Materia: Computación Tolerante a Fallas.

NRC: 179961

Maestra: Lopez Franco, Michael Emanuel

Aula: X-02

Sección: D06

Alumno: Zashuvath López Moreno, Ethan Israel

Código: 216493953

Índice

Introducción

Realizaremos la modificación de un código proporcionado por el profesor, el cual fue elaborado por https://gist.github.com/lauralorenz

Objetivo:

Realizar modificaciones al código de Laura Lorenz

Contenido

Código base proporcionado por el profesor:

```python
import requests
import json
from collections import namedtuple
from contextlib import closing
import sqlite3

from prefect import task, Flow

## extract
@task
def get_complaint_data():
    r = requests.get("https://www.consumerfinance.gov/data-research/consumer-complaints/search/api/v1/", params={'size':10})
    response_json = json.loads(r.text)
    return response_json['hits']['hits']

## transform
@task
def parse_complaint_data(raw):
    complaints = []
    Complaint = namedtuple('Complaint', ['data_received', 'state', 'product', 'company', 'complaint_what_happened'])
    for row in raw:
        source = row.get('_source')
        this_complaint = Complaint(
            data_received=source.get('date_recieved'),
            state=source.get('state'),
            product=source.get('product'),
            company=source.get('company'),
            complaint_what_happened=source.get('complaint_what_happened')
        )
        complaints.append(this_complaint)
    return complaints

```

```python
## load
@task
def store_complaints(parsed):
    create_script = 'CREATE TABLE IF NOT EXISTS complaint (timestamp TEXT, state TEXT, product TEXT, company TEXT, complaint_what_happened TEXT)'
    insert_cmd = "INSERT INTO complaint VALUES (?, ?, ?, ?, ?)"

    with closing(sqlite3.connect("cfpbcomplaints.db")) as conn:
        with closing(conn.cursor()) as cursor:
            cursor.executescript(create_script)
            cursor.executemany(insert_cmd, parsed)
            conn.commit()

with Flow("my etl flow") as f:
    raw = get_complaint_data()
    parsed = parse_complaint_data(raw)
    store_complaints(parsed)

f.run()
```

Código modificado

```python
import requests
import json
from collections import namedtuple
from contextlib import closing
import sqlite3
import logging

from prefect import task, Flow

# Cambio 1: Configuración de la URL de la API y el nombre de la base de datos
API_URL = "https://www.consumerfinance.gov/data-research/consumer-complaints/search/api/v1/"
DATABASE_NAME = "cfpbcomplaints.db"

# Cambio 2: Configuración de registro
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

# Cambio 3: Función para realizar solicitudes HTTP de manera segura
def safe_request(url, params=None):
    try:
        response = requests.get(url, params=params)
        response.raise_for_status()
        return response.text
    except requests.exceptions.RequestException as e:
        logger.error(f"Error en la solicitud HTTP: {e}")
        raise

@task
def get_complaint_data():
    # Cambio 4: Uso de la función safe_request para solicitudes HTTP
    response_text = safe_request(API_URL, params={'size': 10})
    response_json = json.loads(response_text)
    return response_json['hits']['hits']

@task
def parse_complaint_data(raw):
    complaints = []
    Complaint = namedtuple('Complaint', ['data_received', 'state', 'product', 'company', 'complaint_what_happened'])
    for row in raw:
        source = row.get('_source')
        this_complaint = Complaint(
            data_received=source.get('date_received'),
            state=source.get('state'),
            product=source.get('product'),
            company=source.get('company'),
            complaint_what_happened=source.get('complaint_what_happened')
        )
        complaints.append(this_complaint)
    return complaints

@task
def store_complaints(parsed):
    create_script = "CREATE TABLE IF NOT EXISTS complaint (timestamp TEXT, state TEXT, product TEXT, company TEXT, complaint_what_happened TEXT)"
    insert_cmd = "INSERT INTO complaint VALUES (?, ?, ?, ?, ?)"

    try:
        # Cambio 5: Manejo de errores al interactuar con la base de datos
        with closing(sqlite3.connect(DATABASE_NAME)) as conn:
            with closing(conn.cursor()) as cursor:
                cursor.executescript(create_script)
                cursor.executemany(insert_cmd, parsed)
                conn.commit()
    except sqlite3.Error as e:
        logger.error(f"Error al interactuar con la base de datos: {e}")
```

```
        raise

with Flow("my etl flow") as f:
    raw = get_complaint_data()
    parsed = parse_complaint_data(raw)
    store_complaints(parsed)

if __name__ == "__main__":
    f.run()
```

Cambio 1: Configuración de la URL de la API y el nombre de la base de datos: Aquí se establece la URL de la API de donde se obtendrán los datos y el nombre de la base de datos SQLite donde se almacenarán los datos extraídos.

Cambio 2: Configuración de registro: Se configura el sistema de registro (logging) para registrar mensajes informativos a nivel de INFO. Esto permitirá la creación de registros que ayuden en la depuración y el seguimiento del flujo de trabajo.

Cambio 3: Función para realizar solicitudes HTTP de manera segura: Se define una función llamada **'safe_request'** que se utiliza para realizar solicitudes HTTP a la API de manera segura. La función captura excepciones de solicitudes HTTP y registra errores en caso de que ocurran.

Cambio 4: Uso de la función safe_request para solicitudes HTTP: En la función **'get_complaint_data()'**, se utiliza la función **'safe_request'** para realizar la solicitud HTTP a la API. Esto asegura que cualquier error en la solicitud sea manejado adecuadamente.

Cambio 5: Manejo de errores al interactuar con la base de datos: En la función **'store_complaints(parsed)'**, se utiliza un bloque **'try...except'** para manejar errores al interactuar con la base de datos SQLite. Si ocurre un error al crear la tabla o al insertar datos, se registrará un mensaje de error.

Conclusión

Las modificaciones que se realizaron en el código son para mejorar la robustez y la seguridad del flujo de trabajo ETL (Extract, Transform and Load), además de añadir configuraciones para la API y el registro de errores, una función segura para las solicitudes HTTP y la implementación de manejadores de errores para las interacciones con SQLite.

Bibliografía

JSONPlaceholder - fake online REST API for developers. (s. f.). https://jsonplaceholder.cypress.io/

Prefect. (2020, 17 abril). Getting started with Prefect (PyData Denver) [Vídeo]. YouTube. https://www.youtube.com/watch?v=FETN0iivZps

PyData.(2019, 4 enero). Task failed successfully - Jeremiah Lowin [Vídeo]. YouTube. https://www.youtube.com/watch?v=TlawR_gi8-YLauraLorenz's gists.

(s. f.). Gist. https://gist.github.com/lauralorenz