using System;

using System.Collections;

using System.Collections.Generic;

using System.Configuration;

using System.Data;

using System.Data.SqlClient;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace AMAT.O365.SeedAutomation.DAL

{

public class DatabaseConnection

{

private SqlConnection myConnection;

private SqlCommand Command;

private SqlDataReader myReader;

private string constr;

private string sql;

private int timeout;

private string error;

private bool isProc;

private List<string> pivotCols;

public void Dispose()

{

//this.myConnection.Close();

//GC.SuppressFinalize(this);

}

public DatabaseConnection()

{

constr = ConfigurationManager.ConnectionStrings["SeedAutomationCon"].ConnectionString;

timeout = 30;

error = "";

}

public string getError()

{

return error;

}

public static string InputStr(string inputStr)

{

return inputStr.Replace("'", "''");

}

public void addParameter(string paramName, string param)

{

// If the connection isn't opened yet, open it before adding parameters

if (this.Command == null) this.openDB();

this.Command.Parameters.AddWithValue(paramName, param);

}

public void addParameter(string paramName, int param)

{

// If the connection isn't opened yet, open it before adding parameters

if (this.Command == null) this.openDB();

this.Command.Parameters.AddWithValue(paramName, param);

}

public void addParameter(string paramName, double param)

{

// If the connection isn't opened yet, open it before adding parameters

if (this.Command == null) this.openDB();

this.Command.Parameters.AddWithValue(paramName, param);

}

public void addParameter(string paramName, DateTime param)

{

// If the connection isn't opened yet, open it before adding parameters

if (this.Command == null) this.openDB();

this.Command.Parameters.AddWithValue(paramName, param);

}

public void addParameter(SqlParameter param)

{

// If the command doesn't exist yet, create it before adding parameters

if (this.Command == null) this.openDB();

if (this.Command.CommandType != CommandType.StoredProcedure)

this.Command.CommandType = CommandType.StoredProcedure;

this.Command.Parameters.Add(param);

}

public int dbVal(string sql)

{

this.sql = sql;

this.openDB();

this.executeSQL();

int data;

try

{

if (this.Command.ExecuteScalar() == null)

data = -1;

else

data = int.Parse((this.Command.ExecuteScalar().ToString()));

}

catch (Exception e)

{

data = -1;

error = e.Message;

}

this.closeDB();

return data;

}

public string dbString(string sql)

{

this.sql = sql;

this.openDB();

this.executeSQL();

string data;

try

{

if (this.Command.ExecuteScalar() == null)

data = "";

else

data = (this.Command.ExecuteScalar().ToString());

}

catch (Exception e)

{

data = null;

error = e.Message;

}

this.closeDB();

return data;

}

public SqlDataReader dbReader(string SQLProc, CommandType SQLType)

{

if (SQLType == CommandType.StoredProcedure)

InitializeProc(SQLProc);

else

{

this.openDB();

this.Command.CommandText = SQLProc;

}

this.Command.CommandType = SQLType;

myReader = this.Command.ExecuteReader();

return myReader;

}

public void CloseReader()

{

if (!myReader.IsClosed)

{

myReader.Close();

myReader.Dispose();

myConnection.Close();

}

}

public DataTable dbTable(string sql)

{

InitializeProc(sql);

DataTable data = new DataTable();

SqlDataReader reader;

try

{

reader = dbReader(sql, CommandType.StoredProcedure);

data.Load(reader);

}

catch (Exception e)

{

data = null;

error = e.Message;

}

this.closeDB();

return data;

}

public DataTable Pivot(string sql, string keyColumn, string pivotNameColumn, string pivotValueColumn, List<string> myCols)

{

this.pivotCols = myCols;

DataTable data = Pivot(sql, keyColumn, pivotNameColumn, pivotValueColumn);

this.pivotCols = null;

return data;

}

public DataTable Pivot(string sql, string keyColumn, string pivotNameColumn, string pivotValueColumn)

{

DataTable tmp = new DataTable();

DataRow r;

string LastKey = "//dummy//";

int i, pValIndex, pNameIndex;

string s;

bool FirstRow = true;

this.sql = sql;

this.openDB();

this.executeSQL();

SqlDataReader dataValues;

try

{

dataValues = this.Command.ExecuteReader();

// Add non-pivot columns to the data table:

pValIndex = dataValues.GetOrdinal(pivotValueColumn);

pNameIndex = dataValues.GetOrdinal(pivotNameColumn);

for (i = 0; i <= dataValues.FieldCount - 1; i++)

if (i != pValIndex && i != pNameIndex)

tmp.Columns.Add(dataValues.GetName(i), dataValues.GetFieldType(i));

if (pivotCols is object)

{

foreach (string cName in pivotCols)

tmp.Columns.Add(cName);

}

r = tmp.NewRow();

// now, fill up the table with the data:

while (dataValues.Read())

{

// see if we need to start a new row

if (dataValues[keyColumn].ToString() != LastKey)

{

// if this isn't the very first row, we need to add the last one to the table

if (!FirstRow)

tmp.Rows.Add(r);

r = tmp.NewRow();

FirstRow = false;

// Add all non-pivot column values to the new row:

for (i = 0; i <= dataValues.FieldCount - 3; i++)

r[i] = dataValues[tmp.Columns[i].ColumnName];

LastKey = dataValues[keyColumn].ToString();

}

// assign the pivot values to the proper column; add new columns if needed:

s = dataValues[pNameIndex].ToString();

if (!tmp.Columns.Contains(s))

tmp.Columns.Add(s, dataValues.GetFieldType(pValIndex));

r[s] = dataValues[pValIndex];

}

// add that final row to the datatable:

tmp.Rows.Add(r);

// Close the DataReader

dataValues.Close();

// and that's it!

}

catch (Exception e)

{

tmp = null;

error = e.Message;

}

this.closeDB();

return tmp;

}

public void ClearParameters()

{

// Clears all parameters from the command object

this.isProc = false;

this.Command.Parameters.Clear();

}

private void InitializeProc(string procedureName)

{

this.isProc = true;

this.timeout = 0;

// If the command doesn't exist yet, create it before adding parameters

if (this.Command == null)

this.openDB();

// If the connection isn't opened yet, open it

else if (this.Command.Connection.State != ConnectionState.Open)

this.Command.Connection.Open();

this.Command.CommandType = CommandType.StoredProcedure;

this.Command.CommandText = procedureName;

}

public void executeProc(string procedureName)

{

InitializeProc(procedureName);

try

{

this.Command.ExecuteNonQuery();

}

catch (SqlException e)

{

error = e.Message;

}

this.closeDB();

}

public int execProc(string procedureName)

{

InitializeProc(procedureName);

// Add a SQL parameter to capture the return value

SqlParameter returnValue = new SqlParameter("@returnVal", SqlDbType.Int);

returnValue.Direction = ParameterDirection.ReturnValue;

this.Command.Parameters.Add(returnValue);

try

{

this.Command.ExecuteNonQuery();

}

catch (SqlException e)

{

error = e.Message;

}

this.closeDB();

// Return the result of the stored procedure

return Convert.ToInt32(returnValue.Value);

}

public Tuple<int, string, string> execMultipleOutputProc(string procedureName)

{

InitializeProc(procedureName);

// Add a SQL parameter to capture the return value

SqlParameter returnValue = new SqlParameter("@returnVal", SqlDbType.Int);

returnValue.Direction = ParameterDirection.ReturnValue;

this.Command.Parameters.Add(returnValue);

SqlParameter returnString = new SqlParameter("@retString", SqlDbType.NVarChar, 200);

returnString.Direction = ParameterDirection.Output;

this.Command.Parameters.Add(returnString);

SqlParameter returnNextApprover = new SqlParameter("@returnNextApprover", SqlDbType.NVarChar, 1000);

returnNextApprover.Direction = ParameterDirection.Output;

this.Command.Parameters.Add(returnNextApprover);

try

{

this.Command.ExecuteNonQuery();

}

catch (SqlException e)

{

error = e.Message;

}

this.closeDB();

// Return the result of the stored procedure

return Tuple.Create(Convert.ToInt32(returnValue.Value), Convert.ToString(returnString.Value), Convert.ToString(returnNextApprover.Value));// Convert.ToInt32(returnValue.Value);

}

public string execOutProc(string procedureName)

{

InitializeProc(procedureName);

// Add a SQL parameter to capture the return value

SqlParameter returnValue = new SqlParameter("@Message", SqlDbType.VarChar);

returnValue.Direction = ParameterDirection.Output;

returnValue.Size = Int32.MaxValue;

this.Command.Parameters.Add(returnValue);

try

{

this.Command.ExecuteNonQuery();

}

catch (SqlException e)

{

error = e.Message;

}

this.closeDB();

// Return the result of the stored procedure

return returnValue.Value.ToString();

}

public int execOutIntProc(string procedureName)

{

InitializeProc(procedureName);

// Add a SQL parameter to capture the return value

SqlParameter returnValue = new SqlParameter("@InsertedId", SqlDbType.Int);

returnValue.Direction = ParameterDirection.Output;

this.Command.Parameters.Add(returnValue);

try

{

this.Command.ExecuteNonQuery();

}

catch (SqlException e)

{

error = e.Message;

}

this.closeDB();

// Return the result of the stored procedure

if (returnValue.Value != DBNull.Value)

return Convert.ToInt32(returnValue.Value);

else

return 0;

}

public void exec(string sql)

{

this.timeout = 0;

this.sql = sql;

this.openDB();

this.executeSQL();

try

{

this.Command.ExecuteScalar();

}

catch (Exception e)

{

error = e.Message;

}

this.closeDB();

}

public List<string> dbArray(string sql)

{

List<string> data = new List<string>();

try

{

this.sql = sql;

this.openDB();

this.executeSQL();

this.myReader = Command.ExecuteReader();

while (this.myReader.Read())

{

for (int y = 0; y < this.myReader.FieldCount; y++)

{

data.Add(myReader[y].ToString());

}

}

myReader.Close();

this.closeDB();

}

catch (Exception e)

{

error = e.Message;

}

return data;

}

public DataTable getValues(string procedureName)

{

InitializeProc(procedureName);

DataTable dt = new DataTable();

SqlDataReader reader;

reader = this.Command.ExecuteReader();

dt.Load(reader);

reader.Close();

reader.Dispose();

this.closeDB();

return dt;

}

public DataTable getDBValues(string procedureName)

{

InitializeProc(procedureName);

DataSet ds = new DataSet();

DataTable dt = new DataTable();

ds.Tables.Add(dt);

ds.EnforceConstraints = false;

SqlDataReader reader;

reader = this.Command.ExecuteReader();

dt.Load(reader);

reader.Close();

reader.Dispose();

this.closeDB();

return dt;

}

public DataSet getDataSetValues(string procedureName)

{

InitializeProc(procedureName);

DataSet ds = new DataSet();

SqlDataAdapter adapter = new SqlDataAdapter(this.Command);

adapter.Fill(ds);

adapter.Dispose();

this.closeDB();

return ds;

}

public void dbDict<T>(string SQL, ref T dicResults)

where T : IDictionary

{

this.sql = SQL;

this.openDB();

this.executeSQL();

SqlDataReader reader;

try

{

reader = this.Command.ExecuteReader();

string firstKey = "";

List<string> valueArray = new List<string>();

// Loop thru records. Add records to the dictionary, typing the first column as appropriate.

while (reader.Read())

{

if (dicResults is Dictionary<string, string>)

dicResults.Add(reader[0].ToString(), reader[1].ToString());

else if (dicResults is Dictionary<string, List<string>>)//Assumes Keys are ordered!

{

if (firstKey != reader[0].ToString())

{

if (firstKey != "")

dicResults.Add(firstKey, valueArray);

firstKey = reader[0].ToString();

valueArray = new List<string>();

}

valueArray.Add(reader[1].ToString());

}

else if (dicResults is Dictionary<int, string>)

dicResults.Add((int)reader[0], reader[1].ToString());

}

if (firstKey != "") //Add last one

{

dicResults.Add(firstKey, valueArray);

}

reader.Close();

reader.Dispose();

}

catch (Exception e)

{

error = e.Message;

}

this.closeDB();

}

#region BulkInsert

public void BullInsert()

{

}

#endregion

#region basic db stuff

private void openDB()

{

this.myConnection = new SqlConnection(constr);

this.myConnection.Open();

this.Command = new SqlCommand(this.sql, this.myConnection);

}

private void executeSQL()

{

Command.CommandTimeout = this.timeout;

if (this.isProc)

Command.CommandType = CommandType.StoredProcedure;

else

Command.CommandType = CommandType.Text;

}

private void closeDB()

{

this.myConnection.Close();

}

#endregion

}

}