Arquivo: database.py

```
from PyQt6.QtWidgets import *
import sqlite3
import logging
import pandas as pd
import re

class DatabaseATASAPIManager:
    def __init__(self, db_path):
        self.db_path = str(db_path)
        self.connection = None

    def set_database_path(self, db_path):
        """Permite alterar dinamicamente o caminho do banco de dados."""
        self.db_path = db_path
        logging.info(f"Database path set to: {self.db_path}")

    def connect_to_database(self):
        """Cria uma nova conexão com o banco de dados."""
        try:
            conn = sqlite3.connect(self.db_path, check_same_thread=False)
            return conn
        except sqlite3.Error as e:
            logging.error(f"Erro ao conectar ao banco de dados: {e}")
            raise

    def close_connection(self):
        if self.connection:
            logging.info("Fechando conexão com o banco de dados...")
            self.connection.close()
            self.connection = None
            logging.info(f"Conexão com o banco de dados fechada em {self.db_path}")

    def is_closed(self):
        return self.connection is None

    def __enter__(self):
        self.connect_to_database()
        return self.connection

    def __exit__(self, exc_type, exc_val, exc_tb):
        self.close_connection()

    def save_dataframe(self, df, table_name):
        """Salva um DataFrame no banco de dados, lidando com duplicações."""
        try:
            with self.connect_to_database() as conn:
                cursor = conn.cursor()

                # Verifica se a tabela existe
                cursor.execute(f"PRAGMA table_info({table_name})")
```

```
                if not cursor.fetchall():
                    raise ValueError(f"A tabela '{table_name}' não existe no banco de dados.")

                # Remove duplicações com base na coluna 'item', se existir
                if 'item' in df.columns:
                    items = df['item'].tolist()
                    placeholders = ', '.join('?' for _ in items)
                    delete_query = f"DELETE FROM {table_name} WHERE item IN ({placeholders})"
                    cursor.execute(delete_query, items)

                # Salva o DataFrame
                df.to_sql(table_name, conn, if_exists='append', index=False)
                logging.info(f"DataFrame salvo na tabela '{table_name}' com sucesso.")
        except sqlite3.IntegrityError as e:
            duplicated_items = df.loc[df.duplicated(subset=['item'], keep=False), 'item']
                    error_message = f"Erro de integridade: Valores duplicados em 'item': {duplicated_items.to_list()}"
            logging.error(error_message)
            QMessageBox.warning(None, "Erro de Duplicação", error_message)
        except Exception as e:
            logging.error(f"Erro ao salvar DataFrame na tabela '{table_name}': {e}")


    def delete_record(self, table_name, column, value):
        """Deleta um registro com base em uma coluna e valor."""
        query = f"DELETE FROM {table_name} WHERE {column} = ?"
        self.execute_update(query, (value,))

    def execute_query(self, query, params=None):
        """Executa uma consulta de leitura no banco de dados."""
        try:
            with self.connect_to_database() as conn:
                cursor = conn.cursor()
                cursor.execute(query, params or ())
                result = cursor.fetchall()
                logging.info(f"Consulta executada: {query}")
                return result
        except sqlite3.Error as e:
            logging.error(f"Erro ao executar consulta: {query}, Erro: {e}")
            return None

    def execute_update(self, query, params=None):
        """Executa uma consulta de escrita no banco de dados."""
        try:
            with self.connect_to_database() as conn:
                cursor = conn.cursor()
                cursor.execute(query, params or ())
                conn.commit()
                logging.info(f"Atualização executada: {query}")
                return True
        except sqlite3.Error as e:
            logging.error(f"Erro ao executar atualização: {query}, Erro: {e}")
            return False
```

```
def consultar_registro(self, tabela, campo, valor):
    """
    Consulta um registro na tabela especificada com base no campo e valor fornecidos.

    :param tabela: Nome da tabela no banco de dados.
    :param campo: Nome do campo a ser filtrado (ex: 'cnpj').
    :param valor: Valor a ser buscado no campo.
    :return: Dicionário com os dados do registro, ou None se não encontrado.
    """
    query = f"SELECT * FROM {tabela} WHERE {campo} = ?"
    print(f"Executando consulta: {query} com valor: {valor}")

    resultado = self.execute_query(query, (valor,))

    # Verifica se o resultado da consulta está vazio
    if resultado:
        print(f"Resultado da consulta encontrado: {resultado}")
        try:
            # Obtenção dos nomes das colunas para transformar o resultado em dicionário
            with self.connect_to_database() as conn:
                cursor = conn.cursor()
                cursor.execute(query, (valor,))
                colunas = [desc[0] for desc in cursor.description]

            print(f"Colunas da tabela: {colunas}")
            registro_dict = dict(zip(colunas, resultado[0]))
            print(f"Registro encontrado: {registro_dict}")
            return registro_dict

        except sqlite3.Error as e:
            print(f"Erro ao transformar o resultado em dicionário: {e}")
            logging.error(f"Erro ao transformar o resultado em dicionário: {e}")
            return None

    print(f"Nenhum registro encontrado para {campo} = {valor} na tabela {tabela}.")
    logging.info(f"Nenhum registro encontrado para {campo} = {valor} na tabela {tabela}.")
    return None


def verify_and_create_columns(self, table_name, required_columns):
    """Verifica e cria colunas faltantes em uma tabela."""
    try:
        with self.connect_to_database() as conn:
            cursor = conn.cursor()

            # Verifica as colunas existentes
            cursor.execute(f"PRAGMA table_info({table_name})")
            existing_columns = {row[1]: row[2] for row in cursor.fetchall()}

            # Adiciona colunas ausentes
            for column, column_type in required_columns.items():
                if column not in existing_columns:
```

```
                                cursor.execute(f"ALTER TABLE {table_name} ADD COLUMN {column}
{column_type}")
                        logging.info(f"Coluna '{column}' adicionada à tabela '{table_name}'.")

                conn.commit()
        except sqlite3.Error as e:
            logging.error(f"Erro ao verificar ou criar colunas na tabela '{table_name}': {e}")


    def get_tables_with_keyword(self, keyword):
        """Retorna tabelas contendo uma palavra-chave no nome."""
        try:
            query = "SELECT name FROM sqlite_master WHERE type='table' AND name LIKE ?"
            tables = self.execute_query(query, (f"%{keyword}%",))
            return [table[0] for table in tables]
        except Exception as e:
            logging.error(f"Erro ao buscar tabelas com a palavra-chave '{keyword}': {e}")
            return []


    def load_table_to_dataframe(self, table_name):
        """Carrega uma tabela em um DataFrame."""
        try:
            with self.connect_to_database() as conn:
                return pd.read_sql_query(f"SELECT * FROM {table_name}", conn)
        except Exception as e:
            logging.error(f"Erro ao carregar tabela '{table_name}': {e}")
            return None

    def salvar_consulta_api_no_db(self, data_informacoes):
        # Função salva os dados no banco de dados, verificando se já existe uma entrada
        with self.connect_to_database() as conn:
            cursor = conn.cursor()
            # Verificar e criar a tabela se necessário
            cursor.execute("""
                CREATE TABLE IF NOT EXISTS pregoes_consultados (
                    valorTotalEstimado REAL,
                    valorTotalHomologado REAL,
                    orcamentoSigilosoCodigo INTEGER,
                    orcamentoSigilosoDescricao TEXT,
                    numeroControlePNCP TEXT PRIMARY KEY,
                    linkSistemaOrigem TEXT,
                    linkProcessoEletronico TEXT,
                    anoCompra INTEGER,
                    sequencialCompra INTEGER,
                    numeroCompra TEXT,
                    processo TEXT,
                    orgaoEntidadeCnpj TEXT,
                    orgaoEntidadeRazaoSocial TEXT,
                    orgaoEntidadeEsferaId TEXT,
                    orgaoEntidadePoderId TEXT,
                    unidadeOrgaoCodigoUnidade TEXT,
                    unidadeOrgaoNomeUnidade TEXT,
```

```
                    unidadeOrgaoMunicipioNome TEXT,
                    unidadeOrgaoCodigoIbge TEXT,
                    unidadeOrgaoUfSigla TEXT,
                    unidadeOrgaoUfNome TEXT,
                    modalidadeId INTEGER,
                    modalidadeNome TEXT,
                    justificativaPresencial TEXT,
                    modoDisputaId INTEGER,
                    modoDisputaNome TEXT,
                    tipoInstrumentoConvocatorioCodigo INTEGER,
                    tipoInstrumentoConvocatorioNome TEXT,
                    amparoLegalCodigo INTEGER,
                    amparoLegalNome TEXT,
                    amparoLegalDescricao TEXT,
                    objetoCompra TEXT,
                    informacaoComplementar TEXT,
                    srp BOOLEAN,
                    dataPublicacaoPncp TEXT,
                    dataAberturaProposta TEXT,
                    dataEncerramentoProposta TEXT,
                    situacaoCompraId INTEGER,
                    situacaoCompraNome TEXT,
                    existeResultado BOOLEAN,
                    dataInclusao TEXT,
                    dataAtualizacao TEXT,
                    usuarioNome TEXT
                )
            """)
            # Inserir ou atualizar a entrada no banco de dados
            cursor.execute("""
                INSERT INTO pregoes_consultados (
                                valorTotalEstimado, valorTotalHomologado, orcamentoSigilosoCodigo,
orcamentoSigilosoDescricao,
                        numeroControlePNCP, linkSistemaOrigem, linkProcessoEletronico, anoCompra,
sequencialCompra,
                            numeroCompra, processo, orgaoEntidadeCnpj, orgaoEntidadeRazaoSocial,
orgaoEntidadeEsferaId,
                        orgaoEntidadePoderId, unidadeOrgaoCodigoUnidade, unidadeOrgaoNomeUnidade,
unidadeOrgaoMunicipioNome,
                    unidadeOrgaoCodigoIbge, unidadeOrgaoUfSigla, unidadeOrgaoUfNome, modalidadeId,
modalidadeNome,
                                        justificativaPresencial, modoDisputaId, modoDisputaNome,
tipoInstrumentoConvocatorioCodigo,
                            tipoInstrumentoConvocatorioNome, amparoLegalCodigo, amparoLegalNome,
amparoLegalDescricao,
                                objetoCompra, informacaoComplementar, srp, dataPublicacaoPncp,
dataAberturaProposta,
                                dataEncerramentoProposta, situacaoCompraId, situacaoCompraNome,
existeResultado, dataInclusao,
                    dataAtualizacao, usuarioNome
                ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?,
?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
                ON CONFLICT(numeroControlePNCP) DO UPDATE SET
```

```
                    valorTotalEstimado = excluded.valorTotalEstimado,
                    valorTotalHomologado = excluded.valorTotalHomologado,
                    orcamentoSigilosoCodigo = excluded.orcamentoSigilosoCodigo,
                    orcamentoSigilosoDescricao = excluded.orcamentoSigilosoDescricao,
                    linkSistemaOrigem = excluded.linkSistemaOrigem,
                    linkProcessoEletronico = excluded.linkProcessoEletronico,
                    anoCompra = excluded.anoCompra,
                    sequencialCompra = excluded.sequencialCompra,
                    numeroCompra = excluded.numeroCompra,
                    processo = excluded.processo,
                    orgaoEntidadeCnpj = excluded.orgaoEntidadeCnpj,
                    orgaoEntidadeRazaoSocial = excluded.orgaoEntidadeRazaoSocial,
                    orgaoEntidadeEsferaId = excluded.orgaoEntidadeEsferaId,
                    orgaoEntidadePoderId = excluded.orgaoEntidadePoderId,
                    unidadeOrgaoCodigoUnidade = excluded.unidadeOrgaoCodigoUnidade,
                    unidadeOrgaoNomeUnidade = excluded.unidadeOrgaoNomeUnidade,
                    unidadeOrgaoMunicipioNome = excluded.unidadeOrgaoMunicipioNome,
                    unidadeOrgaoCodigoIbge = excluded.unidadeOrgaoCodigoIbge,
                    unidadeOrgaoUfSigla = excluded.unidadeOrgaoUfSigla,
                    unidadeOrgaoUfNome = excluded.unidadeOrgaoUfNome,
                    modalidadeId = excluded.modalidadeId,
                    modalidadeNome = excluded.modalidadeNome,
                    justificativaPresencial = excluded.justificativaPresencial,
                    modoDisputaId = excluded.modoDisputaId,
                    modoDisputaNome = excluded.modoDisputaNome,
                                                        tipoInstrumentoConvocatorioCodigo   =
excluded.tipoInstrumentoConvocatorioCodigo,
                    tipoInstrumentoConvocatorioNome = excluded.tipoInstrumentoConvocatorioNome,
                    amparoLegalCodigo = excluded.amparoLegalCodigo,
                    amparoLegalNome = excluded.amparoLegalNome,
                    amparoLegalDescricao = excluded.amparoLegalDescricao,
                    objetoCompra = excluded.objetoCompra,
                    informacaoComplementar = excluded.informacaoComplementar,
                    srp = excluded.srp,
                    dataPublicacaoPncp = excluded.dataPublicacaoPncp,
                    dataAberturaProposta = excluded.dataAberturaProposta,
                    dataEncerramentoProposta = excluded.dataEncerramentoProposta,
                    situacaoCompraId = excluded.situacaoCompraId,
                    situacaoCompraNome = excluded.situacaoCompraNome,
                    existeResultado = excluded.existeResultado,
                    dataInclusao = excluded.dataInclusao,
                    dataAtualizacao = excluded.dataAtualizacao,
                    usuarioNome = excluded.usuarioNome
                """, (
                data_informacoes.get("valorTotalEstimado"),
                data_informacoes.get("valorTotalHomologado"),
                data_informacoes.get("orcamentoSigilosoCodigo"),
                data_informacoes.get("orcamentoSigilosoDescricao"),
                data_informacoes.get("numeroControlePNCP"),
                data_informacoes.get("linkSistemaOrigem"),
                data_informacoes.get("linkProcessoEletronico"),
                data_informacoes.get("anoCompra"),
                data_informacoes.get("sequencialCompra"),
```

```
                data_informacoes.get("numeroCompra"),
                data_informacoes.get("processo"),
                data_informacoes.get("orgaoEntidade", {}).get("cnpj"),
                data_informacoes.get("orgaoEntidade", {}).get("razaoSocial"),
                data_informacoes.get("orgaoEntidade", {}).get("esferaId"),
                data_informacoes.get("orgaoEntidade", {}).get("poderId"),
                data_informacoes.get("unidadeOrgao", {}).get("codigoUnidade"),
                data_informacoes.get("unidadeOrgao", {}).get("nomeUnidade"),
                data_informacoes.get("unidadeOrgao", {}).get("municipioNome"),
                data_informacoes.get("unidadeOrgao", {}).get("codigoIbge"),
                data_informacoes.get("unidadeOrgao", {}).get("ufSigla"),
                data_informacoes.get("unidadeOrgao", {}).get("ufNome"),
                data_informacoes.get("modalidadeId"),
                data_informacoes.get("modalidadeNome"),
                data_informacoes.get("justificativaPresencial"),
                data_informacoes.get("modoDisputaId"),
                data_informacoes.get("modoDisputaNome"),
                data_informacoes.get("tipoInstrumentoConvocatorioCodigo"),
                data_informacoes.get("tipoInstrumentoConvocatorioNome"),
                data_informacoes.get("amparoLegal", {}).get("codigo"),
                data_informacoes.get("amparoLegal", {}).get("nome"),
                data_informacoes.get("amparoLegal", {}).get("descricao"),
                data_informacoes.get("objetoCompra"),
                data_informacoes.get("informacaoComplementar"),
                data_informacoes.get("srp"),
                data_informacoes.get("dataPublicacaoPncp"),
                data_informacoes.get("dataAberturaProposta"),
                data_informacoes.get("dataEncerramentoProposta"),
                data_informacoes.get("situacaoCompraId"),
                data_informacoes.get("situacaoCompraNome"),
                data_informacoes.get("existeResultado"),
                data_informacoes.get("dataInclusao"),
                data_informacoes.get("dataAtualizacao"),
                data_informacoes.get("usuarioNome")
            ))
            conn.commit()

    def criar_tabela_itens_pregao(self, numeroCompra, anoCompra, unidadeOrgaoCodigoUnidade):
        table_name = f"resultAPI_{numeroCompra}_{anoCompra}_{unidadeOrgaoCodigoUnidade}"
        column_order = [
                    'grupo', 'item PRIMARY KEY', 'catalogo', 'descricao', 'descricao_detalhada',
'unidade', 'quantidade', 'valor_estimado',
                                        'valor_homologado_item_unitario',   'percentual_desconto',
'valor_estimado_total_do_item',
            'valor_homologado_total_item', 'marca_fabricante', 'modelo_versao', 'situacao',
                    'uasg', 'orgao_responsavel', 'num_pregao', 'ano_pregao', 'srp', 'objeto',
'melhor_lance', 'valor_negociado', 'ordenador_despesa', 'empresa',
            'cnpj', 'endereco', 'cep', 'municipio', 'telefone', 'email', 'responsavel_legal'
        ]

        with self.connect_to_database() as conn:
            cursor = conn.cursor()
            columns_definition = ", ".join(column_order)
```

```
                        create_table_query = f"CREATE TABLE IF NOT EXISTS '{table_name}'
({columns_definition})"
            cursor.execute(create_table_query)
            conn.commit()

    def popular_db_consulta_itens_api(self, resultados_completos, data_informacoes, numeroCompra,
anoCompra, unidadeOrgaoCodigoUnidade, controle_atas_dict):
        table_name = f"resultAPI_{numeroCompra}_{anoCompra}_{unidadeOrgaoCodigoUnidade}"

        # Informações gerais para inserção
        data_informacoes_to_insert = {
            "ano_pregao": data_informacoes.get("anoCompra"),
            "num_pregao": data_informacoes.get("numeroCompra"),
            "uasg": data_informacoes.get("unidadeOrgao", {}).get("codigoUnidade"),
            "orgao_responsavel": data_informacoes.get("unidadeOrgao", {}).get("nomeUnidade"),
            "objeto": data_informacoes.get("objetoCompra"),
            "srp": data_informacoes.get("srp")
        }

        # Carregar dados da tabela controle_atas_api
        controle_atas_api = self.execute_query("SELECT item, catalogo FROM controle_atas_api WHERE
catalogo IS NOT NULL")
        controle_atas_dict = {row[0]: row[1] for row in controle_atas_api}

        # Para cada item em resultados_completos, insere ou atualiza os dados no banco de dados
        for item in resultados_completos:
            numero_item = item.get("numeroItem")

            # Verifica se existe um valor de catalogo no controle_atas_dict
            catalogo = controle_atas_dict.get(numero_item)

            quantidade = item.get("quantidadeHomologada", 0) or 0
            valor_estimado = item.get("valorUnitarioEstimado", 0) or 0
            valor_homologado_item_unitario = item.get("valorUnitarioHomologado")

            if valor_estimado and valor_homologado_item_unitario is not None:
                percentual_desconto = (
                    ((valor_estimado - valor_homologado_item_unitario) / valor_estimado * 100)
                    if valor_estimado else None
                )
            else:
                percentual_desconto = None

            valor_homologado_total_item = quantidade * (valor_homologado_item_unitario or 0)
            valor_estimado_total_do_item = quantidade * valor_estimado

            # Determina a situação com base no valor booleano
                   situacao = 'Adjudicado e Homologado' if item.get("temResultado") == 1 else
'Fracassado/Deserto/Cancelado ou Anulado'

            # Valida o formato de CNPJ ou CPF
            ni_fornecedor = item.get("niFornecedor")
            if ni_fornecedor:
```

```
                # Mantém o formato se já estiver formatado corretamente
                if re.match(r'^\d{2}\.\d{3}\.\d{3}/\d{4}-\d{2}$', ni_fornecedor):  # CNPJ
                    cnpj = ni_fornecedor
                elif re.match(r'^\d{3}\.\d{3}\.\d{3}-\d{2}$', ni_fornecedor):  # CPF
                    cnpj = ni_fornecedor
                else:
                    # Se estiver apenas com números, formata como CNPJ ou CPF
                    if len(ni_fornecedor) == 14:  # CNPJ
                                                                                cnpj    =
f"{ni_fornecedor[:2]}.{ni_fornecedor[2:5]}.{ni_fornecedor[5:8]}/{ni_fornecedor[8:12]}-{ni_forneced
or[12:]}"
                    elif len(ni_fornecedor) == 11:  # CPF
                                                                                cnpj    =
f"{ni_fornecedor[:3]}.{ni_fornecedor[3:6]}.{ni_fornecedor[6:9]}-{ni_fornecedor[9:]}"
                    else:
                        cnpj = ni_fornecedor  # Deixa como está se não for reconhecido
            else:
                cnpj = None

            # Combina dados específicos do item com as informações gerais e os cálculos
            data_to_insert = {
                "item": numero_item,
                "catalogo": catalogo,
                "descricao": item.get("descricao"),
                "descricao_detalhada": item.get("descricao"),
                "unidade": item.get("unidadeMedida"),
                "quantidade": item.get("quantidadeHomologada", 0) or 0,
                "valor_estimado": item.get("valorUnitarioEstimado", 0) or 0,
                "valor_homologado_item_unitario": item.get("valorUnitarioHomologado"),
                "percentual_desconto": percentual_desconto,
                "valor_homologado_total_item": valor_homologado_total_item,
                "valor_estimado_total_do_item": valor_estimado_total_do_item,
                     "situacao": 'Adjudicado e Homologado' if item.get("temResultado") == 1 else
'Fracassado/Deserto/Cancelado ou Anulado',
                "cnpj": cnpj,  # Usa o valor formatado
                "empresa": item.get("nomeRazaoSocialFornecedor"),
                **data_informacoes_to_insert  # Adiciona as informações de data_informacoes
            }

            # SQL para inserção
            placeholders = ", ".join(["?"] * len(data_to_insert))
            columns = ", ".join(data_to_insert.keys())
                    insert_query = f"INSERT OR REPLACE INTO '{table_name}' ({columns}) VALUES
({placeholders})"
            self.execute_update(insert_query, tuple(data_to_insert.values()))
```