Arquivo: db_manager.py

```
import sqlite3
import logging
import pandas as pd

def carregar_dados_dispensa(id_processo, caminho_banco_dados):
    try:
        logging.debug(f"Conectando ao banco de dados: {caminho_banco_dados}")
        connection = sqlite3.connect(caminho_banco_dados)
        query = f"SELECT * FROM controle_dispensas WHERE id_processo='{id_processo}'"
        logging.debug(f"Executando consulta SQL: {query}")
        df_registro_selecionado = pd.read_sql_query(query, connection)
        connection.close()
                logging.debug(f"Dados carregados com sucesso para id_processo {id_processo}:
{df_registro_selecionado}")
        return df_registro_selecionado
    except Exception as e:
        logging.error(f"Erro ao carregar dados do banco de dados: {e}", exc_info=True)
        return pd.DataFrame()  # Retorna um DataFrame vazio em caso de erro

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

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

    def connect_to_database(self):
        try:
            connection = sqlite3.connect(self.db_path)
            return connection
        except sqlite3.Error as e:
            logging.error(f"Failed to connect to database at {self.db_path}: {e}")
            raise

    def execute_query(self, query, params=None):
        with self.connect_to_database() as conn:
            try:
                cursor = conn.cursor()
                if params:
                    cursor.execute(query, params)
                else:
                    cursor.execute(query)
                return cursor.fetchall()
            except sqlite3.Error as e:
                logging.error(f"Error executing query: {query}, Error: {e}")
                return None

    def execute_update(self, query, params=None):
```

```
    with self.connect_to_database() as conn:
        try:
            cursor = conn.cursor()
            if params:
                cursor.execute(query, params)
            else:
                cursor.execute(query)
            conn.commit()
        except sqlite3.Error as e:
            logging.error(f"Error executing update: {query}, Error: {e}")
            return False
        return True

def close_connection(self):
    if self.connection:
        self.connection.close()

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

def fetch_all(self, query):
    """
    Executa uma consulta SQL e retorna todos os registros.

    :param query: A string de consulta SQL a ser executada.
    :return: Uma lista de dicionários, onde cada dicionário representa uma linha do resultado.
    """
    with sqlite3.connect(self.db_path) as conn:
        cursor = conn.cursor()
        cursor.execute(query)
        columns = [col[0] for col in cursor.description]  # Nomes das colunas
        rows = cursor.fetchall()
        # Converte cada linha em um dicionário com chaves sendo os nomes das colunas
        data = [dict(zip(columns, row)) for row in rows]
    return data

def delete_data(self, id_processo):
    """Exclui um registro da tabela 'controle_dispensas' pelo id_processo."""
    query = "DELETE FROM controle_dispensas WHERE id_processo = ?"
    return self.execute_update(query, (id_processo,))

def get_distinct_years(self):
    """Busca todos os anos distintos da tabela de dispensas."""
    # A consulta extrai os 4 últimos caracteres da coluna 'id_processo'
    query = "SELECT DISTINCT SUBSTR(id_processo, -4) FROM controle_dispensas ORDER BY 1 DESC"
    try:
        # Usando o método execute_query já existente na classe
        rows = self.execute_query(query)
        # Retorna uma lista de anos. Ex: ['2025', '2024']
        return [row[0] for row in rows] if rows else []
    except Exception as e:
        logging.error(f"Erro ao buscar anos distintos: {e}")
        return []
```