**ViSo-Nice ERD**

Thank you for using ViSo-Nice ERD. I hope that it serves you well.

**Purpose**

The purpose of this application is to provide a visual representation of a database, where the relations are not necessarily registered as constraints, and to manage your database and provide a code base scripting for general queries.

**Design and Function**

The ERD application is designed for maximum visibility of table structures, with a Tab canvas to allow for grouping of tables by logical functioning.

Current version supports MS SQL database, and C# scripting output. With your assistance we may extend the functionality to include more databases and coding languages.

Projects are saved as several separated Json text files. Each Tab Canvas is saved separately to allow developers to work on the same database project without merge issues. Note that file lock is not implemented and is part of further development.

Filenames are structure with the Project name and an extension. Tabs contains the tab name as peart of the file name. File extensions are:

* \*.eprj – Project file.
* \*.estp – Code Base Scripting file.
* \*.eclu – Canvas object containing the tables on the canvas.

Note that only tables on the canvas are updated and worked with. Tables not on the canvas are only listed in the tables list.

**Getting Started** (Download the ViSo-Nice sample projects files for guidance)

**File -> New Project.**

Creates a new project file.

**Project Setup**

* Project Name – The name of the project that represents the database.
* File Directory – The fil directory where the project file and all the related files will be stored. Note that project files may share a directory if the names of the projects differ.
* Allow Database Relations – Allows developers to add new database relations. These relations will be created in the database. This is used for strongly relational database models, and are represented on the canvas as a blue line
* Allow Virtual Relations – Allows developers to add a relation that is loosely coupled to the table structure and will not be implemented in your database. These relations do not enforce database rules, are only for visual representations, and are represented as a green line on the canvas.
* Keep Columns Unique – This will assist in ensuring that column types are unique through out the database. E.g. MyColumn will be represented as MyColumn with the same datatype throughout the database.

**Database Setup**

* Database Type – Select the database type to use for the project.
* The rest is rather straight forward.

Buttons

* Accept – Saves the project and closes the window.
* Add Alternative Connections – Allows alternative connections to other databases. This allows developers to have a development, or range of development databases, and compare these in the same model without the need to change the connection information. Note that the Username and Password is not required for alternative connections. This is to allow developers the add a connection to a production database. Any action against this connection will require the user to provide the username and password before the action will execute. The username and password will be stored in memory for the duration of the application lifetime.

**File -> Open Project.**

Opens and existing project.

**File -> Save Project**

Saves the project.

**Project -> Edit Project**

Opens the Project Setup for editing

**Project -> Add Project Canvas**

Opens the Tab setup Window, to add a new project canvas to the project.

Canvases are pan-able by pressing and holding the left mouse button, and then dragging the mouse.

Canvases are zoomed by using the mouse wheel.

Fields

* Tab Name – The name to be displayed as the Tab name. Remember the tab name will be used as the file name for the tab, so no special characters.
* Table Prefix – Leave empty if tables for this tab are not to be prefixed. For existing database models where prefixing of tables was not done this option should be left empty, as this will enforce the prefix to each table.

**Database -> Get DB Tables**

Reads all the tables from the database, not the columns, as columns will only be read once the table is dropped on a canvas and populates the ‘Database Tables’ list to the left of the application.

This list contains the Table names, first the tables on the canvas, with the canvas name, and then the tables not placed on a canvas.

Note that tables in the list can be placed only on one canvas. Tables not on a canvas can be dragged to the required canvas.

By double clicking on a table in the list that have a canvas name next to id will direct the application the that canvas and try to zoom to the table.

**Database -> Refresh canvases from DB.**

Forces changes made in the database to the tables placed on canvases.

**Database -> Compare to DB**

Compares the database and the tables on canvases to each other and opens the results in the comparison results window. From here actions can be edited for action and a script file can be generated for editing.

**Database -> Forward Engineer**

Forward engineers the ERD Model and the tables placed on canvases to the database. Note that only tables placed on canvases are used.

**Database -> Script Changes**

The name is rather confusing, as this scripts the ERD Model, in its entirety, including changes to a \*.sql file.

**Build -> Script Builder Setup**

Opens the Scrip Builder Setup Window.

Note this is a rather big section and is best explained by sample. The Sample project files contains a rather extensive example of how the script builder set can be done.

Basics are:

Toe the left is a list of parameters that can be used in your build.

Each file is represented a Tab, but all Tab’s are saved in one file, so if you don’t want to build a file anymore, remove id by closing it and save your changes.

To the right is displayed how your code will look based on the first table, of the first canvas on your model.

Each File Tab can consist of several build Types. The Build types are executed in the order they are added. Subsequent build types can be referenced in any preceding build type by adding a parameter [[x]] in the preceding type matching the number of the subsequent type.

All parameters must be encapsulated by double square brackets e.g. [[TableName]].

The Repeat Options will determine how many times the Build Type will execute.

The parameter [[?\*]] can be used for repetitive characters by replacing the \* with the desired character or string. E.g. building a coma delimiter for column names will look like this [[?,]]. For an example see the Sample Projects ‘Mapping File’ tab, Build Types for ‘Body’ and ‘Primary Key Mapping’

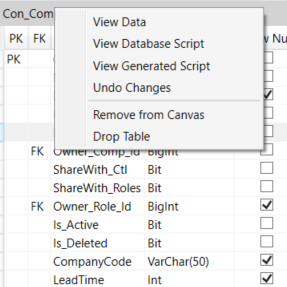
**Build -> Run Script Builder**

Creates the files defined in the script builder setup

**Table Objects**

The most important to know about the table Object is that the Table Header and Selected Column provides its own Context Menu Items. Each specific to its object type.

Table Context Menu



Column Context Menu

