

JULADB – AN EMBEDDED DATABASE ENGINE

This software is an implementation of an embedded database engine. It was written entirely in C# language for the .NET 4.0 platform. Project was created as a part of author's master's thesis.

FEATURES

The features of the JuliaDB relational database management system consist of:

- A fully functional SQL engine, with parser, query planner, query optimizer and plan executor.
- Storing data in memory with an option to store it in on a hard drive.
- Expression evaluator, supporting literals, variables, operators, scalar functions and aggregate functions.
- Support for 3 data types: varchar (string), number and boolean.
- Support for a subset of the SQL language, including:
 - Queries, with SELECT, FROM, WHERE, GROUP BY and ORDER BY clauses,
 - Subset of Data Definition Language, including CREATE TABLE and DROP TABLE,
 - Subset of Data Manipulation Language, with UPDATE and DELETE statements.
- Support for column and table aliases, LIKE operator, NULL values.
- Proper error handling – error codes and messages are returned in case of any issues during query processing.
- A simple query optimizer, able to improve query plans to use the hash-join algorithm.
- ADO.NET provider, with Connection, Command, DataReader and DataAdapter implementations.
- LINQ provider, able to translate strongly-typed LINQ queries to SQL queries understood by JuliaDB.
- JuliaDB Navigator – GUI giving the ability to execute any SQL statements on the database and read the results.

REQUIREMENTS

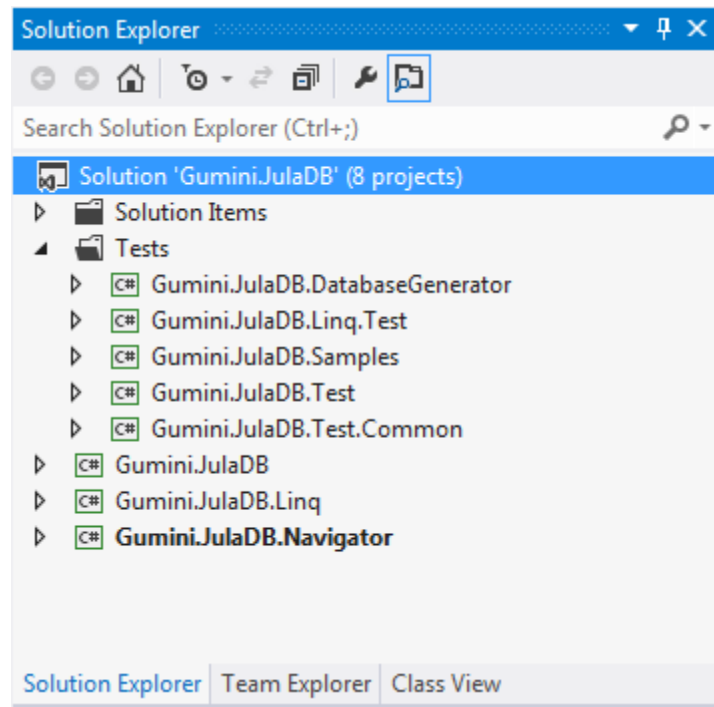
To be able to compile the solution and run the tests, you should have Windows with either Visual Studio 2010 or Visual Studio 2012 installed. All required dependencies and 3rd party libraries are included with the source code.

To build the full source code, the following steps are required:

1. Go to <http://juladb.codeplex.com/SourceControl/latest> and click the Download button to get the latest source, zipped.
2. Unzip and double-click the .sln file.
3. Visual Studio may prompt to connect to source control. This isn't necessary to compile the projects or run the tests, so these messages can be safely skipped.
4. Build the solution either by clicking Build -> Build Solution or pressing F6. This should results in a clean build, with no errors.

STRUCTURE

JuliaDB consists of several projects. All projects are contained within a single solution Gumini.JuliaDB.sln. This is the project tree, as displayed by VS2012 Solution Explorer:



All projects, namespaces and assemblies are named in a consistent manner and are prefixed with Gumini.JulaDB.

The main components of JulaDB are:

- **Gumini.JulaDB** – this is a DLL that contains the core database functionality. It is responsible for all steps of query processing. Additionally, it contains an implementation of ADO.NET provider as a standard way of communicating with JulaDB. Implementation of all ADO.NET components can be found in Gumini.JulaDB.Provider namespace. This DLL should be used by all applications willing to communicate with JulaDB.
- **Gumini.JulaDB.Linq** – this is a DLL with an implementation of LINQ provider for JulaDB. It is an optional component that can be used by clients that want to issue strongly-typed LINQ queries to JulaDB RDBMS.
- **Gumini.JulaDB.Navigator** – an executable desktop application written in WPF (Windows Presentation Foundation). It enables the user to execute SQL statements in JulaDB without having to write a custom program using ADO.NET or LINQ.

There's also a set of components for testing purposes. They are stored in the "Tests" folder in the solution tree.

- **Gumini.JulaDB.Test** – contains a set of automated tests that cover functionality from the main Gumini.JulaDB assembly. Tests were written using the Visual Studio built-in test framework.
- **Gumini.JulaDB.Linq.Test** – assembly with tests that cover the functionality of LINQ provider to JulaDB. Similarly to the main test assembly, tests were written using the Unit Testing Framework.
- **Gumini.JulaDB.Test.Common** – contains shared utility classes and methods used by both main and LINQ test assemblies.
- **Gumini.JulaDB.DatabaseGenerator** – a console application used to automatically generate a database and fill it with random data. The database has a simple Customer-Order schema. Number of rows to generate in each of these tables is specified as a parameter. This utility is useful in testing of large data and JOIN performance.

- **Gumini.Juladb.Samples** – a console application containing the code samples used in master’s thesis.

EXTERNAL DEPENDENCIES

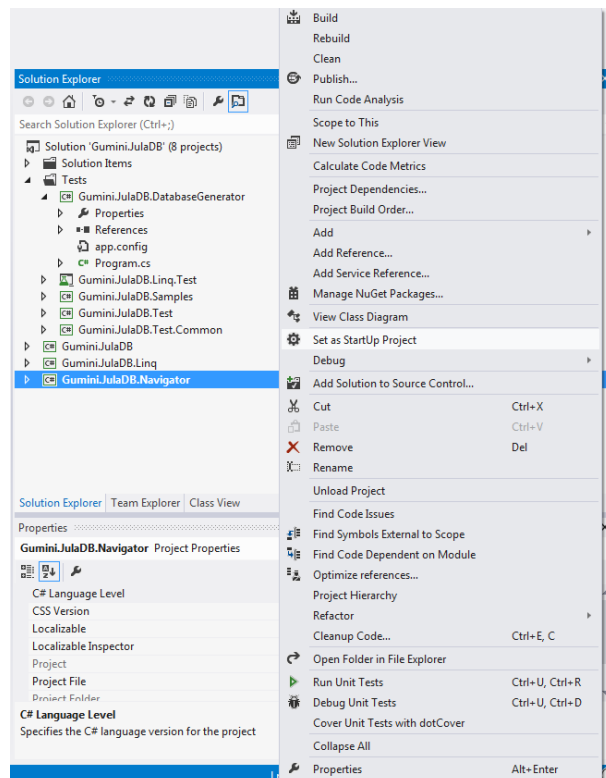
Projects rely on a few external components. All are included with the source code as compiled assemblies. Where possible, references were added using NuGet package manager.

- **Irony - .NET Language Implementation Kit** – used for parsing SQL (<http://irony.codeplex.com/>).
- **Remotion.Linq** – a helper library used to simplify creation of LINQ providers.
- **RandomNameGenerator** – used by DatabaseGenerator to fill the Customer table with random names (<http://www.markdavidrogers.com/oxitesample/Blog/random-name-generator-net-library>).
- **GalaSoft.MvvmLight.WPF4** – a helper library for implementing the MVVM pattern in Juladb Navigator.
- **ICSharpCode.AvalonEdit** – a text editor component used by Juladb Navigator.

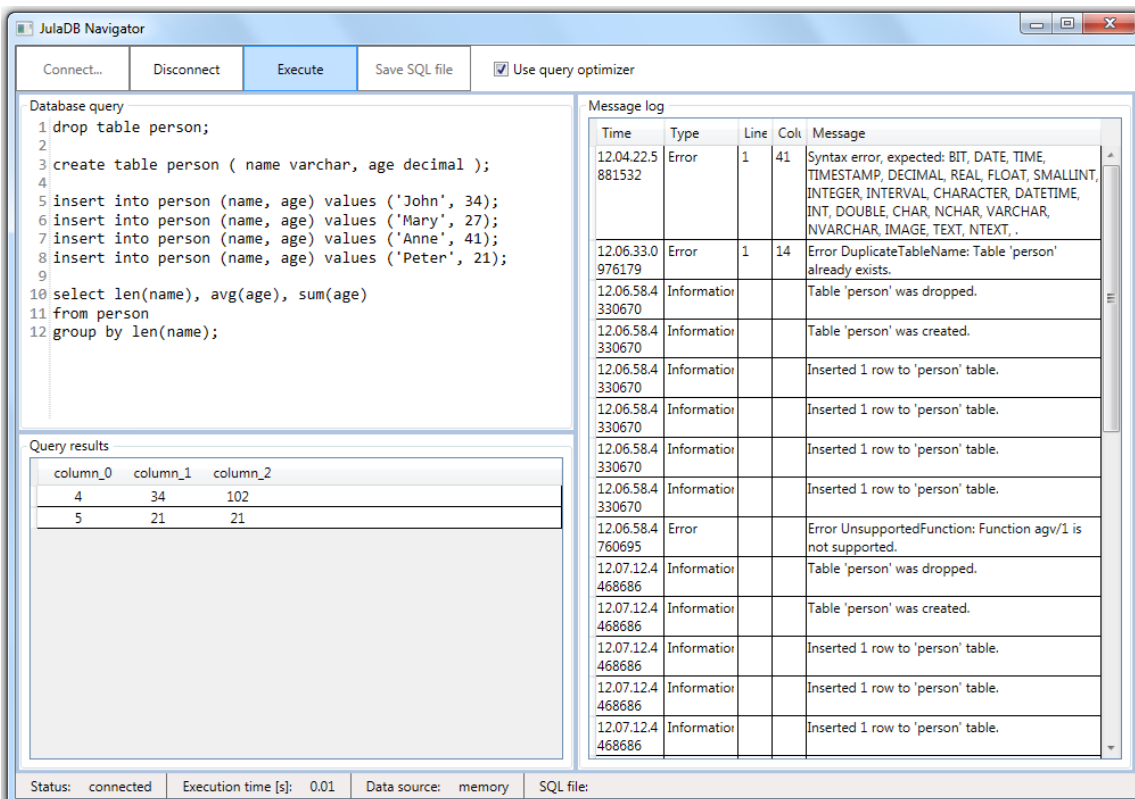
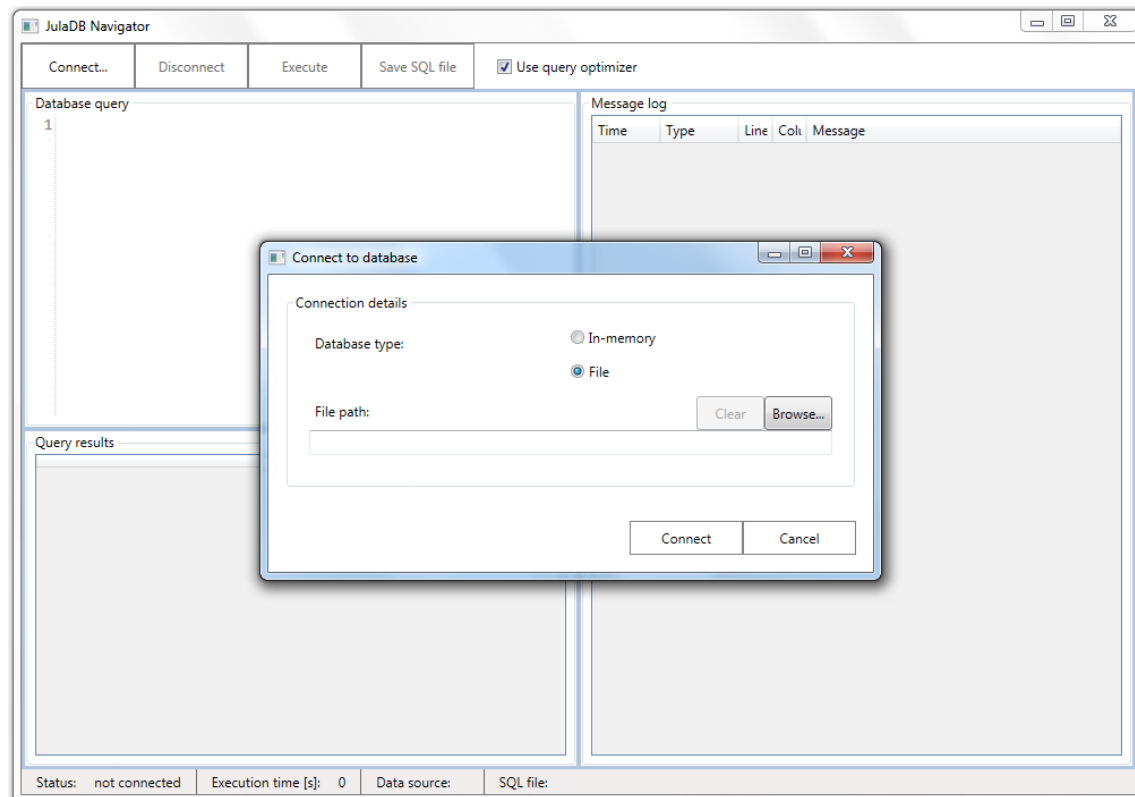
USING JULADB NAVIGATOR

Juladb comes with a graphical user interface application called Juladb Navigator. It can be used to establish connection with the database, write and execute SQL statements and read the results. It will display query results in a graphical table and show information like number of rows affected by the DML statement and error messages in a message log.

To run the application, you should run the Gumini.Juladb.Navigator.exe file which can be found in Gumini.Juladb.Navigator\bin\Debug or Gumini.Juladb.Navigator\bin\Release folder (depending on a mode in which the solution was built). Alternatively, you can right click on the Gumini.Juladb.Navigator in the Solution Explorer window, select “Set as StartUp project” and press F5 to start the application in the Debug mode.



The screenshots below show the sample use of this application.



RUNNING THE TESTS

Two assemblies included in the solution contain a set of automated tests. All were implemented using the build-in Visual Studio testing framework. At the time of writing, there are 118 tests in total:

- Gumini.JulaDB.Test – 96 tests
- Gumini.JulaDB.Linq.Test – 22 tests

To run the tests in Visual Studio 2012:

1. Ensure that the solution was fully built
2. View the Test Explorer window, which should display all test for both assemblies
3. Press “Run All”
4. Wait for all tests to execute (it shouldn’t take longer than half a minute) and read the results. All tests should pass.

Vast majority of tests operate on an in-memory database. One of the tests (FileDatabaseTest.CreateNewFileDatabase) creates a file database to confirm file operations work properly. Therefore, a write access to the current test directory is required, otherwise this test will fail.

