**Step 1 HTML Cleaning for DB**

**user**

Please act as a data scientist and full stack developer.

We have a jupyter lab that we are connecting to a postgres database.

DB\_HOST = '10.0.0.105'

DB\_PORT = '5432'

DB\_NAME = 'postgres'

DB\_USER = 'postgres\_admin'

DB\_PASSWORD = 'redacted'

Here is the schema.

####

CREATE TABLE feeds (

    feed\_id SERIAL PRIMARY KEY,

    feed\_url TEXT UNIQUE NOT NULL,

    feed\_name TEXT,

    source\_type TEXT,

    relevance\_score FLOAT,

    last\_updated TIMESTAMP,

    active BOOLEAN DEFAULT TRUE,

    created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE articles (

    article\_id SERIAL PRIMARY KEY,

    feed\_id INTEGER REFERENCES feeds(feed\_id),

    title TEXT NOT NULL,

    link TEXT UNIQUE NOT NULL,

    description TEXT,

    publication\_date TIMESTAMP,

    content TEXT,

    tags TEXT[],

    sentiment\_score FLOAT,

    engagement\_score INTEGER,

    fetched\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

    content\_markup\_free TEXT,  -- New column for content without markup

    description\_markup\_free TEXT  -- New column for description without markup

);

CREATE TABLE authors (

    author\_id SERIAL PRIMARY KEY,

    author\_name TEXT UNIQUE NOT NULL

);

CREATE TABLE author\_articles (

    article\_id INTEGER REFERENCES articles(article\_id),

    author\_id INTEGER REFERENCES authors(author\_id),

    PRIMARY KEY (article\_id, author\_id)

);

####

This database is from collected RSS feeds.

We need to clean all of the html and other mark up junk from

    description TEXT,

    content TEXT,

and store the cleaned results in these columns

    content\_markup\_free TEXT,  -- New column for content without markup

    description\_markup\_free TEXT  -- New column for description without markup

Please help me with some python that will do this.

**ChatGPT**

To clean the HTML and other markup junk from the `description` and `content` fields in your PostgreSQL database and store the cleaned results in the respective columns (`content\_markup\_free` and `description\_markup\_free`), you can use Python with the `psycopg2` library to interact with PostgreSQL and the `BeautifulSoup` library from `bs4` for HTML parsing and cleaning.

Below is the Python script to achieve this:

### Python Script

```python

import psycopg2

from bs4 import BeautifulSoup

# Database connection details

DB\_HOST = '10.0.0.105'

DB\_PORT = '5432'

DB\_NAME = 'postgres'

DB\_USER = 'postgres\_admin'

DB\_PASSWORD = 'redacted'

def clean\_html\_content(raw\_html):

    """Remove HTML tags and clean up the content."""

    soup = BeautifulSoup(raw\_html, 'html.parser')

    return soup.get\_text()

def update\_cleaned\_columns():

    """Fetch content and description, clean them, and update the database."""

    try:

        # Connect to the PostgreSQL database

        connection = psycopg2.connect(

            host=DB\_HOST,

            port=DB\_PORT,

            database=DB\_NAME,

            user=DB\_USER,

            password=DB\_PASSWORD

        )

        cursor = connection.cursor()

        # Select rows that need cleaning

        fetch\_query = """

        SELECT article\_id, content, description

        FROM articles

        WHERE content IS NOT NULL OR description IS NOT NULL;

        """

        cursor.execute(fetch\_query)

        rows = cursor.fetchall()

        for row in rows:

            article\_id, content, description = row

            cleaned\_content = clean\_html\_content(content) if content else None

            cleaned\_description = clean\_html\_content(description) if description else None

            # Update the cleaned content in the database

            update\_query = """

            UPDATE articles

            SET content\_markup\_free = %s, description\_markup\_free = %s

            WHERE article\_id = %s;

            """

            cursor.execute(update\_query, (cleaned\_content, cleaned\_description, article\_id))

        # Commit the transaction

        connection.commit()

        print(f"Processed {cursor.rowcount} rows successfully.")

    except (Exception, psycopg2.Error) as error:

        print("Error while working with PostgreSQL:", error)

    finally:

        # Close the database connection

        if connection:

            cursor.close()

            connection.close()

            print("PostgreSQL connection is closed.")

if \_\_name\_\_ == "\_\_main\_\_":

    update\_cleaned\_columns()

```

### Steps to Execute:

1. \*\*Install dependencies\*\*:

   - Install `psycopg2` for PostgreSQL interaction:

     ```bash

     pip install psycopg2

     ```

   - Install `BeautifulSoup` for cleaning HTML:

     ```bash

     pip install beautifulsoup4

     ```

2. \*\*Run the script\*\*:

   Save the script to a `.py` file (e.g., `clean\_html.py`) and run it:

   ```bash

   python clean\_html.py

   ```

### Explanation:

- \*\*Database Connection\*\*:

  The script connects to your PostgreSQL database using the provided credentials.

- \*\*HTML Cleaning\*\*:

  The `clean\_html\_content` function uses `BeautifulSoup` to remove all HTML tags and return plain text.

- \*\*Updating Database\*\*:

  - The script fetches articles where `content` or `description` is not `NULL`.

  - Cleans these fields and updates the `content\_markup\_free` and `description\_markup\_free` columns.

- \*\*Commit Changes\*\*:

  Updates are committed in batches for performance and transactional integrity.

This script ensures your database columns `content\_markup\_free` and `description\_markup\_free` have cleaned content. Let me know if you need further assistance!

**user**

does this notebook need a cell to close the database connection?

**ChatGPT**

Yes, it is a good practice to ensure the database connection is closed when it is no longer needed to prevent resource leaks. In the provided script, the connection is closed in the `finally` block of the `try-except-finally` structure.

If you're using this script within a Jupyter Notebook, you can add a cell to ensure the connection is explicitly closed if you're testing and don't want to rely on the `finally` block or if you're handling the connection lifecycle manually.

### Example of Connection Management in Jupyter Notebook

```python

# Import necessary libraries

import psycopg2

from bs4 import BeautifulSoup

# Database connection details

DB\_HOST = '10.0.0.105'

DB\_PORT = '5432'

DB\_NAME = 'postgres'

DB\_USER = 'postgres\_admin'

DB\_PASSWORD = 'redacted'

# Create a connection and cursor

connection = psycopg2.connect(

    host=DB\_HOST,

    port=DB\_PORT,

    database=DB\_NAME,

    user=DB\_USER,

    password=DB\_PASSWORD

)

cursor = connection.cursor()

# Define the cleaning function

def clean\_html\_content(raw\_html):

    """Remove HTML tags and clean up the content."""

    soup = BeautifulSoup(raw\_html, 'html.parser')

    return soup.get\_text()

# Cell to fetch, clean, and update

def clean\_and\_update():

    """Fetch content and description, clean them, and update the database."""

    fetch\_query = """

    SELECT article\_id, content, description

    FROM articles

    WHERE content IS NOT NULL OR description IS NOT NULL;

    """

    cursor.execute(fetch\_query)

    rows = cursor.fetchall()

    for row in rows:

        article\_id, content, description = row

        cleaned\_content = clean\_html\_content(content) if content else None

        cleaned\_description = clean\_html\_content(description) if description else None

        update\_query = """

        UPDATE articles

        SET content\_markup\_free = %s, description\_markup\_free = %s

        WHERE article\_id = %s;

        """

        cursor.execute(update\_query, (cleaned\_content, cleaned\_description, article\_id))

    connection.commit()

    print(f"Processed {len(rows)} rows successfully.")

# Cleanup cell to close the connection

def close\_connection():

    """Close the database connection."""

    if cursor:

        cursor.close()

    if connection:

        connection.close()

        print("Database connection closed.")

```

### Steps to Use in Jupyter Notebook

1. \*\*Setup Database Connection\*\*:

   Run the cell that establishes the `connection` and `cursor`.

2. \*\*Clean and Update\*\*:

   Execute the function `clean\_and\_update()` to clean and update the database.

3. \*\*Close Connection\*\*:

   Run the `close\_connection()` function after finishing all operations to cleanly close the connection.

By explicitly closing the connection in its own cell or function, you prevent accidental open connections during iterative testing in Jupyter Notebook.

**user**

this is an on going project, will this tool check if the cell has already been cleaned and can be skipped to save time as the database grows?

**ChatGPT**

To optimize the script for a growing database and avoid reprocessing already-cleaned rows, you can add a check to skip rows where `content\_markup\_free` or `description\_markup\_free` has already been populated. This ensures the tool only processes rows that haven't been cleaned yet, saving time as the database grows.

Here’s an updated version of the script that includes this check:

USE THIS CODE IN Jupyter Labs

### Updated Python Script

```python

import psycopg2

from bs4 import BeautifulSoup

# Database connection details

DB\_HOST = '10.0.0.105'

DB\_PORT = '5432'

DB\_NAME = 'postgres'

DB\_USER = 'postgres\_admin'

DB\_PASSWORD = 'redacted'

def clean\_html\_content(raw\_html):

    """Remove HTML tags and clean up the content."""

    soup = BeautifulSoup(raw\_html, 'html.parser')

    return soup.get\_text()

def update\_cleaned\_columns():

    """Fetch uncleaned content and description, clean them, and update the database."""

    try:

        # Connect to the PostgreSQL database

        connection = psycopg2.connect(

            host=DB\_HOST,

            port=DB\_PORT,

            database=DB\_NAME,

            user=DB\_USER,

            password=DB\_PASSWORD

        )

        cursor = connection.cursor()

        # Select only rows that have not been cleaned

        fetch\_query = """

        SELECT article\_id, content, description

        FROM articles

        WHERE (content IS NOT NULL AND content\_markup\_free IS NULL)

           OR (description IS NOT NULL AND description\_markup\_free IS NULL);

        """

        cursor.execute(fetch\_query)

        rows = cursor.fetchall()

        print(f"Found {len(rows)} rows to process.")

        for row in rows:

            article\_id, content, description = row

            cleaned\_content = clean\_html\_content(content) if content else None

            cleaned\_description = clean\_html\_content(description) if description else None

            # Update the cleaned content in the database

            update\_query = """

            UPDATE articles

            SET content\_markup\_free = %s, description\_markup\_free = %s

            WHERE article\_id = %s;

            """

            cursor.execute(update\_query, (cleaned\_content, cleaned\_description, article\_id))

        # Commit the transaction

        connection.commit()

        print(f"Processed {len(rows)} rows successfully.")

    except (Exception, psycopg2.Error) as error:

        print("Error while working with PostgreSQL:", error)

    finally:

        # Close the database connection

        if connection:

            cursor.close()

            connection.close()

            print("PostgreSQL connection is closed.")

if \_\_name\_\_ == "\_\_main\_\_":

    update\_cleaned\_columns()

```python

### Key Changes in the Updated Script:

1. \*\*Selective Fetch\*\*:

   The `fetch\_query` now checks whether the cleaned columns (`content\_markup\_free` and `description\_markup\_free`) are `NULL`. This ensures only rows that haven't been cleaned are processed.

   ```sql

   WHERE (content IS NOT NULL AND content\_markup\_free IS NULL)

      OR (description IS NOT NULL AND description\_markup\_free IS NULL);

   ```

2. \*\*Efficiency\*\*:

   - By skipping already-cleaned rows, the script saves processing time and database resources.

   - The logic ensures you won’t process rows unnecessarily, even as the database grows.

3. \*\*Progress Indicator\*\*:

   A `print` statement shows how many rows need processing, which is helpful for debugging and monitoring.

### Benefits:

- \*\*Scalability\*\*:

  The script can handle a growing database without needing to reprocess rows.

- \*\*Idempotency\*\*:

  Re-running the script multiple times doesn’t affect previously processed rows, ensuring safe incremental processing.

### Example Workflow for Continuous Use:

1. Schedule this script to run periodically (e.g., using `cron` on Linux or Task Scheduler on Windows).

2. As new articles are added, the script will only clean those rows that haven't been processed yet.