```
using System;
using System.IO;
using System.Collections.Generic;
using System.Data;
using System.Data.SQLite;
using System.Reflection;
using System.Runtime.InteropServices;

using SMath.Manager;
using SMath.Math;
using SMath.Math.Numeric;

namespace SQLitePlugin
{
    public class SQLitePlugin : IPluginLowLevelEvaluationFast
    {

        #region Private fields

        private List<SQLiteConnection> Connections = new List<SQLiteConnection>();

        #endregion

        #region Private methods

        private SQLiteConnection Attach( string dbpath )
        {
            var connectionString = "Data Source=" + dbpath + ";Version=3;FailIfMissing=True";

            var connection = Connections.Find( x => x.ConnectionString == connectionString );

            if ( connection == null )
            {
                connection = new SQLiteConnection( connectionString );

                connection.Open();

                if ( connection.State == ConnectionState.Open )
                {
                    Connections.Add( connection );

                    return connection;
                }

                throw new Exception( "Failed to open database: " + dbpath );
            }

            if ( connection.State != ConnectionState.Open )
            {
                connection.Open();

                if ( connection.State != ConnectionState.Open )
                {
                    throw new Exception( "Failed to open database: " + dbpath );
                }

                return connection;
            }

            return connection;
        }

        #endregion

        #region Public methods

        public TermInfo[] GetTermsHandled( SessionProfile sessionProfile )
        {
            return new[]
            {
                new TermInfo(
                    "SQLiteQuery",
                    TermType.Function,
                    "(path, query) - Opens the specified database file if necessary, executes the given query↙
```

```
    and returns the results.",
                FunctionSections.Unknown,
                true,
                new ArgumentInfo( ArgumentSections.String, true ),
                new ArgumentInfo( ArgumentSections.String, true )
            )
        };
    }


    public void Initialize()
    {
        var is64Bit = Marshal.SizeOf( typeof( IntPtr ) ) == 8;

        try
        {
        //TODO:...
        }
        catch {}
    }


    public bool TryEvaluateExpression( Entry value, Store context, out Entry result )
    {
        result = null;

        if ( value.Type != TermType.Function )
        {
            return false;
        }

        if ( value.Text == "SQLiteQuery" && value.ArgsCount == 2 )
        {
            var arg1 = Computation.Preprocessing( value.Items[0], context );
            var arg2 = Computation.Preprocessing( value.Items[1], context );

            var answer = new List<Term>();

            var path = arg1.ToString().Trim( '"' );

            path = Environment.ExpandEnvironmentVariables( path );

            if ( !string.IsNullOrEmpty( context.FileName ) && !Path.IsPathRooted( path ) )
            {
                path = Path.Combine( Path.GetDirectoryName( context.FileName ), path );
            }

            var connection = Attach( path );

            var query = new SQLiteCommand( arg2.ToString().Trim( '"' ), connection );

            var reader = query.ExecuteReader();

            var nr = 0;
            var nc = 0;

            while ( reader.Read() )
            {
                nc = reader.FieldCount;

                for ( var n = 0; n < nc; n++ )
                {
                    var type = reader.GetFieldType(n);

                    try
                    {
                        if ( type == typeof( DBNull ) )
                        {
                            answer.AddRange( TermsConverter.ToTerms( "'NULL" ) );
                        }
                        else if ( type == typeof( string ) )
                        {
                            answer.AddRange( TermsConverter.ToTerms( "\"" + reader[n] + "\"" ) );
                        }
```

```
                            else
                            {
                                var val = ( double ) reader[n];

                                answer.AddRange( new TNumber( val ).obj.ToTerms() );
                            }
                        }
                        catch
                        {
                            answer.AddRange( TermsConverter.ToTerms( "\"undef\"" ) );
                        }
                    }

                    nr++;
                }

                answer.AddRange( TermsConverter.ToTerms( nr.ToString() ) );
                answer.AddRange( TermsConverter.ToTerms( nc.ToString() ) );

                if ( nc == 0 && nr == 0 )
                {
                    answer.Add( new Term( Functions.Matrix, TermType.Function, 2 ) );
                }
                else
                {
                    answer.Add( new Term( Functions.Mat, TermType.Function, 2 + nc * nr ) );
                }

                result = Entry.Create( answer.ToArray() );

                return true;
            }

            return false;
        }


        public void Dispose()
        {
            for( var n = 0; n < Connections.Count; n++ )
            {
                Connections[n].Close();
            }
        }

        #endregion

        #region Properties

        public static string CurrentDir
        {
            get
            {
                return Path.GetDirectoryName( new Uri( Assembly.GetExecutingAssembly().CodeBase ).LocalPath )↙
;
            }
        }

        public static string SQLite
        {
            get { return Path.Combine( CurrentDir, "System.Data.SQLite.dll" ); }
        }

        public static string SQLite32
        {
            get { return Path.Combine( CurrentDir, "System.Data.SQLite.x86.dll" ); }
        }

        public static string SQLite64
        {
            get { return Path.Combine( CurrentDir, "System.Data.SQLite.x64.dll" ); }
        }

        #endregion
```

```
    }
}
```