```
# data.processor

import geopandas as gpd
import pandas as pd
import streamlit as st
from shapely import wkt
from sqlalchemy import create_engine, text

from modules.config import Config


# Función auxiliar robusta para fechas en español (ene-70 -> datetime)
def parse_spanish_date_robust(x):
    if isinstance(x, pd.Timestamp):
        return x
    if pd.isna(x) or x == "":
        return pd.NaT
    x = str(x).lower().strip()
    trans = {
        "ene": "Jan",
        "feb": "Feb",
        "mar": "Mar",
        "abr": "Apr",
        "may": "May",
        "jun": "Jun",
        "jul": "Jul",
        "ago": "Aug",
        "sep": "Sep",
        "oct": "Oct",
        "nov": "Nov",
        "dic": "Dec",
    }
    for es, en in trans.items():
        if es in x:
            x = x.replace(es, en)
            break
    try:
        # Intentar formato mes-año corto (Jan-70)
        return pd.to_datetime(x, format="%b-%y")
    except:
        try:
            # Intentar formato estándar
            return pd.to_datetime(x)
        except:
            return pd.NaT
```

```
@st.cache_data(show_spinner="Procesando datos...", ttl=600)
def load_and_process_all_data():
    gdf_stations = pd.DataFrame()
    gdf_municipios = pd.DataFrame()
    gdf_subcuencas = pd.DataFrame()
    gdf_predios = pd.DataFrame()
    df_long = pd.DataFrame()
    df_enso = pd.DataFrame()

    try:
        if "DATABASE_URL" not in st.secrets:
            st.error("Falta DATABASE_URL.")
            return None, None, None, None, None, None

        engine = create_engine(st.secrets["DATABASE_URL"])

        # 1. ESTACIONES
        try:
            sql_est = text(
                "SELECT id_estacion, nom_est, alt_est, municipio, depto_region, ST_AsText(geom) as wkt FROM
estaciones"
            )
            df_est = pd.read_sql(sql_est, engine)

            if "wkt" in df_est.columns:
                df_est["geometry"] = df_est["wkt"].apply(
                    lambda x: wkt.loads(x) if x else None
                )
                gdf_stations = gpd.GeoDataFrame(
                    df_est, geometry="geometry", crs="EPSG:4326"
                )
            else:
                gdf_stations = df_est.copy()

            # Etiqueta única
            def create_lbl(row):
                n, c = str(row["nom_est"]).strip(), str(row["id_estacion"]).strip()
                return n if c in n else f"{n} [{c}]"

            gdf_stations["station_label"] = gdf_stations.apply(create_lbl, axis=1)

            # Dropear 'nom_est' para evitar duplicados al renombrar
            if "nom_est" in gdf_stations.columns:
                gdf_stations = gdf_stations.drop(columns=["nom_est"])
            gdf_stations = gdf_stations.rename(
                columns={
                    "station_label": Config.STATION_NAME_COL,
                    "alt_est": Config.ALTITUDE_COL,
                    "municipio": Config.MUNICIPALITY_COL,
                    "depto_region": Config.REGION_COL,
                }
            )

            if "geometry" in gdf_stations.columns:
                gdf_stations = gdf_stations.dropna(subset=["geometry"])
                gdf_stations["latitude"] = gdf_stations.geometry.y
                gdf_stations["longitude"] = gdf_stations.geometry.x
        except Exception as e:
            st.warning(f"Estaciones: {e}")

        # 2. PRECIPITACIÓN
        try:
            sql_ppt = text(
                'SELECT id_estacion_fk, "fecha_mes_año", precipitation FROM precipitacion_mensual'
            )
            df_ppt = pd.read_sql(sql_ppt, engine)

            # LIMPIEZA DE FECHAS (AQUÍ ESTÁ LA SOLUCIÓN RAÍZ)
            # Convertir fechas usando el parser robusto para manejar 'ene-70'
            df_ppt[Config.DATE_COL] = df_ppt["fecha_mes_año"].apply(
                parse_spanish_date_robust
            )
            # Eliminar filas con fechas inválidas (NaT)
            df_ppt = df_ppt.dropna(subset=[Config.DATE_COL])

            if not gdf_stations.empty:
                df_long = pd.merge(
                    df_ppt,
                    gdf_stations[["id_estacion", Config.STATION_NAME_COL]],
                    left_on="id_estacion_fk",
                    right_on="id_estacion",
                    how="inner",
                )
                df_long = df_long.rename(
                    columns={"precipitation": Config.PRECIPITATION_COL}
                )
                df_long[Config.YEAR_COL] = df_long[Config.DATE_COL].dt.year
                df_long[Config.MONTH_COL] = df_long[Config.DATE_COL].dt.month
        except Exception as e:
            st.error(f"Precipitación: {e}")
```

```
# 3. GEOMETRÍAS
try:
    sql_geo = text(
        "SELECT nombre, tipo_geometria, ST_AsText(geom) as wkt FROM geometrias"
    )
    df_geo = pd.read_sql(sql_geo, engine)
    if not df_geo.empty:
        df_geo["geometry"] = df_geo["wkt"].apply(
            lambda x: wkt.loads(x) if x else None
        )
        gdf_all = gpd.GeoDataFrame(df_geo, geometry="geometry", crs="EPSG:4326")
        gdf_municipios = gdf_all[gdf_all["tipo_geometria"] == "municipio"]
        gdf_subcuencas = gdf_all[
            gdf_all["tipo_geometria"].isin(["subcuenca", "cuenca"])
        ]
        gdf_predios = gdf_all[gdf_all["tipo_geometria"] == "predio"]
except:
    pass

# 4. ENSO (Indices)
try:
    df_enso = pd.read_sql(text("SELECT * FROM indices_climaticos"), engine)
    df_enso.columns = [c.lower() for c in df_enso.columns]

    # LIMPIEZA DE FECHAS ENSO (IGUALMENTE CRÍTICO)
    col_fecha = "fecha" if "fecha" in df_enso.columns else "fecha_mes_año"
    df_enso[Config.DATE_COL] = df_enso[col_fecha].apply(
        parse_spanish_date_robust
    )
    df_enso = df_enso.dropna(subset=[Config.DATE_COL])

    if "oni" in df_enso.columns:
        df_enso = df_enso.rename(columns={"oni": Config.ENSO_ONI_COL})
    elif "anomalia_oni" in df_enso.columns:
        df_enso = df_enso.rename(columns={"anomalia_oni": Config.ENSO_ONI_COL})
except:
    pass

return (
    gdf_stations,
    gdf_municipios,
    df_long,
    df_enso,
    gdf_subcuencas,
    gdf_predios,
        )

    except Exception as e:
        st.error(f"Error DB: {e}")
        return None, None, None, None, None, None


def complete_series(df):
    if df is None or df.empty:
        return df
    df = df.sort_values(Config.DATE_COL)
    df[Config.PRECIPITATION_COL] = df[Config.PRECIPITATION_COL].interpolate(
        method="linear", limit_direction="both"
    )
    return df
```