import sqlite3

import json

class DataBaseHandler:

def \_\_init\_\_(self, db\_path, log):

"""

Initialize the database handler.

Args:

db\_path (str): The path to the SQLite database.

log: Logger object for logging messages.

"""

self.conn = sqlite3.connect(db\_path, timeout=30)

self.cursor = self.conn.cursor()

self.log = log

def create\_tables(self):

"""

Create tables if they don't exist already.

"""

# ... (omitted for brevity)

def get\_row\_info(self, fields, table\_name, condition=None, return\_as\_list=False):

"""

Extract information from the database based on specified conditions and fields.

Args:

fields (list or str): Columns to retrieve.

table\_name (str): Name of the table.

condition (dict): Dictionary containing field and value for filtering data.

return\_as\_list (bool): Set to True to return selected rows as a list.

Returns:

List or nested list of selected rows.

"""

# ... (omitted for brevity)

def get\_column\_names(self, table\_name):

"""

Get column names of a given table.

Args:

table\_name (str): Name of the table.

Returns:

List of column names.

"""

# ... (omitted for brevity)

def check\_field\_value(self, row\_data, crawl\_data):

"""

Check whether a specific field has been filled or not.

Args:

row\_data (dict): A single record fetched from the database.

crawl\_data (dict): The data to compare with the database.

Returns:

True if the field has been filled, False otherwise.

"""

# ... (omitted for brevity)

def check\_existing\_data(self, row\_id, column\_name, table\_name):

"""

Check if the fields in the database have been updated or not.

Args:

row\_id (str): Unique id of each entry in the database.

column\_name (str): Name of the field to check for an update.

table\_name (str): Name of the table to check.

Returns:

Data as json if existing, else False.

"""

# ... (omitted for brevity)

def parse\_json\_fields(self, record):

"""

Parse fields to JSON.

Args:

record (dict): Record which contains multiple fields.

Returns:

JSON object of the record.

"""

# ... (omitted for brevity)

def replace\_recode\_to\_history\_table(self, data, column\_name, table\_name):

"""

Move old data from the main table to the history table.

Args:

data (dict): A dictionary that will be inserted into the History Table.

column\_name (str): Name of the column used to identify each row uniquely.

table\_name (str): Name of the table where the data needs to be moved.

Returns:

True if success, otherwise False.

"""

# ... (omitted for brevity)

def insert\_recode\_to\_table(self, data, table\_name):

"""

Insert a new line or update an existing one in a given table.

Args:

data (dict): A dictionary containing field name and value pairs.

table\_name (str): Name of the table.

"""

# ... (omitted for brevity)

def update\_database(self, data, column\_name, table\_name):

"""

Update records in a table based on a column.

Args:

data (dict): A dictionary containing field name and value pairs.

column\_name (str): Column name used to filter rows.

table\_name (str): Table where the changes are made.

"""

# ... (omitted for brevity)

def close\_connection(self):

"""

Close the connection to the database.

"""

self.conn.commit()

self.conn.close()