ZimaDB - Technical Documentation

Vilém Zouhar, Petr Chmel

January 2019

1 General

1.1 Introduction

ZimaDB aims to provide a full and robust implementation of a specific subset of the SQLite database. For more general info please visit github.com/zouharvi/zimadb, which serves also as the project landing page.

1.2 Environment

ZimaDB is programmed from scratch in C++. The complete source code and additional files can be found at github.com/zouharvi/zimadb. We also use TravisCI for continuous integration.

1.3 Development pipeline

1.3.1 Linux/Debian

The main target is the Linux/Debian build. We use a custom makefile, which should in theory work on all Linux distributions with basic development packages installed (gcc, make). The makefile is located in the root directory and contains several object dependencies (standard makefile procedure), so that the whole project does not need to be recompiled every time.

1.3.2 Microsoft Windows

Apart from one external library, the C++ code should be 100% portable. The only issue with running the makefile on Windows (with for example Clang) is that it uses some Bash commands. This could be solved by running the makefile on MinGW, but we opted for CMake, which creates a Visual Studio solution (targets VS2015), which in turn invokes MSBuild and the code ends up being processed with *cl*.

1.4 External libraries

The goal was to create as portable implementation as possible. During development, however, we found that for comfortable usage of ZimaDB (command history, line edits, etc..) it is necessary to use a third-party library. We opted for the GNU/Readline library (en.wikipedia.org/wiki/GNU_Readline) as it is freely distributed under the GNU license and has been tested on many systems. On MS Windows build the usage of this library is disabled.

2 Functionality

2.1 Data types

The following data types are supported by ZimaDB.

- BOOLEAN
- (UNSIGNED) INT, TINYINT
- VARCHAR(n) (if no size specified, 32 is used)
- DOUBLE
- PRIMARY KEY attribute
- ASC, DESC attribtute

2.2 Allowed operations

The following operations are supported by ZimaDB.

- SELECT
- INSERT
- UPDATE
- DELETE
- CREATE TABLE
- DROP TABLE
- TRUNCATE TABLE

Aggregation functions are not permitted as well se nested SELECT statements. This is because the implementation is beyond the size of this project and requires advanced knowledge of databases. The JOIN ON operation can be simulated using selects on multiple tables. We don't support the table dot notation, so any multiple select must be done on distinct tables.

2.3 Other modifiers

Several other modifiers are supported as well:

- PRIMARY KEY attribute
- ASC, DESC attribtute
- wildcard asterisk (SELECT)

3 Modules

The whole project is split into three main modules: *core, engine, share*. The first goal was to keep them as separate as possible (even to the point of having the whole project be just a result of static linking of these three modules). This, however, became a great obstacle in development and all of the source code was joined (although kept in separate folders).

3.1 Core

This module contains the Tokenizer, Parser, Grammar processor and the Core interface. In general Core is responsible for the frontend and correct parsing of the input for the Engine. Core contains the program entry point.

3.2 Share

Share is used for communication between Core and Engine. It contains objects representing various queries as well as some utility classes.

3.3 Engine

Engine operates mainly on files. It is responsible for correctly interpreting the file format and processing queries from Share. It uses Pager with B-Trees.

3.3.1 Paging and file format

Paging is rather simple and the file format, which was created solely for this project, is somewhat inefficient in the space it takes when compared to other databases, such as SQLite. Namely, the header of the file takes up a whole page, the table definition of any table takes up at least one page, too.
However, data pages were designed to take the least space possible, so they only contain the data inserted by user as the additional data regarding the table is found in the table definition page.
The page size is currently 4096 bytes and is set as a constant.

3.3.2 B-Tree

B-Tree implementation is technically a (5,10)-tree which uses preemptive splitting.

4 Development process

4.1 Testing

The makefile contains a basic testing suite for various inputs, that are located in *test/scripts*. Use *make test* to run all test. Finally we tested the program with Valgrind to check for any memory leaks.

4.2 Issue tracking

Some issues which needed to be coordinated were posted on github.com/zouharvi/ZimaDB/issues.

5 Code statistics

Using the CLOC program we analyzed our code with the following results:

Language	files	blank	comment	code
C++	26	496	115	3136
C/C++ Header	27	182	74	711
Bourne Shell	14	22	0	131
Markdown	4	37	0	124
JSON	2	0	0	62
make	1	13	1	59
YAML	2	2	0	27
Dockerfile	1	5	3	10
CMake	1	5	0	5
SUM:	78	762	193	4265

This totals to 5220 lines of code in the repository combined. The total size of the source files (in UTF8) is 134.066 KB.