verificar si su comando está correcto `https://crontab.guru/`. A continuación un par de ejemplos:

- Si usted desea actualizar alejandría una vez al día, a las 1:00, debe ingresar este parámetro (esta es la secuencia recomendada):

  0 1 * * *

- Si usted desea actualizar alejandría una vez a la semana, el día martes, a las 01:36, debe ingresar este parámetro:

  36 1 * * 2

- Si usted desea actualizar alejandría una vez al mes, el día 15, a las 13:10, debe ingresar este parámetro:

  10 13 15 * *

Paso 2:
Ingresar a la terminal de ubuntu e ingresar el comando:

```
crontab -e
```

Paso 3: Al realizar el paso 2, se abrirá un archivo de texto, allí debe escribir al final de las líneas comentadas la secuencia del paso 1 seguido de la dirección donde está ubicado la versión de python que se va a usar (debe conocerla en su máquina local) y la dirección donde está ubicado el archivo actualizar.py incluido en el github del proyecto alejandría. La línea se escribe en este orden:

```
10 5 15 * * (dirección de la versión de python) (ubicación del archivo actualización.py)
```

Capítulo 8

Ejemplos de uso

Luego de crear la base de datos alejandría o acceder por medio de una terminal, es necesario conocer cómo moverse en ella, por ello hemos diseñado una serie de ejemplos de uso para guíar a quienes desean hacer búsquedas, sin embargo, se recomienda tener conocimientos básicos en lenguaje SQL.

8.1. Usando python.

Lo primero que debe hacer es digitar por consola la siguiente instrucción para abrir un archivo de texto de python (también puede abrir un editor de texto para compilar lenguaje python y saltarse este paso).

```
python3
```

Ahora, debe digitar las siguientes líneas de código en el orden indicado

```
import pandas as pd
eng = "postgresql://lucy:usuario@localhost:5432/alejandria"
```

Luego, debe conocer la instrucción de SQL para hacer la búsqueda, esta se llamará 'query'. Basta con digitar la siguiente línea de código para recibir la información deseada en formato DataFrame de pandas.

```
pd.read_sql(query,con=eng)
```

8.2. Usando la interfaz de pgAdmin.

Otra forma de buscar información en la base de datos de alejandría es a través de la interfaz de pgAdmin, luego de tener la herramienta abierta se debe hacer click derecho sobre la base de datos y seleccionar"Query Tools", allí verá un recuadro de texto donde deberá ingresar un 'query' en lenguaje SQL luego, en el menú superior de su pantalla debe dar click en el botón de búsqueda para obtener el resultado.

8.3. Usando pgAdmin por consola.

Para acceder a alejandría directamente desde pgAdmin por consola debe iniciar digitando el siguiente comando.

```
sudo su postgres
```

Seguido de esto debe escribir la contraseña del equipo que esté usando para acceder al servidor y luego, digitar en este orden las siguientes líneas:

```
psql
\c alejandria
```

Ya está conectado a la base de datos por consola, lo que resta es escribir su 'query' con la búsqueda que desee (usando este método debe terminar la búsqueda con el símbolo ; al final de cada query).

8.4. Ejemplos de 'query'

A continuación presentamos algunos ejemplos del 'query' que debe añadir en cada uno de los métodos descritos anteriormente para realizar búsquedas en Alejandría.

Ejemplo 1:

Se requiere un listado de la altitud de las estaciones meteorológicas del IDEAM con su respectivo código de estación y nombre de estación.

```
query = "SELECT codigo_estacion,nombre_estacion,altitud FROM estacion"
```

Debe obtener:

```
      cod_estacion            nombre_estacion  altitud
0         52057100                 rumichaca    2582.0
1         52055170              la josefina    2450.0
2         52055220               el paraiso    3120.0
3         44015070                 el pepino     760.0
4         48015040              puerto narino     158.0
...            ...                       ...       ...
8968    4401700167  nivel sangoyaco garganta    761.0
8969    4401700168     nivel mulato palmeras    966.0
8970    4401700172     nivel mocoa piscikart    408.0
8971    4401700173    nivel rumiyaco lagarto    647.0
8972    4401700174           nivel rio pepino    868.0
```

Ejemplo 2:

Se desean obtener los diferentes municipios con su respectivo departamento en los que alejandria contiene datos, query debe ser:

```
query = '''
SELECT DISTINCT nombre_municipio,nombre_departamento
FROM municipio
INNER JOIN departamento USING(cod_departamento)
'''
```

Debe obtener lo siguiente:

```
         nombre_municipio   nombre_departamento
0               el cerrito       valle del cauca
1     la jagua de ibirico                 cesar
2       puerto libertador               cordoba
3                  betania             antioquia
4                pinillos               bolivar
...                   ...                   ...
1021                 tona             santander
1022          jesus maria             santander
1023                neira                caldas
1024                manta          cundinamarca
1025            el reten                magdalena
```

Ejemplo 3:

Buscar la todos los valores de temperatura que existan en la base de datos en cualquier fecha para la estación de código 48015040 junto con su nombre y municipio.

```
query = '''
SELECT cod_estacion,nombre_estacion,fecha_observacion,valor_observado,nombre_municipio
FROM observacion
INNER JOIN estacion USING(cod_estacion)
INNER JOIN variable USING (cod_variable)
INNER JOIN municipio USING (cod_municipio)
WHERE cod_estacion=48015040 AND cod_variable = 1
'''
```

Con esta búsqueda obtendrá:

	cod_estacion	nombre_estacion	fecha_observacion	valor_observado	nombre_municipio
0	48015040	puerto narino	2007-06-24 18:00:00	25.9	puerto narino
1	48015040	puerto narino	2007-06-25 22:00:00	23.9	puerto narino
2	48015040	puerto narino	2007-06-25 23:00:00	23.6	puerto narino
3	48015040	puerto narino	2007-06-27 00:00:00	23.7	puerto narino
4	48015040	puerto narino	2007-06-27 21:00:00	23.6	puerto narino
...	...	...	...	...	...
81945	48015040	puerto narino	2007-06-22 02:00:00	22.9	puerto narino
81946	48015040	puerto narino	2007-06-23 19:00:00	24.4	puerto narino
81947	48015040	puerto narino	2007-06-23 21:00:00	23.8	puerto narino
81948	48015040	puerto narino	2007-06-24 01:00:00	23.1	puerto narino
81949	48015040	puerto narino	2007-06-24 06:00:00	22.8	puerto narino

Ejemplo 4:

Buscar los códigos de estación asociados a estaciones que esten a una altura mayor a los 1500 msnm.

```
SELECT cod_estacion
FROM estacion
WHERE altitud > 1500
```

Ejemplo 5:

Seleccionar los datos de precipitación de estación 26185501 donde los valores esten en categoria 0 y la información este entre 2020 y junio de 2022.

```
SELECT valor_observado,fecha_observacion
FROM observacion
WHERE cod_variable=2
AND cod_estacion =26185501
AND categoria_dato <> 1
AND fecha_observacion < '2022-06-30 23:59:00'
AND fecha_observacion > '2019-12-31 23:59:00'
```

Ejemplo 6:

Seleccionar los datos de precipitación (fecha,valor observado) de la estación 26185501 que sean mayores a 10.0 mm.

```
SELECT valor_observado, fecha_observacion
FROM observacion
WHERE cod_variable=2
AND cod_estacion =26185501
AND categoria_dato <> 1
AND valor_observado > 10.0
```

Ejemplo 7: SubQuerys

Buscar todas las estaciones ubicadas por encima de los 1500 msnm y además que tengan información de precipitación y temperatura.

```
SELECT cod_estacion,altitud,latitud,longitud
FROM estacion
WHERE estacion.altitud > 1500

-- Lo que entrega cod_estacion debe estar incluido debe estar incluido en las
-- dos listas que entregan los dos subquerys
AND cod_estacion
IN ( --seleccion de todas las estaciones que tienen temperatura
    SELECT estacion.cod_estacion
    FROM estacion
    INNER JOIN observacion
    ON estacion.cod_estacion=observacion.cod_estacion
    WHERE observacion.cod_variable = 1
    GROUP BY estacion.cod_estacion)
AND cod_estacion
IN(--seleccion de todas las estaciones que tienen precipitacion
    SELECT estacion.cod_estacion FROM estacion
    INNER JOIN observacion ON estacion.cod_estacion=observacion.cod_estacion
    WHERE observacion.cod_variable = 2
    GROUP BY estacion.cod_estacion)
```