

## Unidad 3 - Tarea 4 - Almacenamiento y Consultas de Datos en Big Data

### **Nombre del estudiante**

Giovanny Alejandro Pardo

### **Grupo:**

Big Data (202016911\_27)

### **Tutora**

Sandra Milena Patino Avella

Universidad Nacional Abierta y a Distancia-UNAD

Escuela de ciencias básicas, tecnológicas e ingeniería

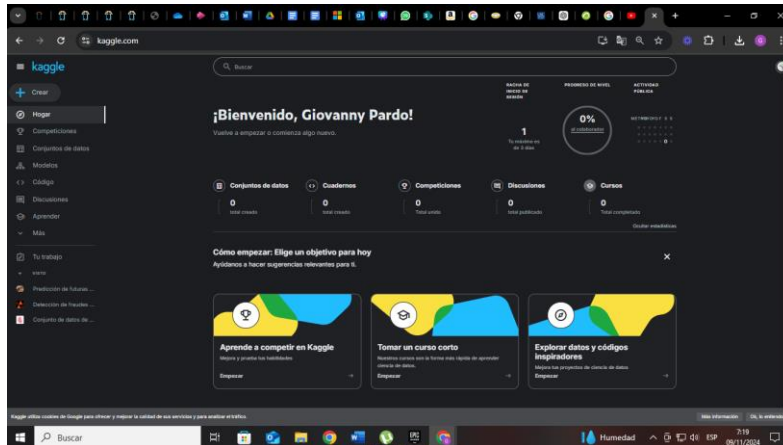
Ingeniería de sistemas

Palmira – noviembre 10 del 2024

## Fase 2: Apache Hbase

Para el desarrollo de esta actividad nos dirigiremos a la página kaggle para descargar un dataset de información de productos para usarla en la base de datos HBase y realizar las respectivas consultas.

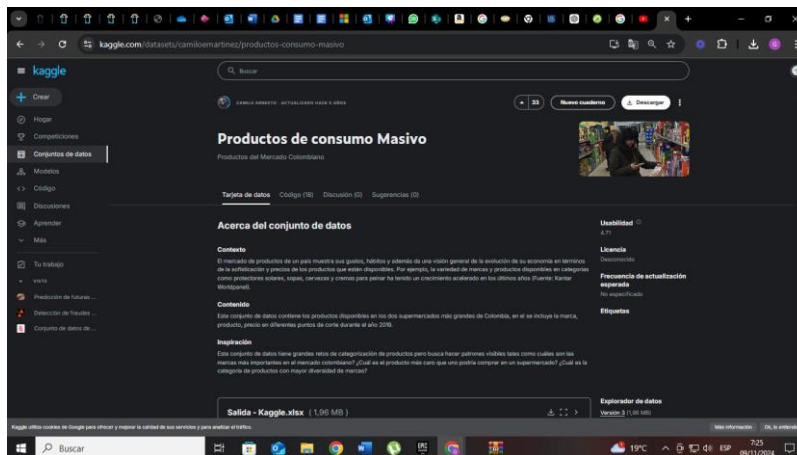
Imagen 01



Nota: En esta imagen se evidencia que estamos en la página oficial de kaggle.

Para nuestro caso usaremos un dataset de productos de consumo masivo el dataset cuenta con 25638 registros y 13 tipos de datos.

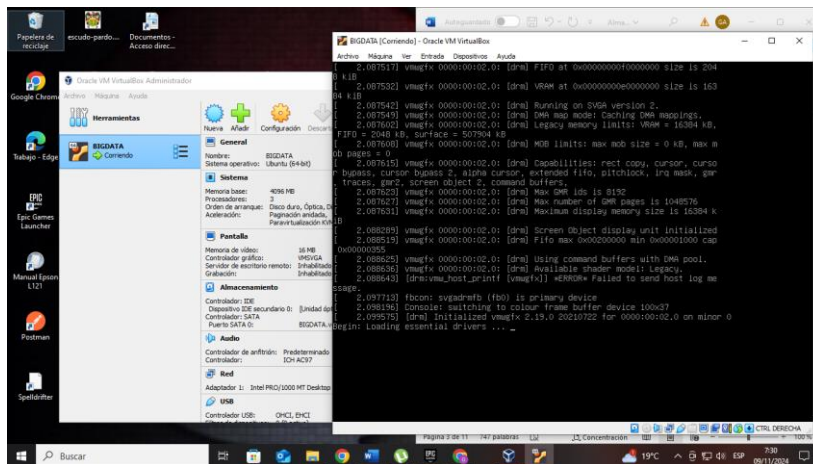
Imagen 02



Nota: En esta imagen se evidencia el dataset a usar.

Ahora con el dataset descargado procedemos a abrir nuestra máquina virtual y arrancar nuestro servidor Ubuntu Serer.

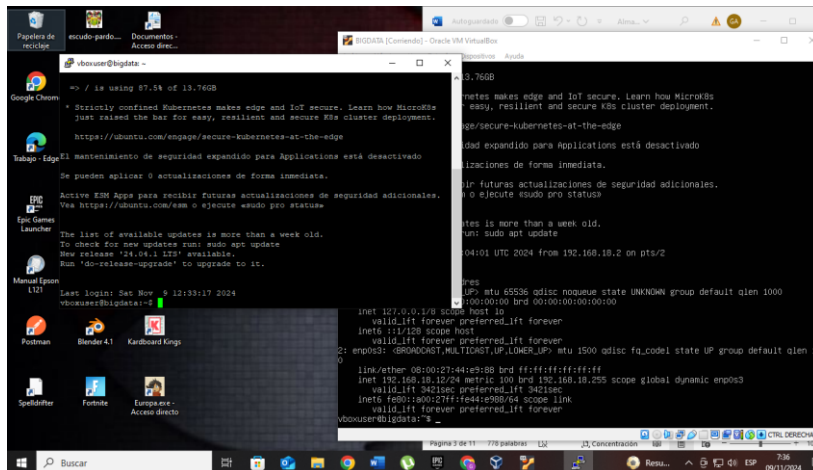
Imagen 03



Nota: En esta imagen se evidencia el arranque de nuestro Servidor Ubuntu.

Ahora procedemos a conectarnos a nuestro servidor Ubuntu desde nuestro equipo mediante la aplicación Putty agregando nuestra IP al Servidor Ubuntu y ingresando las credenciales.

Imagen 04

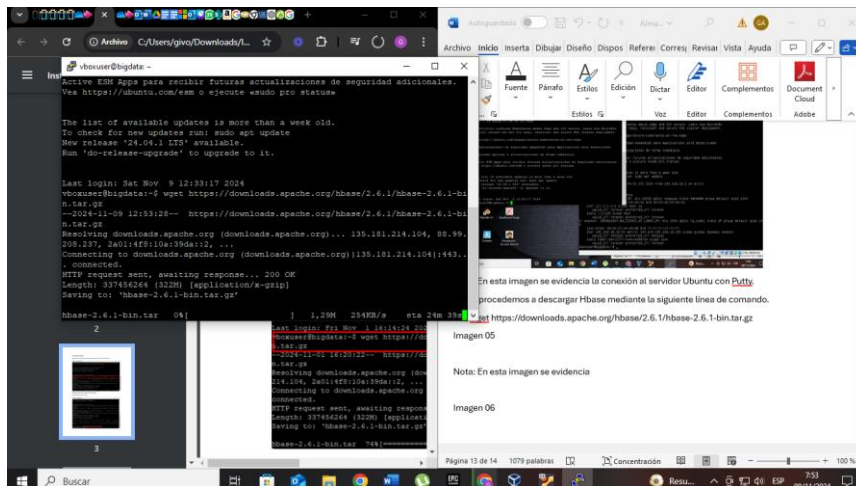


Nota: En esta imagen se evidencia la conexión al servidor Ubuntu con Putty.

Ahora procedemos a descargar Hbase mediante la siguiente línea de comando.

>>\$ wget <https://downloads.apache.org/hbase/2.6.1/hbase-2.6.1-bin.tar.gz>

## Imagen 05



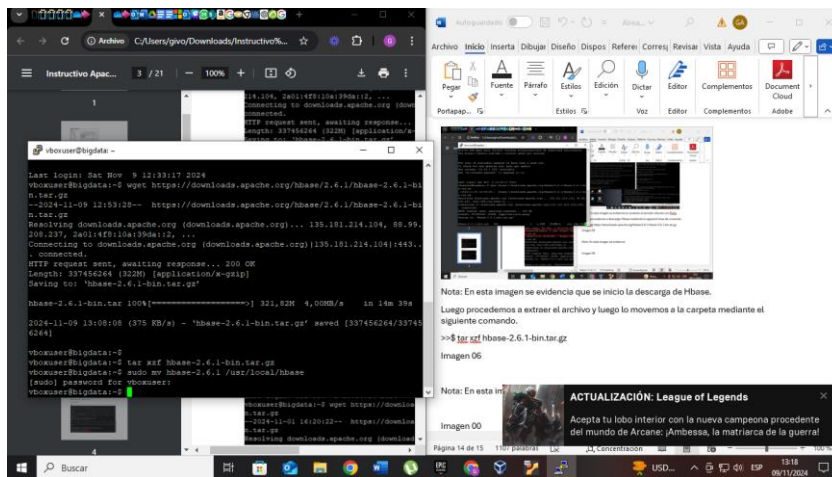
Nota: En esta imagen se evidencia que se inicio la descarga de Hbase.

Luego procedemos a extraer el archivo y luego lo movemos a la carpeta mediante los siguientes comandos.

```
>>$ tar xzf hbase-2.6.1-bin.tar.gz
```

```
>>$ sudo mv hbase-2.6.1 /usr/local/hbase
```

## Imagen 06



Nota: En esta imagen se evidencia la extracción del archivo y luego se mueve al directorio /usr/local/hbase

Ahora procedemos a configurar las variables de entorno, para ello editamos el archivo **bashrc** con nano mediante la siguiente línea de comando.

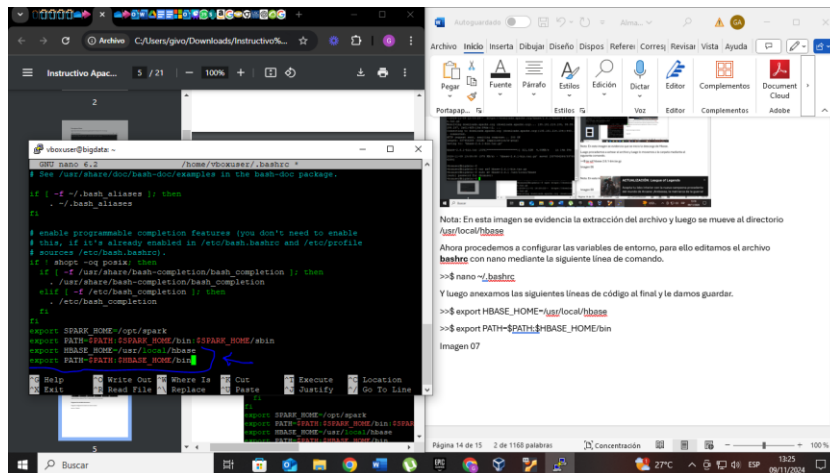
```
>>$ nano ~/.bashrc
```

Y luego anexamos las siguientes líneas de código al final y le damos guardar.

```
>>$ export HBASE_HOME=/usr/local/hbase
```

```
>>$ export PATH=$PATH:$HBASE_HOME/bin
```

Imagen 07



Nota: En esta imagen se evidencia que agregamos las líneas de código al final del archivo.

Ahora se procede a cargar las variables de entorno mediante la siguiente línea de comando.

```
>>$ source ~/.bashrc
```

Luego procedemos a editar el archivo **hbase-site.xml** para configurarlo en modo local, y esto se logra mediante la siguiente línea de código.

```
>>$ sudo nano /usr/local/hbase/conf/hbase-site.xml
```

Luego cuando se abra el archivo anexamos el siguiente código:

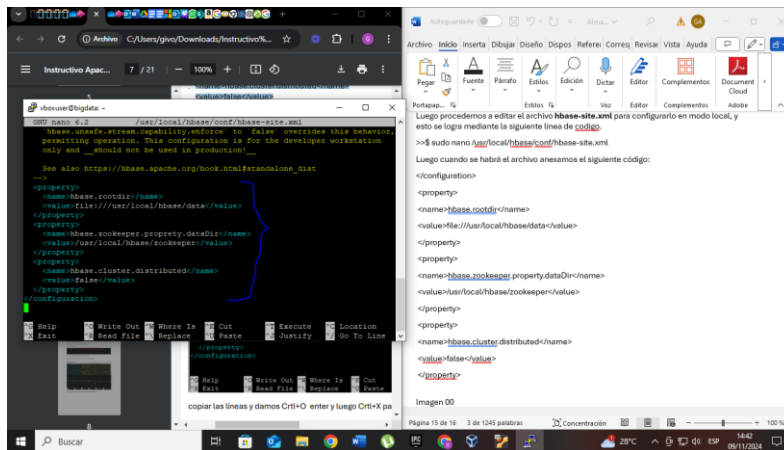
```
</configuration>

<property>
<name>hbase.rootdir</name>
<value>file:///usr/local/hbase/data</value>
</property>

<property>
<name>hbase.zookeeper.property.dataDir</name>
<value>/usr/local/hbase/zookeeper</value>
</property>

<property>
<name>hbase.cluster.distributed</name>
<value>>false</value>
</property>
```

Imagen 08



Nota: En esta imagen se evidencia la sección de código agregada al archivo **hbase-site.xml**

Ahora editamos el archivo `hbase-env.sh` para ello lo abrimos con nano mediante la siguiente línea de código.

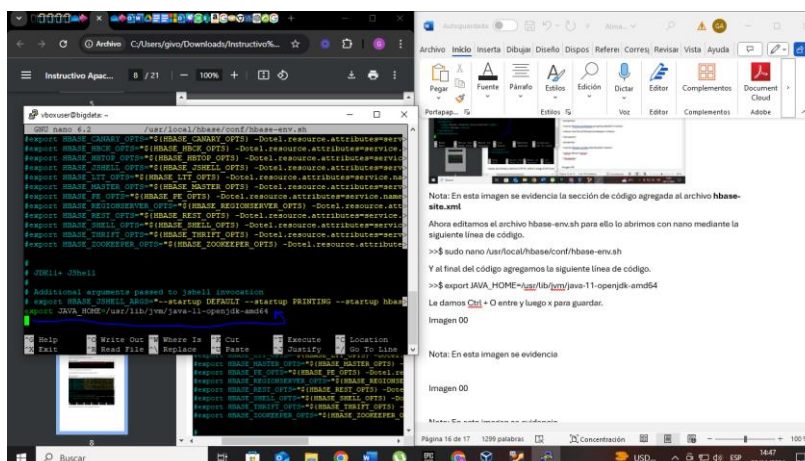
```
>>$ sudo nano /usr/local/hbase/conf/hbase-env.sh
```

Y al final del código agregamos la siguiente línea de código.

```
>>$ export JAVA_HOME=/usr/lib/jvm/java-11-openjdk-amd64
```

Le damos Ctrl + O entre y luego x para guardar.

Imagen 09



Nota: En esta imagen se evidencia cuando se agrega la línea de comando al archivo **hbase-env.sh**

Terminado todas las configuraciones previas, procedemos a iniciar Hbase mediante la siguiente línea de comando.

```
>>$ start-hbase.sh
```

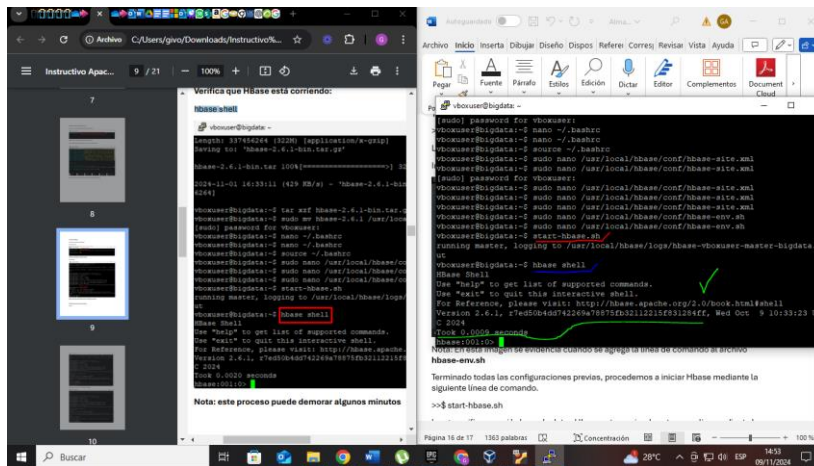
Luego verificamos si la base de datos Hbase esta corriendo esto se realiza mediante la siguiente línea de comando.

```
>>$ hbase Shell
```



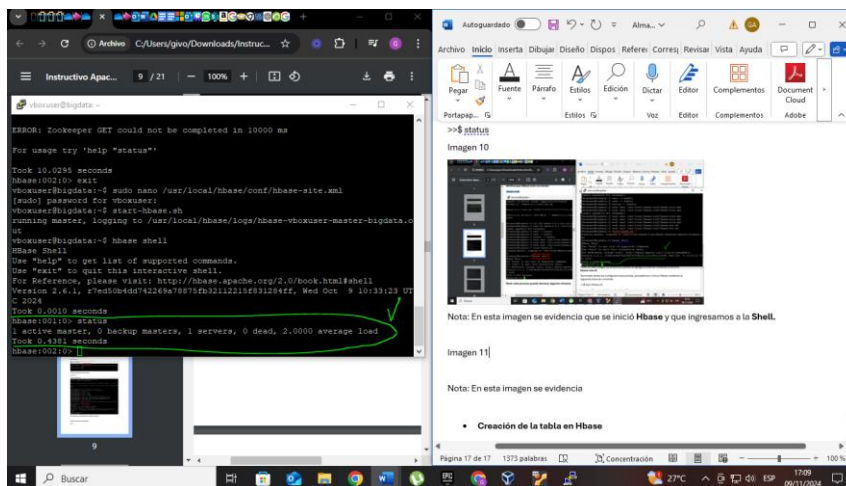
```
>>$ status
```

Imagen 10



Nota: En esta imagen se evidencia que se inició **Hbase** y que ingresamos a la **Shell**.

Imagen 11

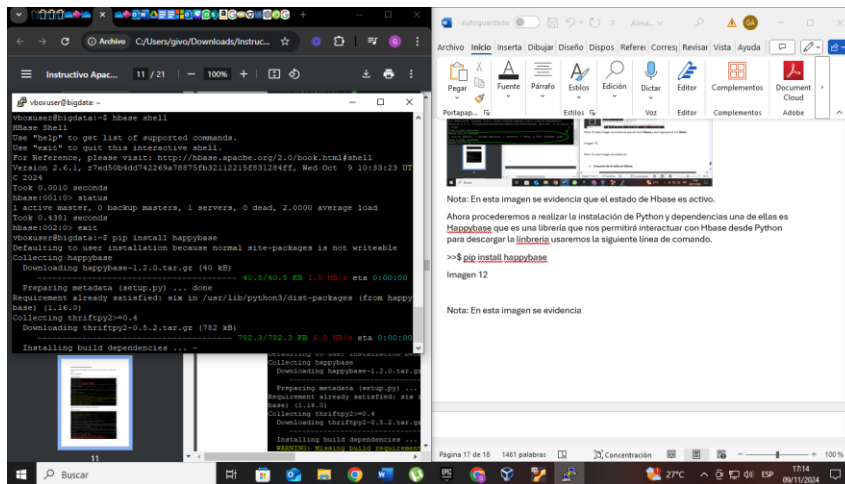


Nota: En esta imagen se evidencia que el estado de Hbase es activo.

Ahora procederemos a realizar la instalación de Python y dependencias una de ellas es Happybase que es una librería que nos permitirá interactuar con Hbase desde Python para descargar la librería usaremos la siguiente línea de comando.

```
>>$ pip install happybase
```

Imagen 12

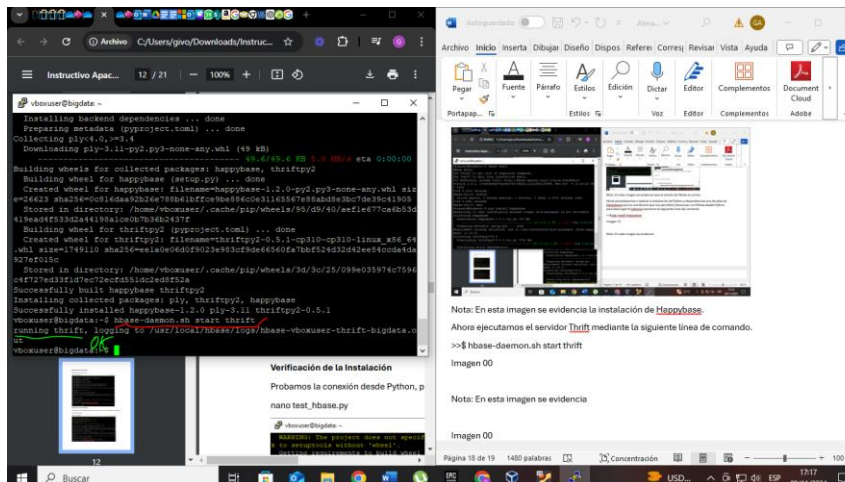


Nota: En esta imagen se evidencia la instalación de Happybase.

Ahora ejecutamos el servidor Thrift mediante la siguiente línea de comando.

```
>>$ hbase-daemon.sh start thrift
```

Imagen 13



Nota: En esta imagen se evidencia que arranco el servidor Thrift.

Ahora procederemos a verificar las instalaciones y para ello probaremos las conexiones desde Python mediante la creación de un archivo de prueba con las siguientes instrucciones.

```
>>$ nano test_hbase.py
```

Y en el archivo agregaremos las siguientes instrucciones o líneas de código que tiene como propósito de conectarse Hbase para verificar la tabla existente si hay tenderemos una respuesta de éxito de conexión, de lo contrario tendremos un mensaje de error.

```
import happybase
```

```
try:
```

```
# Intentar establecer conexión
```



```

connection = happybase.Connection('localhost')

# Listar las tablas existentes

print("Tablas existentes:", connection.tables())

print("Conexión exitosa a HBase")

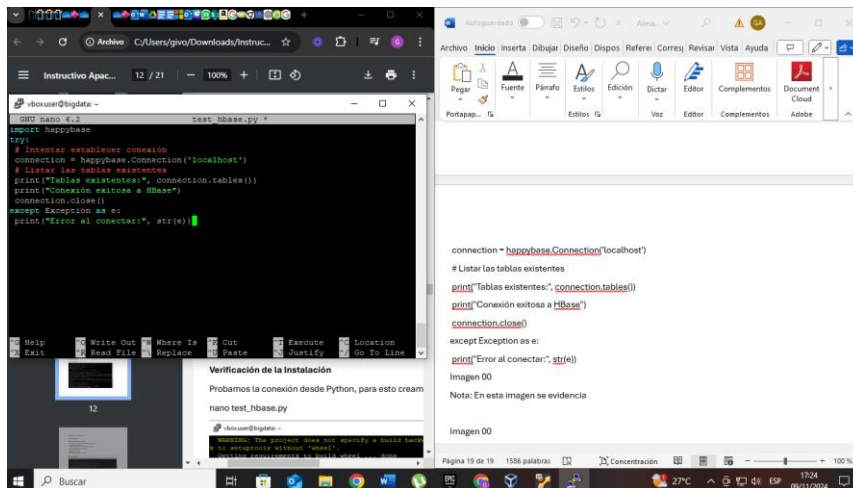
connection.close()

except Exception as e:

    print("Error al conectar:", str(e))

```

Imagen 14

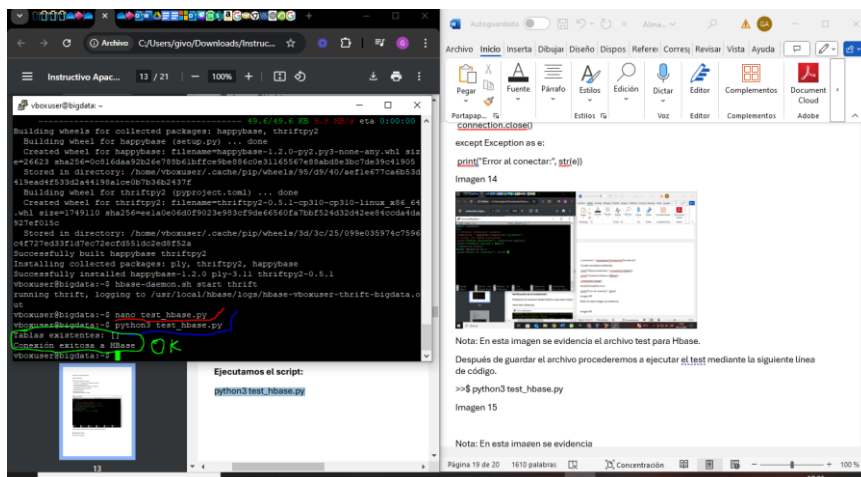


Nota: En esta imagen se evidencia el archivo test para Hbase.

Después de guardar el archivo procederemos a ejecutar el test mediante la siguiente línea de código.

```
>>$ python3 test_hbase.py
```

Imagen 15

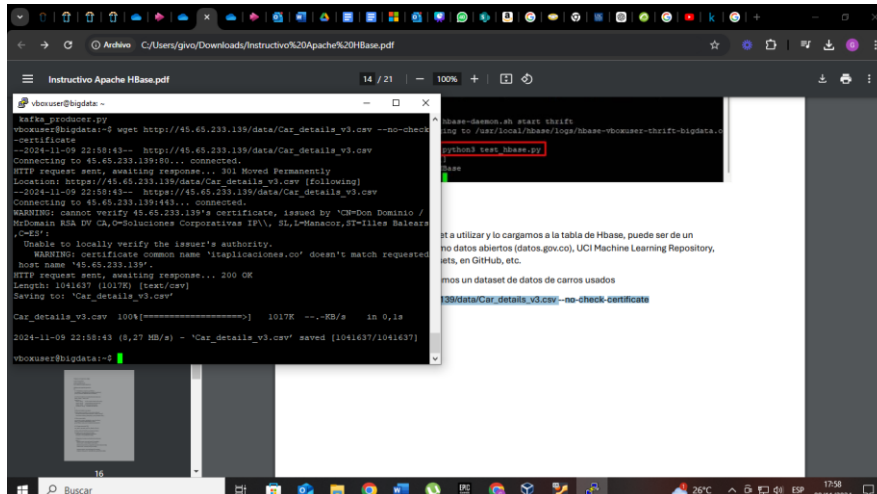


Nota: En esta imagen se evidencia que la conexión a Hbase fue exitosa con el resultado de [] en tablas ósea cero.

Ahora debemos descargar el dataset de pruebas para ello usaremos la siguiente línea de comando.

```
>>$ wget http://45.65.233.139/data/Car_details_v3.csv --no-check-certificate
```

Imagen 16



Nota: En esta imagen se evidencia que se descargo de manera satisfactoria en dataset.

Ahora realizaremos el archivo que se conectara con Hbase, creara una tabla con sus respectivas familias y se realizaran unas consultas para ello abrimos un archivo con nano al cual nombraremos consultas.py y lo realizaremos mediante la siguiente línea de comando.

```
>>$ nano consultas.py
```

Luego ingresamos este bloque de código.

```
import happybase

import pandas as pd

from datetime import datetime

# Bloque principal de ejecución
try:

    # 1. Establecer conexión con HBase
    connection = happybase.Connection('localhost')
    print("Conexión establecida con HBase")

    # 2. Crear la tabla con las familias de columnas
    table_name = 'used_cars'

    families = {
        'basic': dict(), # información básica del coche
        'specs': dict(), # especificaciones técnicas
        'sales': dict(), # información de venta
```

```

'condition': dict() # estado del vehículo
}
# Eliminar la tabla si ya existe
if table_name.encode() in connection.tables():
print(f"Eliminando tabla existente - {table_name}")
connection.delete_table(table_name, disable=True)
# Crear nueva tabla
connection.create_table(table_name, families)
table = connection.table(table_name)
print("Tabla 'used_cars' creada exitosamente")
# 3. Cargar datos del CSV
car_data = pd.read_csv('Car_details_v3.csv')

# Iterar sobre el DataFrame usando el índice
for index, row in car_data.iterrows():
# Generar row key basado en el índice
row_key = f'car_{index}'.encode()

# Organizar los datos en familias de columnas
data = {
b'basic:name': str(row['name']).encode(),
b'basic:year': str(row['year']).encode(),
b'basic:transmission': str(row['transmission']).encode(),
b'basic:fuel': str(row['fuel']).encode(),

b'specs:engine': str(row['engine']).encode(),
b'specs:max_power': str(row['max_power']).encode(),
b'specs:torque': str(row['torque']).encode(),
b'specs:seats': str(row['seats']).encode(),
b'specs:mileage': str(row['mileage']).encode(),

b'sales:selling_price': str(row['selling_price']).encode(),
b'sales:seller_type': str(row['seller_type']).encode(),

```

```
b'condition:km_driven': str(row['km_driven']).encode(),
b'condition:owner': str(row['owner']).encode()
}
```

```
table.put(row_key, data)
```

```
print("Datos cargados exitosamente")
```

```
# 4. Consultas y Análisis de Datos
```

```
print("\n=== Todos los coches en la base de datos (primeros 3) ===")
```

```
count = 0
```

```
for key, data in table.scan():
```

```
if count < 3: # Limitamos a 3 para el ejemplo
```

```
print(f"\nCoche ID: {key.decode()}")
```

```
print(f"Nombre: {data[b'basic:name'].decode()}")
```

```
print(f"Año: {data[b'basic:year'].decode()}")
```

```
print(f"Precio: {data[b'sales:selling_price'].decode()}")
```

```
count += 1
```

```
# 6. Encontrar coches por rango de precio
```

```
print("\n=== Coches con precio menor a 50000 ===")
```

```
for key, data in table.scan():
```

```
if int(data[b'sales:selling_price'].decode()) < 50000:
```

```
print(f"\nCoche ID: {key.decode()}")
```

```
print(f"Nombre: {data[b'basic:name'].decode()}")
```

```
print(f"Precio: {data[b'sales:selling_price'].decode()}")
```

```
# 7. Análisis de propietarios
```

```
print("\n=== Coches por tipo de propietario ===")
```

```
owner_stats = {}
```

```
for key, data in table.scan():
```

```
owner = data[b'condition:owner'].decode()
```

```
owner_stats[owner] = owner_stats.get(owner, 0) + 1
```

```
for owner, count in owner_stats.items():
```

```

print(f"{owner}: {count} coches")

# 8. Análisis de precios por tipo de combustible
print("\n=== Precio promedio por tipo de combustible ===")
fuel_prices = {}
fuel_counts = {}

for key, data in table.scan():
    fuel = data[b'basic:fuel'].decode()
    price = int(data[b'sales:selling_price'].decode())

    fuel_prices[fuel] = fuel_prices.get(fuel, 0) + price
    fuel_counts[fuel] = fuel_counts.get(fuel, 0) + 1

for fuel in fuel_prices:
    avg_price = fuel_prices[fuel] / fuel_counts[fuel]
    print(f"{fuel}: {avg_price:.2f}")

# 9. Top 3 coches con mayor kilometraje
print("\n=== Top 3 coches con mayor kilometraje ===")
cars_by_km = []
for key, data in table.scan():
    cars_by_km.append({
        'id': key.decode(),
        'name': data[b'basic:name'].decode(),
        'km': int(data[b'condition:km_driven'].decode()),
        'price': int(data[b'sales:selling_price'].decode())
    })

for car in sorted(cars_by_km, key=lambda x: x['km'],
reverse=True)[:3]:
    print(f"ID: {car['id']}")
    print(f"Nombre: {car['name']}")
    print(f"Kilometraje: {car['km']}")
    print(f"Precio: {car['price']}\n")

# 10. Análisis de precios por tipo de transmisión

```

```

print("\n=== Precio promedio por tipo de transmisión ===")
transmission_prices = {}
transmission_counts = {}

for key, data in table.scan():
    trans = data[b'basic:transmission'].decode()
    price = int(data[b'sales:selling_price'].decode())

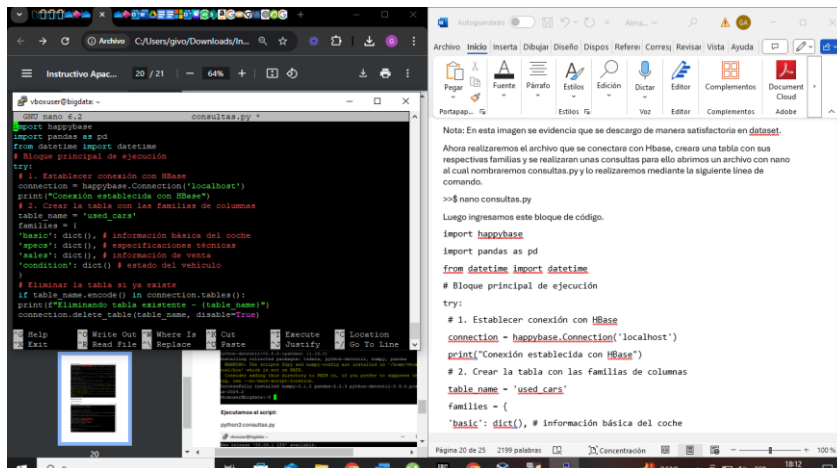
    transmission_prices[trans] = transmission_prices.get(trans, 0) + price
    transmission_counts[trans] = transmission_counts.get(trans, 0) + 1

for trans in transmission_prices:
    avg_price = transmission_prices[trans] / transmission_counts[trans]
    print(f"{trans}: {avg_price:.2f}")
# 11. Ejemplo de actualización de precio
car_to_update = 'car_0'
new_price = 460000
table.put(car_to_update.encode(), {b'sales:selling_price':
str(new_price).encode()})
print(f"\nPrecio actualizado para el coche ID: {car_to_update}")
except Exception as e:
    print(f"Error: {str(e)}")
finally:
    # Cerrar la conexión
    connection.close()

```



Imagen 17



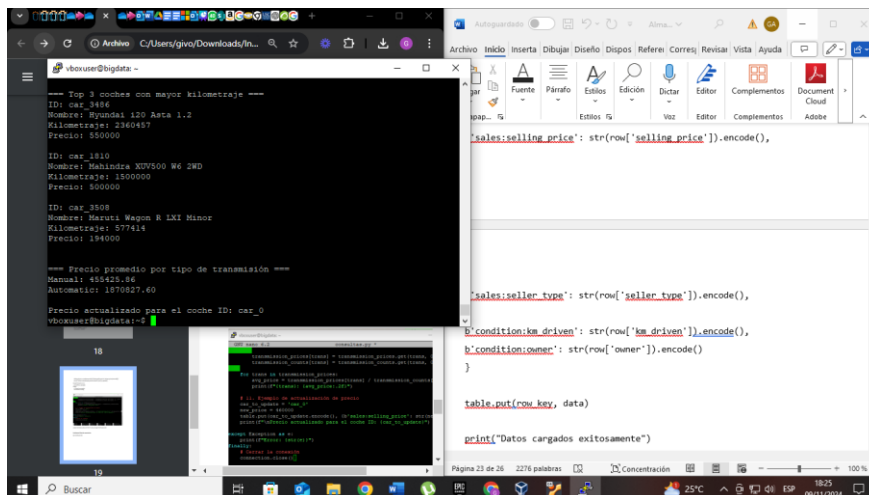
Nota: En esta imagen se evidencia la creación del archivo **consultas.py**

Ahora procedemos a correr el archivo consultas.py mediante la siguiente liena de código.

```
>>$ python3 consultas.py
```

Nota: nos salió un error y fue porque no se tubo encuentra la indentación en el código Python que es muy estricto se ingresa al archivo y se realiza la correcta indentación se guardó y se ejecuta de nuevo y se obtuvo los resultados esperados.

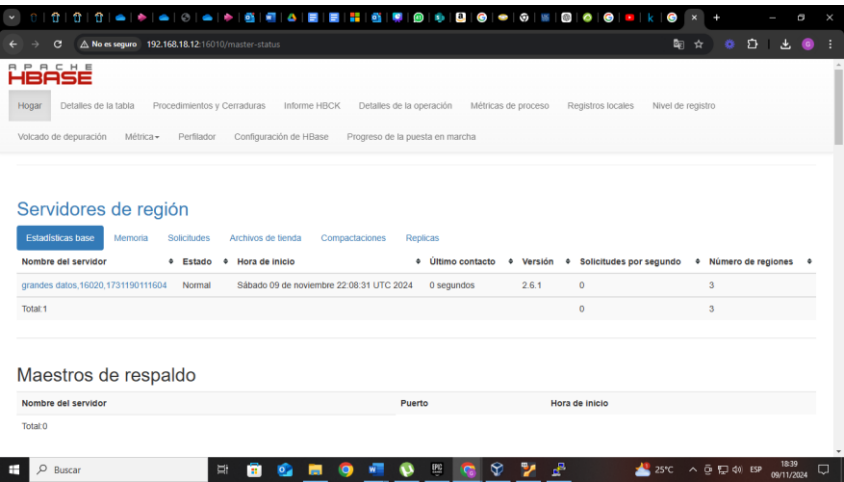
Imagen 18



Nota: En esta imagen se evidencia los resultados de las consultas.

También podemos monitorear el servidor Hbase ingresando por la URL [http://ip\\_maquina\\_virtual:16010](http://ip_maquina_virtual:16010)

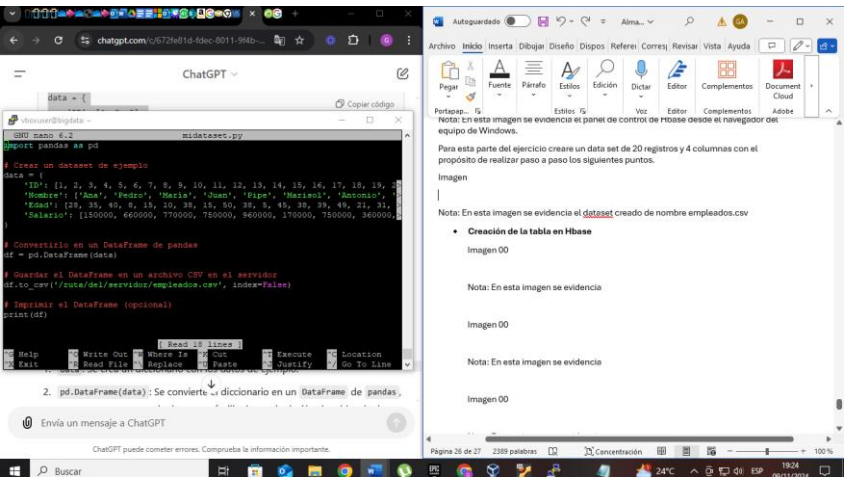
Imagen 19



Nota: En esta imagen se evidencia el panel de control de Hbase desde el navegador del equipo de Windows.

Para esta parte del ejercicio creare un data set de 20 registros y 4 columnas con el propósito de realizar paso a paso los siguientes puntos.

Imagen 20



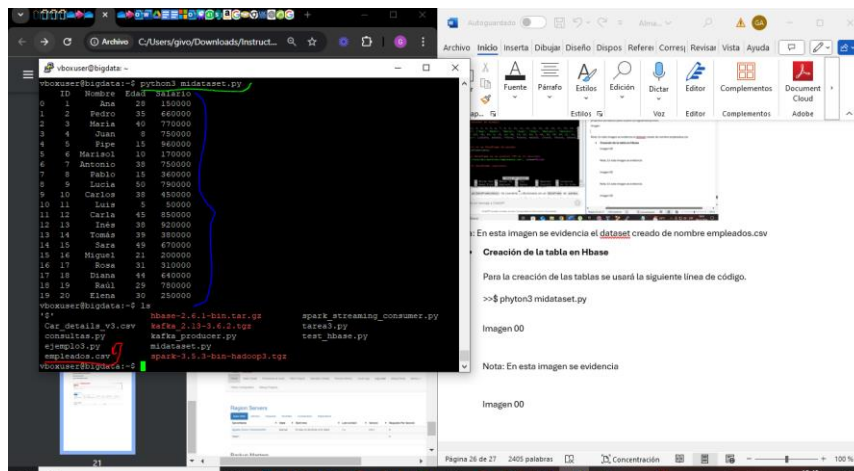
Nota: En esta imagen se evidencia el dataset creado de nombre empleados.csv

- **Creación de la tabla en Hbase**

Para la creación de las tablas se usará la siguiente línea de código.

>>\$ python3 midataset.py

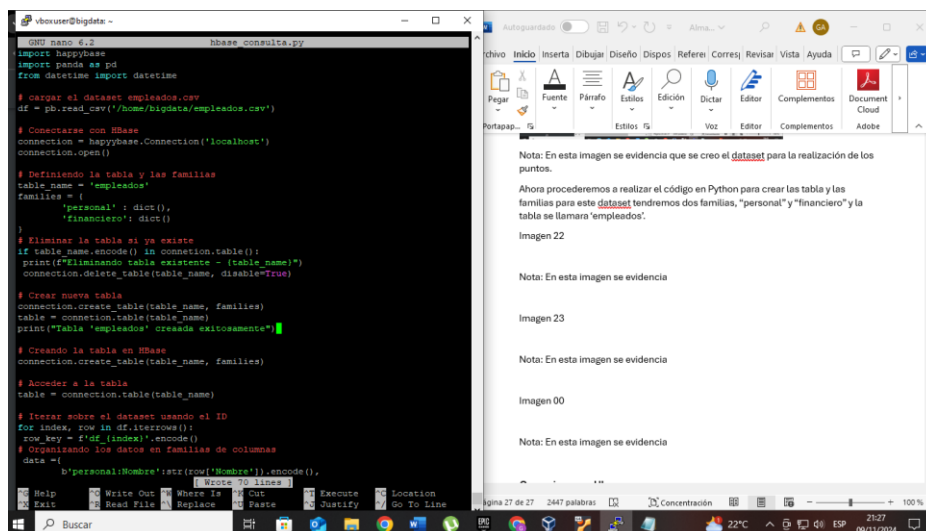
Imagen 21



Nota: En esta imagen se evidencia que se creó el dataset para la realización de los puntos.

Ahora procederemos a realizar el código en Python para crear la tabla y las familias para este dataset tendremos dos familias, “personal” y “financiero” y la tabla se llamará ‘empleados’.

Imagen 22



Nota: En esta imagen se evidencia el código que procesara el dataset.

Corremos el programa mediante la siguiente línea de comando.

>>\$ python3 hbase\_consulta.py

Imagen 23

```
vboxuser@bigdata:~$ python3 hbase_consulta.py
Eliminando tabla existente - empleados
Tabla 'empleados' creada exitosamente
Los datos se cargaron de manera exitosa

--- Se recorren todos los datos ---

ID: df_0
Nombre: Ana
Edad: 25
Salario: 150000

ID: df_1
Nombre: Pedro
Edad: 35
Salario: 600000

ID: df_10
Nombre: Luis
Edad: 5
Salario: 50000

ID: df_11
Nombre: Carla
Edad: 45
Salario: 850000

ID: df_12
Nombre: Inés
Edad: 38
Salario: 920000

ID: df_13
Nombre: Tomás
Edad: 28
Salario: 380000

ID: df_14
Nombre: Sara
Edad: 48
Salario: 670000

ID: df_15
```

Nota: En esta imagen se evidencia los datos arrojados de las diferentes consultas.

Ahora procederemos a explicar cada parte de código y que es lo que hace.

- **Operaciones en Hbase**

1. Cargar los datos en la tabla de HBase.

Imagen 24

```
vboxuser@bigdata: ~
GNU nano 6.2 hbase_consulta.py

# cargar el dataset empleados.csv
df = pd.read_csv('empleados.csv')

# Conectarse con HBase
connection = happybase.Connection('localhost')
connection.open()

# Definiendo la tabla y las familias
table_name = 'empleados'
families = {
    'personal': dict(),
    'financiero': dict()
}

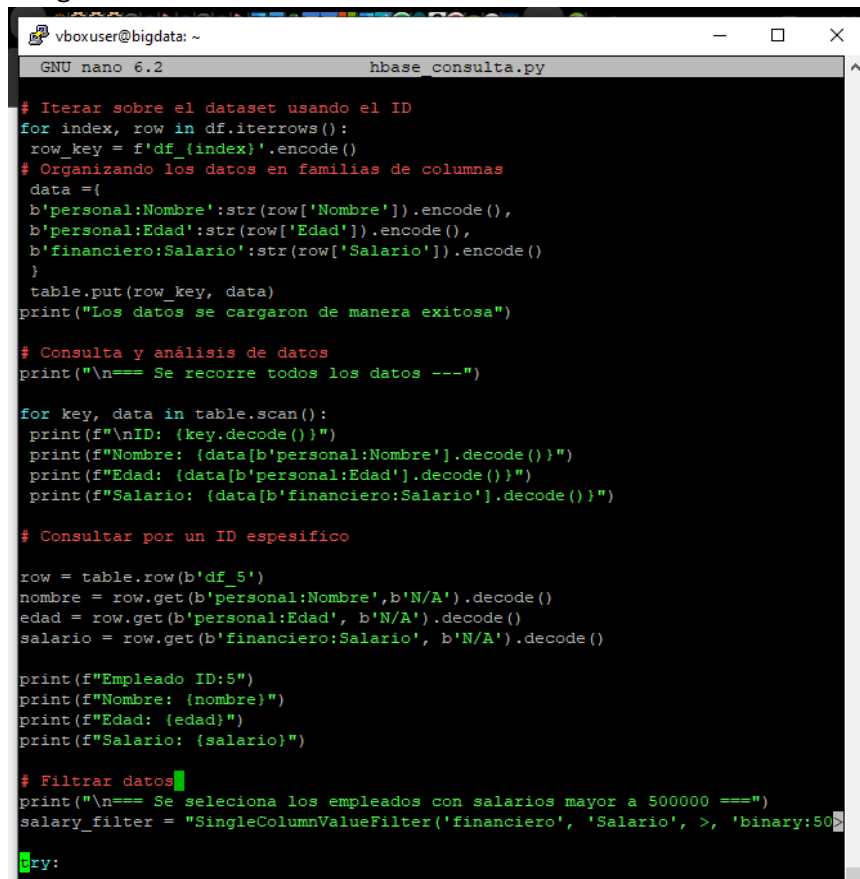
# Eliminar la tabla si ya existe
if table_name.encode() in connection.tables():
    print(f"Eliminando tabla existente - {table_name}")
    connection.delete_table(table_name, disable=True)

# Crear nueva tabla
connection.create_table(table_name, families)
table = connection.table(table_name)
print("Tabla 'empleados' creada exitosamente")
```

Nota: En esta imagen se evidencia el cargue del dataset

2. Realizar consultas de selección, filtrado y recorrido sobre los datos.

Imagen 25



```
GNU nano 6.2 hbase_consulta.py

# Iterar sobre el dataset usando el ID
for index, row in df.iterrows():
    row_key = f'df_{index}'.encode()
    # Organizando los datos en familias de columnas
    data = {
        b'personal:Nombre':str(row['Nombre']).encode(),
        b'personal:Edad':str(row['Edad']).encode(),
        b'financiero:Salario':str(row['Salario']).encode()
    }
    table.put(row_key, data)
print("Los datos se cargaron de manera exitosa")

# Consulta y análisis de datos
print("\n=== Se recorre todos los datos ===")

for key, data in table.scan():
    print(f"\nID: {key.decode()}")
    print(f"Nombre: {data[b'personal:Nombre'].decode()}")
    print(f"Edad: {data[b'personal:Edad'].decode()}")
    print(f"Salario: {data[b'financiero:Salario'].decode()}")

# Consultar por un ID específico

row = table.row(b'df_5')
nombre = row.get(b'personal:Nombre', b'N/A').decode()
edad = row.get(b'personal:Edad', b'N/A').decode()
salario = row.get(b'financiero:Salario', b'N/A').decode()

print(f"Empleado ID:5")
print(f"Nombre: {nombre}")
print(f"Edad: {edad}")
print(f"Salario: {salario}")

# Filtrar datos
print("\n=== Se selecciona los empleados con salarios mayor a 500000 ===")
salary_filter = "SingleColumnValueFilter('financiero', 'Salario', >, 'binary:500000')>

zy:
```

Nota: En esta imagen se evidencia las consultas de recorrido, consulta y filtrado sobre los datos.

3. Realizar operaciones de escritura (inserción, actualización y eliminación) sobre los datos.

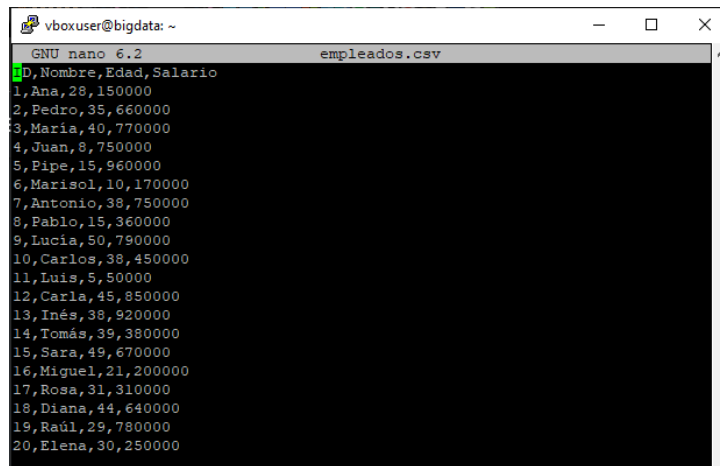
Imagen 26

Nota: En esta imagen se evidencia

- **Documentación**

1. Describir la estructura de la tabla en HBase.  
Tenemos un dataset que está compuesta por 4 columnas y 20 registros.

Imagen 27

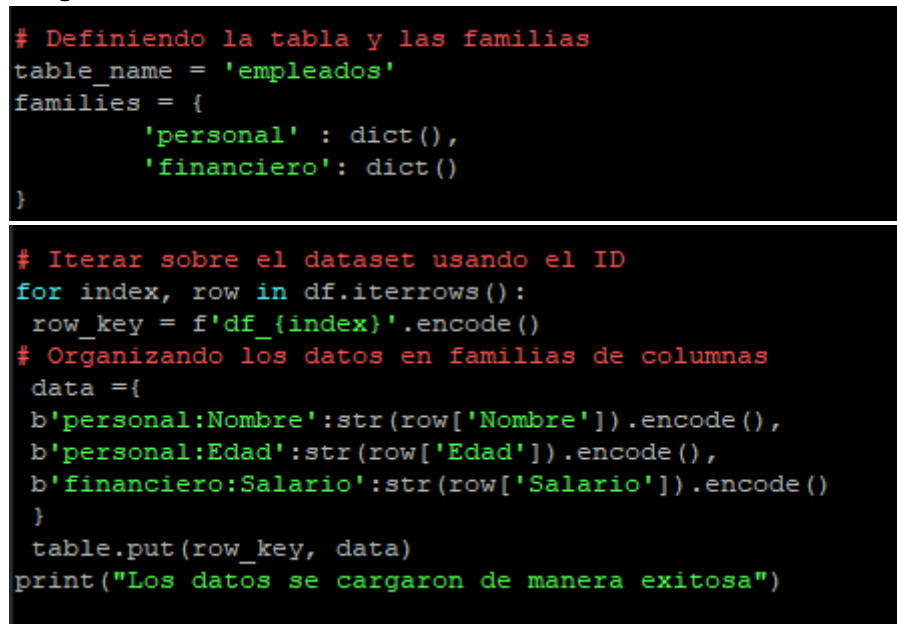


```
GNU nano 6.2 empleados.csv
ID,Nombre,Edad,Salario
1,Ana,28,150000
2,Pedro,35,660000
3,Maria,40,770000
4,Juan,8,750000
5,Pipe,15,960000
6,Marisol,10,170000
7,Antonio,38,750000
8,Pablo,15,360000
9,Lucia,50,790000
10,Carlos,38,450000
11,Luis,5,50000
12,Carla,45,850000
13,Inés,38,920000
14,Tomás,39,380000
15,Sara,49,670000
16,Miguel,21,200000
17,Rosa,31,310000
18,Diana,44,640000
19,Raúl,29,780000
20,Elena,30,250000
```

Nota: En esta imagen se evidencia el dataset de nombre empleados.csv

Ahora en Hbase crearemos una tabla a la cual nombraremos ‘empleados’ y definimos dos familias una se llamará ‘personal’ la cual contendrá las columnas de nombre y edad, la otra familia es ‘financiero’ la cual contendrá la columna salario.

Imagen 28



```
# Definiendo la tabla y las familias
table_name = 'empleados'
families = {
    'personal' : dict(),
    'financiero': dict()
}

# Iterar sobre el dataset usando el ID
for index, row in df.iterrows():
    row_key = f'df_{index}'.encode()
    # Organizando los datos en familias de columnas
    data = {
        b'personal:Nombre':str(row['Nombre']).encode(),
        b'personal:Edad':str(row['Edad']).encode(),
        b'financiero:Salario':str(row['Salario']).encode()
    }
    table.put(row_key, data)
print("Los datos se cargaron de manera exitosa")
```

Nota: En estas imágenes se evidencia la creación de la tabla y familias, en la siguiente imagen se muestra como se carga los datos en la tabla y las familias.

2. Explicar las operaciones realizadas en HBase y los resultados obtenidos.

En la imagen 29 se puede ver que mediante un ciclo “for” se itera sobre la tabla y se muestra el ID, el nombre, la edad y el salario.

Imagen 29



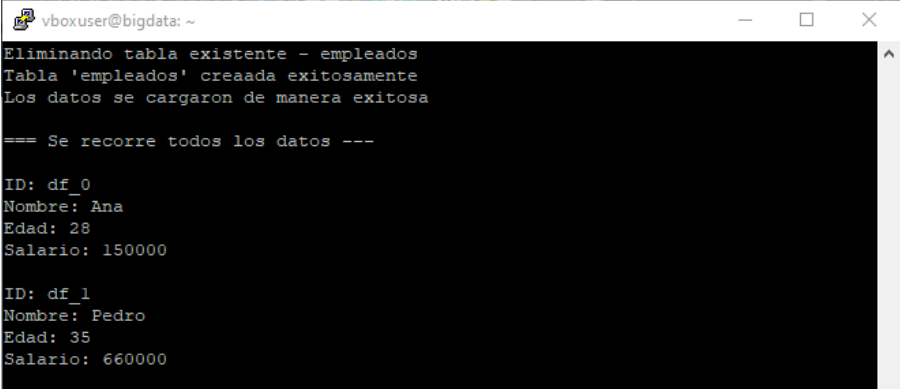
```
# Consulta y análisis de datos
print("\n=== Se recorre todos los datos ---")

for key, data in table.scan():
    print(f"\nID: {key.decode()}")
    print(f"Nombre: {data[b'personal:Nombre'].decode()}")
    print(f"Edad: {data[b'personal:Edad'].decode()}")
    print(f"Salario: {data[b'financiero:Salario'].decode()}")
```

Nota: En esta imagen se evidencia la parte del código que muestra el dataset.

En la imagen 28 se evidencia el resultado en la terminal.

Imagen 30



```
vboxuser@bigdata: ~
Eliminando tabla existente - empleados
Tabla 'empleados' creada exitosamente
Los datos se cargaron de manera exitosa

=== Se recorre todos los datos ---

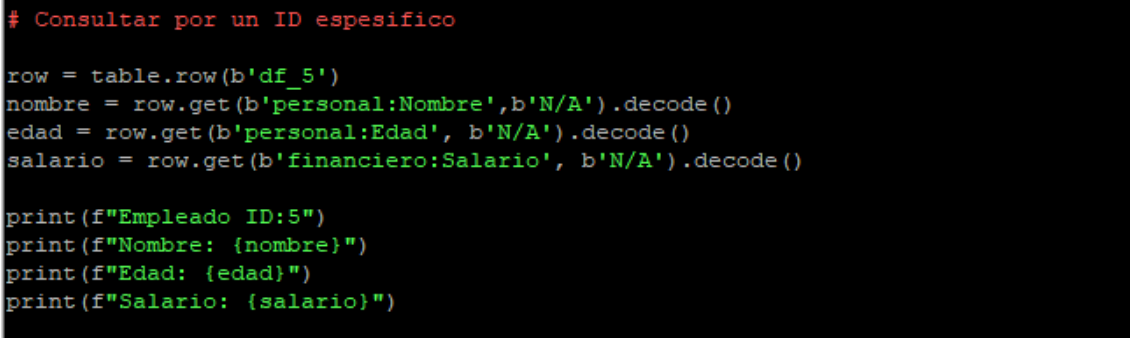
ID: df_0
Nombre: Ana
Edad: 28
Salario: 150000

ID: df_1
Nombre: Pedro
Edad: 35
Salario: 660000
```

Nota: En esta imagen se evidencia el resultado de recorrer el dataset.

En la imagen 31 se evidencia el código para realizar la consulta por un ID el cual es "df\_5".

Imagen 31



```
# Consultar por un ID específico

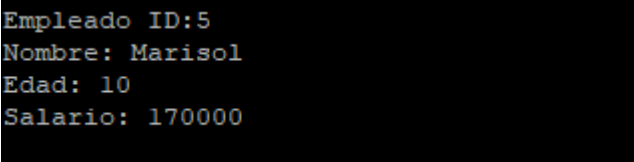
row = table.row(b'df_5')
nombre = row.get(b'personal:Nombre', b'N/A').decode()
edad = row.get(b'personal:Edad', b'N/A').decode()
salario = row.get(b'financiero:Salario', b'N/A').decode()

print(f"Empleado ID:5")
print(f"Nombre: {nombre}")
print(f"Edad: {edad}")
print(f"Salario: {salario}")
```

Nota: En esta imagen se evidencia las líneas de código que nos mostrara los datos del ID df\_5.

En la imagen 32 se evidencia el resultado de la consulta por ID.

Imagen 32



```
Empleado ID:5
Nombre: Marisol
Edad: 10
Salario: 170000
```

Nota: En esta imagen se evidencia que en el ID 5 tenemos a Marisol con una edad de 10 y un salario de \$170000.

En la imagen 33 podemos ver una consulta filtrando los datos, aquí se solicita mostrar las personas que tengan un salario mayor a \$500.000.

Imagen 33

```
# Filtrar datos
print("\n=== Se selecciona los empleados con salarios mayor a 500000 ===")
salary_filter = "SingleColumnValueFilter('financiero', 'Salario', '>', 'binary:500000')

try:
    for key, data in table.scan(filter=salary_filter.encode()):
        print(f"\nUsuario ID: {key.decode()}")
        print(f"Nombre: {data[b'personal:Nombre'].decode()}")
        print(f"Edad: {data[b'personal:Edad'].decode()}")
        print(f"Salario: {data[b'financiero:Salario'].decode()}")
except Exception as e:
    print("Error al realizar la consulta:", e)

# Cerrar la conexión
connection.close()
```

Nota: En esta imagen se evidencia el código para filtrar datos.

En la imagen 34 se muestra el resultado en consola es que tenemos 8 personas que cumplen con la condición del filtro.

Imagen 34

```
=== Se selecciona los empleados con salarios mayor a 500000 ===

Usuario ID: df_1
Nombre:Pedro
Edad:35
Salario:660000

Usuario ID: df_11
Nombre:Carla
Edad:45
Salario:850000

Usuario ID: df_12
Nombre:Inés
Edad:38
Salario:920000
```

Nota: En esta imagen se evidencia el resultado de la consulta de filtrado

En la imagen 35 se puede ver el código que donde se define que al ID 5 se le cambiara el valor de la variable salario que es \$170.000 y se cambiara por \$ 355.967.

Imagen 35

```
# Insercion de datos
print("\n=== Se insertara un nuevo salario para el empleado de ID 5 ===")

usuario_actualizar = 'df_5'
nuevo_salario = 355967
table.put(usuario_actualizar.encode(), {b'financiero:Salario':str(nuevo_salario).encode()})

# Guarda los datos en variables

row = table.row(b'df_5')
nombre = row.get(b'personal:Nombre', b'N/A').decode()
edad = row.get(b'personal:Edad', b'N/A').decode()
salario = row.get(b'financiero:Salario', b'N/A').decode()

# Se muestran los datos por consola

print(f"Empleado ID:5")
print(f"Nombre: {nombre}")
print(f"Edad: {edad}")
print(f"Salario: {salario}")

print(f"\n=== Salario actualizado para el ID: {usuario_actualizar}")
```

Nota: En esta imagen se evidencia la parte del código para el cambio de salario para el ID 5 y se muestra por pantalla.

Imagen 36

```
=== Se consulta la información del ID ===  
Empleado ID:5  
Nombre: Marisol  
Edad: 10  
Salario: 170000  
  
=== Se insertara un nuevo salario para el empleado de ID 5 ===  
Empleado ID:5  
Nombre: Marisol  
Edad: 10  
Salario: 355967  
  
=== Salario actualizado para el ID: df_5
```

Nota: En estas imágenes se evidencian el antes y después de la actualización del salario de ID 5