

# CoGaDB User Guide

Version 0.4.1

Sebastian Breß

TU Dortmund University

sebastian.bress@tu-dortmund.de

## Prerequisites

- Operating System: Ubuntu Version 12.04.3 LTS or higher (64 bit)
- Compiler: g++ version 4.6 or higher
- Tools
  - NVIDIA CUDA Toolkit 5.0 or higher
  - cmake version 2.6 or higher
  - make
  - Doxygen (Documentation Generation)
  - Pdf Latex (Documentation Generation)
  - Bison (supported from version 2.5)
  - Flex
  - Mercurial (version control system)
- Libraries:
  - Boost Libraries version 1.48 or higher
  - Readline Library
  - Intel TBB Library
  - BAM Library (optional, required for Genomics Extension)
  - XSLT Library (Documentation Generation)
- Hardware
  - Intel CPU (optional, required for using Intel PCM Library)
  - NVIDIA GPU (optional, required for GPU acceleration)

# Installation Steps

First you need to install all libraries specified in the prerequisites section. In case you run an up to date Ubuntu, you can use the following commandline:

```
sudo apt-get install gcc g++ make cmake flex bison libtbb-dev libreadline6 libreadline6-dev doxygen doxygen-gui graphviz texlive-bibtex-extra texlive-fonts-extra texlive-fonts-extra-doc ghostscript texlive-fonts-recommended xsltproc libxslt1-dev nvidia-cuda-toolkit sharutils
```

To install boost, you can install it with the following command line:

```
sudo apt-get install libboost-filesystem-dev libboost-system-dev libboost-thread-dev libboost-program-options-dev libboost-serialization-dev libboost-chrono-dev libboost-date-time-dev libboost-random-dev libboost-iostreams-dev
```

Now we are able to compile CoGaDB. Enter the main directory `gpubms` in a terminal. We now have to generate a build system using `cmake`. It is considered good practice to separate the source tree from the separated build system. Therefore, we will generate the build system into a sub directory.

Development is often easier if we have compiled the source code with debugging symbols and without optimization. On the other hand, we need to compile source with optimizations and without debugging symbols to achieve maximal performance. Therefore, we will generate two build systems. One builds CoGaDB in debugging mode, whereas the other builds CoGaDB in release mode. You can enter the following commands to achieve this:

```
mkdir debug_build
cd debug_build
cmake -DCMAKE_BUILD_TYPE=Debug ..
make
cd ..
mkdir release_build
cd release_build
cmake -DCMAKE_BUILD_TYPE=Release ..
make
cd ..
```

Now, `debug_build` contains the build system in debug mode, whereas the build system in `release_build` compiles CoGaDB in release mode. Just enter a directory, hit `make`, and launch using `./cogadb/bin/cogadbd`. Since CoGaDB consists of many source files, you can speedup the build process by instructing `make` to compile multiple source files in parallel. You can do this by

using the `-j` option of `make`, where you specify the number of parallel build processes, which is typically your number of CPU cores.

That's it, you can now start developing CoGaDB!

## Configuration of the Cmake Build System Generator

You can influence the behavior of `cmake` by adjusting the Variables in the `CMakeCache.txt`. You can also set a variable using the commandline interface. Let us assume we want to change the build type of our build system from Debug to Release. The build type is stored in the variable `CMAKE_BUILD_TYPE`. Now, we can change the build type using the following command:

```
cmake -DCMAKE_BUILD_TYPE=Release ..
```

There are many variables that allow customization. We list here the most important ones:

`ENABLE_SIMD_ACCELERATION`

`ENABLE_BRANCHING_SCAN`

`CMAKE_VERBOSE_MAKEFILE`

`CMAKE_CXX_FLAGS_RELEASE`

`CMAKE_CXX_FLAGS_DEBUG`

## Setting up a Database and Issue Queries

At first, we have to create a directory, where CoGaDB can store its database:

```
set path_to_database=/home/DATA/coga_databases/ssb_sf1
```

Then, we have to create a database and import data. This can be done in two ways: using the `sql` interface (`create table`, `insert into`), or using a utility command. CoGaDB supports utility commands for importing databases from two common OLAP benchmarks: the TPC-H and the Star Schema Benchmark. Note that you have to generate the `*.tbl` files using the `dbgen` tool. Assuming we have generated a database for the star schema benchmark of scale factor one and stored the resulting `*.tbl` files in `/home/DATA/benchmarks/star_schema_benchmark/SF1/`, we can import the data with the following command:

```
create_ssb_database /home/DATA/benchmarks/star_schema_benchmark/SF1/
```

For the TPC-H benchmark, the command is `create_tpch_database`.

Now CoGaDB imports the data and stores them in the database. Depending on the scale factor, this can take a while. After the import finishes, we can start working with the database. Since CoGaDB is an in-memory database, we first have to load the database in the main memory:

```
loaddatabase
```

Then, we can start issuing queries. We can either use SQL or build in aliases for stored queries. We provide stored queries for all queries of the star schema benchmark. The template command is `ssbXY`, which executes SSB-Query `X.Y` (`X` has to be a number between 1 and 4; `Y` has to be a

number between 1 and 3 except when X is 3, in this case 4 is valid for Y as well).

Now, we can launch queries:

```
CoGaDB>select sum(lo_extendedprice*lo_discount) as revenue from lineorder, dates where  
lo_orderdate = d_datekey and d_weeknuminyear = 6 and d_year = 1994 and lo_discount between 5  
and 7 and lo_quantity between 26 and 35;
```

```
+-----+  
| REVENUE      |  
+=====+  
| 2.49945e+10  |  
+-----+
```

1 rows

Execution Time: 155.28039 ms

You can get a complete list of supported script commands by typing *help*. You can issue any number of commands in the startup script *startup.coga*, to automatically load a database and set certain parameters.

## Support

In case you have any questions or suggestions, please do not hesitate to contact the development team via Sebastian Breß (sebastian.bress@tu-dortmund.de).