import pandas as pd

from sqlalchemy import create\_engine, text

from typing import Dict, Union, Optional, Any

import pyodbc

import teradatasql

from utils import log\_error, validate\_connection\_params, clean\_df\_columns

class DatabaseConnector:

    """Class to handle database connections and data retrieval"""

    @staticmethod

    def get\_sqlserver\_data(connection\_params: Dict[str, str], query: str) -> Optional[pd.DataFrame]:

        """

        Retrieve data from SQL Server using provided connection parameters and query.

        Args:

            connection\_params: Dictionary containing connection parameters

            query: SQL query to execute

        Returns:

            Optional[pd.DataFrame]: DataFrame containing query results, None if error occurs

        """

        if not validate\_connection\_params(connection\_params):

            log\_error("Invalid connection parameters for SQL Server")

            return None

        try:

            # Get server name from either 'server' or 'host' parameter

            server = connection\_params.get('server') or connection\_params.get('host')

            if not server:

                raise ValueError("Server/Host is required")

            database = connection\_params.get('database')

            if not database:

                raise ValueError("Database name is required")

            # Build connection string based on authentication type

            if connection\_params.get('use\_windows\_auth', True):

                conn\_str = (

                    f"Driver={{ODBC Driver 17 for SQL Server}};"

                    f"Server={server};"

                    f"Database={database};"

                    "Trusted\_Connection=yes"

                )

            else:

                username = connection\_params.get('username')

                password = connection\_params.get('password')

                if not username or not password:

                    raise ValueError("Username and password are required for SQL authentication")

                conn\_str = (

                    f"Driver={{ODBC Driver 17 for SQL Server}};"

                    f"Server={server};"

                    f"Database={database};"

                    f"Uid={username};"

                    f"Pwd={password}"

                )

            # Create engine using the pyodbc connection string

            # Format: mssql+pyodbc://username:password@server/database?driver=ODBC+Driver+17+for+SQL+Server

            if connection\_params.get('use\_windows\_auth', True):

                engine = create\_engine(

                    f"mssql+pyodbc://{server}/{database}?driver=ODBC+Driver+17+for+SQL+Server&trusted\_connection=yes",

                    fast\_executemany=True

                )

            else:

                engine = create\_engine(

                    f"mssql+pyodbc://{username}:{password}@{server}/{database}?driver=ODBC+Driver+17+for+SQL+Server",

                    fast\_executemany=True

                )

            # Connect and execute query

            with engine.connect() as connection:

                df = pd.read\_sql(text(query), connection)

                df = clean\_df\_columns(df)  # Clean column names for consistency

                return df

        except Exception as e:

            log\_error(f"Error retrieving data from SQL Server: {str(e)}")

            return None

    @staticmethod

    def get\_teradata\_data(connection\_params: Dict[str, str], query: str) -> Optional[pd.DataFrame]:

        """

        Retrieve data from Teradata using provided connection parameters and query.

        Args:

            connection\_params: Dictionary containing connection parameters

            query: SQL query to execute

        Returns:

            Optional[pd.DataFrame]: DataFrame containing query results, None if error occurs

        """

        try:

            if not validate\_connection\_params(connection\_params):

                log\_error("Invalid connection parameters for Teradata")

                return None

            # Create Teradata connection

            conn = teradatasql.connect(

                host=connection\_params['host'],

                user=connection\_params['username'],

                password=connection\_params['password'],

                database=connection\_params['database']

            )

            # Execute query and return results as DataFrame

            df = pd.read\_sql(query, conn)

            conn.close()

            return df

        except Exception as e:

            log\_error(f"Error retrieving data from Teradata: {str(e)}")

            return None

    @staticmethod

    def get\_data\_from\_stored\_proc(connection\_params: Dict[str, str],

                                 proc\_name: str,

                                 params: Optional[Dict[str, Any]] = None) -> Optional[pd.DataFrame]:

        """

        Execute a stored procedure and retrieve its results.

        Args:

            connection\_params: Dictionary containing connection parameters

            proc\_name: Name of the stored procedure to execute

            params: Optional dictionary of parameters for the stored procedure

        Returns:

            Optional[pd.DataFrame]: DataFrame containing procedure results, None if error occurs

        """

        if not validate\_connection\_params(connection\_params):

            log\_error("Invalid connection parameters for stored procedure execution")

            return None

        try:

            # Get server name from either 'server' or 'host' parameter

            server = connection\_params.get('server') or connection\_params.get('host')

            if not server:

                raise ValueError("Server/Host is required")

            database = connection\_params.get('database')

            if not database:

                raise ValueError("Database name is required")

            # Build connection string

            if connection\_params.get('use\_windows\_auth', True):

                conn\_str = f"DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server};DATABASE={database};Trusted\_Connection=yes"

            else:

                username = connection\_params.get('username')

                password = connection\_params.get('password')

                if not username or not password:

                    raise ValueError("Username and password are required for SQL authentication")

                conn\_str = f"DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server};DATABASE={database};UID={username};PWD={password}"

            # Connect and execute stored procedure using pyodbc

            with pyodbc.connect(conn\_str, timeout=30) as conn:

                cursor = conn.cursor()

                if params:

                    # Build parameter string

                    param\_str = ', '.join([f"@{k}=?" for k in params.keys()])

                    exec\_str = f"EXEC {proc\_name} {param\_str}"

                    cursor.execute(exec\_str, list(params.values()))

                else:

                    cursor.execute(f"EXEC {proc\_name}")

                # Fetch results

                columns = [column[0] for column in cursor.description]

                results = cursor.fetchall()

                # Convert to DataFrame and clean column names

                df = pd.DataFrame.from\_records(results, columns=columns)

                df = clean\_df\_columns(df)  # Clean column names for consistency

                return df

        except Exception as e:

            log\_error(f"Error executing stored procedure: {str(e)}")

            return None

    @staticmethod

    def test\_connection(connection\_params: Dict[str, str], db\_type: str) -> bool:

        """

        Test database connection without executing any queries.

        Args:

            connection\_params: Dictionary containing connection parameters

            db\_type: Type of database ('sqlserver' or 'teradata')

        Returns:

            bool: True if connection successful, False otherwise

        """

        try:

            if not validate\_connection\_params(connection\_params):

                return False

            if db\_type.lower() == 'sqlserver':

                try:

                    # Get server name from either 'server' or 'host' parameter

                    server = connection\_params.get('server') or connection\_params.get('host')

                    if not server:

                        raise ValueError("Server/Host is required")

                    database = connection\_params.get('database')

                    if not database:

                        raise ValueError("Database name is required")

                    # Build connection string

                    if connection\_params.get('use\_windows\_auth', True):

                        conn\_str = f"DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server};DATABASE={database};Trusted\_Connection=yes"

                    else:

                        username = connection\_params.get('username')

                        password = connection\_params.get('password')

                        if not username or not password:

                            raise ValueError("Username and password are required for SQL authentication")

                        conn\_str = f"DRIVER={{ODBC Driver 17 for SQL Server}};SERVER={server};DATABASE={database};UID={username};PWD={password}"

                    # Test connection with timeout

                    conn = pyodbc.connect(conn\_str, timeout=30)

                    conn.close()

                    return True

                except Exception as e:

                    log\_error(f"SQL Server connection test failed: {str(e)}")

                    return False

            elif db\_type.lower() == 'teradata':

                conn = teradatasql.connect(

                    host=connection\_params['host'],

                    user=connection\_params['username'],

                    password=connection\_params['password'],

                    database=connection\_params['database']

                )

                conn.close()

                return True

            else:

                log\_error(f"Unsupported database type: {db\_type}")

                return False

        except Exception as e:

            log\_error(f"Error testing connection: {str(e)}")

            return False

    @staticmethod

    def get\_table\_schema(connection\_params: Dict[str, str],

                        table\_name: str,

                        db\_type: str) -> Optional[pd.DataFrame]:

        """

        Retrieve schema information for a specified table.

        Args:

            connection\_params: Dictionary containing connection parameters

            table\_name: Name of the table

            db\_type: Type of database ('sqlserver' or 'teradata')

        Returns:

            Optional[pd.DataFrame]: DataFrame containing schema information, None if error occurs

        """

        try:

            if db\_type.lower() == 'sqlserver':

                query = f"""

                SELECT

                    COLUMN\_NAME,

                    DATA\_TYPE,

                    CHARACTER\_MAXIMUM\_LENGTH,

                    IS\_NULLABLE

                FROM INFORMATION\_SCHEMA.COLUMNS

                WHERE TABLE\_NAME = '{table\_name}'

                ORDER BY ORDINAL\_POSITION

                """

                return DatabaseConnector.get\_sqlserver\_data(connection\_params, query)

            elif db\_type.lower() == 'teradata':

                query = f"""

                SELECT

                    ColumnName,

                    ColumnType,

                    CharacterLength,

                    Nullable

                FROM DBC.Columns

                WHERE TableName = '{table\_name}'

                ORDER BY ColumnId

                """

                return DatabaseConnector.get\_teradata\_data(connection\_params, query)

            else:

                log\_error(f"Unsupported database type: {db\_type}")

                return None

        except Exception as e:

            log\_error(f"Error retrieving table schema: {str(e)}")

            return None