qikkDB docs

Introduction

What is qikkDB?

QikkDB is an ultra fast database that uses graphic cards (GPUs) to accelerate computations over big data. In Instarea we work with big telco data what makes us challenged everyday with optimal approaches to manipulation with massive datasets. This project started as a reaction to our needs for fast querying over long tables filled with event records.

The main focus of qikkDB is to **query a single flat and huge table with realtime response**. This approach will find its place in use-cases in which speed is crucial e.g. fraud detection in finance data, web logs analysis and realtime decisions, realtime population analytics on telco data, etc.

Already suitable for

- Filtering and aggregations over single flat huge table
- Complex polygon operations (contains, intersect, union)
- Numeric and datetime data
- Incremental data which are growing over time

Not yet suitable for

- Joining normalized data in multiple tables
- Complex string manipulation
- Modification of historical data
- Advanced scripting (procedures)

Engine for Advanced Visualization

QikkDB is an essential part of a visualization tool TellStory. This tool enables to create a visual stories over big data with realtime responses. To learn more about TellStory see visit this page.

Basic Concepts of Superb Performance

Along with enormous increase in data volume, IT companies seek new possibilities of effective data processing. Especially, queries with real-time or near real-time responses in context of big data are super expensive or hardly possible to achieve with standard database systems and sometimes even various specialized noSql solutions. qikkDB has utilized new hardware opportunities and it has integrated modern graphic cards that enable fast and efficient processing of vast amount of data. And therefore, even billions of records could be queried in milliseconds.

Utilization of Modern Hardware

Since 2007, when NVIDIA launched the CUDA Toolkit, using graphic processing units (GPUs) for high performance and scientific computing has become increasingly popular. This moved focus of graphics cards vendors from visualization towards more general computing. General processing graphics processing units (GPGPUs) enable highly parallelized processing of data thanks to hundreds and even thousands of cores doing work in single GPU. Although these cores can handle simpler logic than those of CPUs, their high number in single unit creates a powerful army. And in result it can process large data vectors much faster than any CPU.

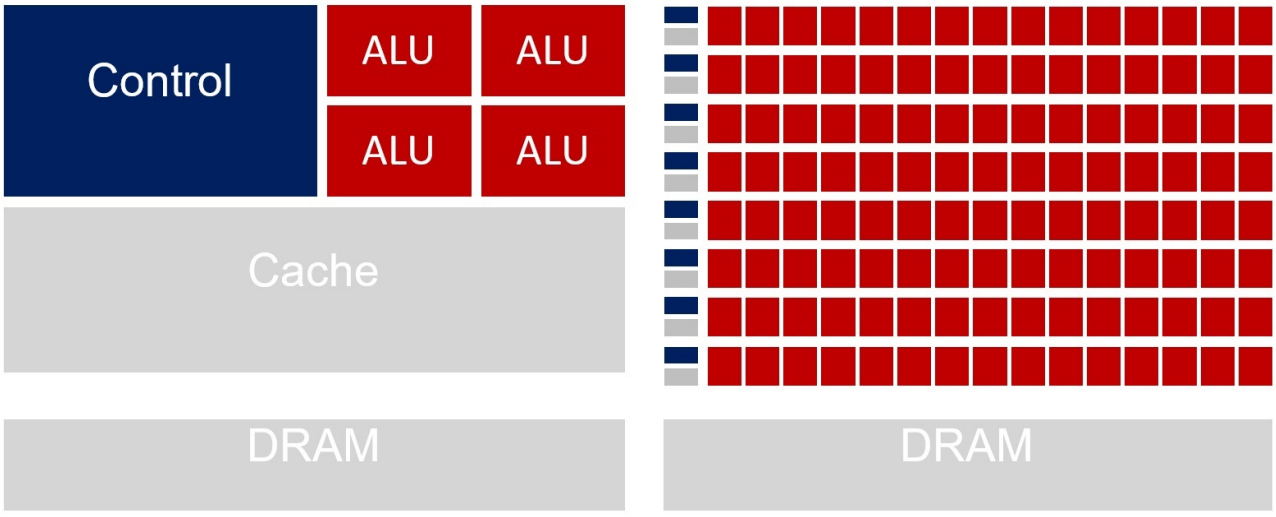


Figure 1. GPUs have thousands of arithmetic logic units (ALUs) in one piece of hardware.

Rebirth of Columnar Data Storage

Utilizing GPUs computation power requires different approach to storing data. The most suitable database architecture that works well with parallel processing is columnar storage. In contrast to conventional relational databases which store data in row-based format, columnar databases store data in separate columns. This concept is decades of years old, but it did not find its place in relational databases that were heavily used until era of big data started. Columnar databases have become popular mostly for analytical workloads on big datasets.

In context of parallel processing, GPUs love long vectors of the same data type as all values are next to each other. In addition to this, data transfers are significantly reduced as only relevant columns are loaded and used for processing. Also, column-oriented databases are more suitable for horizontal scaling, enabling use of low-cost hardware to work in terabytes of data.

Architectural Insight

Query is processed by Parser and then Listener generates individual processing instructions. Disparcher executes these instructions. Dispatcher is actually the control unit of the entire database engine. It remembers execution states (e.g. whether a filter mask is used) and incrementally processes the entire query on one block of data. The blocks are processed in parallel so that each graphic card works with just one block of data at a time. One dispatcher is assigned to each graphics card. After all blocks are processed, the results are combined and returned to the client.

How We Gained Our Speed

Our main advantage is that we have used the latest technology currently available. We use the latest features of the C ++ 17 and NVIDIA CUDA 10 Toolkit. Thanks to the latest technologies, some mathematical operations are optimized and the resulting query time is shorter. In addition, we have saved some time by using the C++ 17’s static if statements together with function templates - some conditions are evaluated during the program compilation, which results in even shorter query execution time. This kind of programming is difficult, but the result is truly rewarding.

In order to achieve the highest processing speed, we have implemented the latest algorithms described in scientific papers in recent years, the oldest ones from 2015 (mostly used for GROUP BY and JOIN clauses). Some algorithms (such as the key aggregation within GROUP BY) have been extended beyond accelerating data structures that bypass the deceleration caused by atomic instructions.

All algorithms implemented in our database engine must take into account parallelism and therefore data that are processed in parallel must not depend on each other. However, there are many that depend on each other and therefore must be pre-prepared or divided into independent parts and re-assembled after processing. Fortunately, this overhead is not so huge and consequently parallel processing is faster than sequential processing on the CPU. This overhead is mainly due to reconstructing the query result (compression of data after filtering with a WHERE condition - we compute a prefix sum of the filter mask and then select the rows that match the condition in parallel.

We try to minimize copying data from RAM to GPU (VRAM). The intermediate results remain in the GPU VRAM and we copy the final output back (which is usually very small thanks to GROUP BY or LIMIT clauses). We also use so-called memory pinning, which ensures that the data stored in RAM is not pageable. As a result, the subsequent copying of data to the GPU will take approximately half the time (depending on the graphics card, the differences between the graphics cards are huge). We have also implemented our own graphics memory allocator, which is approximately 3x faster than the CUDA allocator, and our own cache memory, which makes it no longer necessary to copy the input data when you run a similar query.

Data Flow Within System

Datasets are compressed and persisted on disc storage, but also cached in both CPU memory and GPU memory to minimize latency. GPU processing needs data to be present in its local memory (GPU VRAM) and this results in main limitation of GPU computing – moving data between CPU and GPU memory.

QikkDB reduces this transfer time using three strategies. Firstly, when a query is hit, qikkDB analyzes which columns are relevant and only those are copied to GPU VRAM. Secondly, thanks to compression of data, much lower volume needs to be transferred. Finally, data are kept in GPU VRAM as late as possible to avoid duplicate transfers.

When data are transferred, GPU decompresses it and run functions on top of the data including various filters, aggregations, etc. At this point GPU can finally show how big beast it is □.

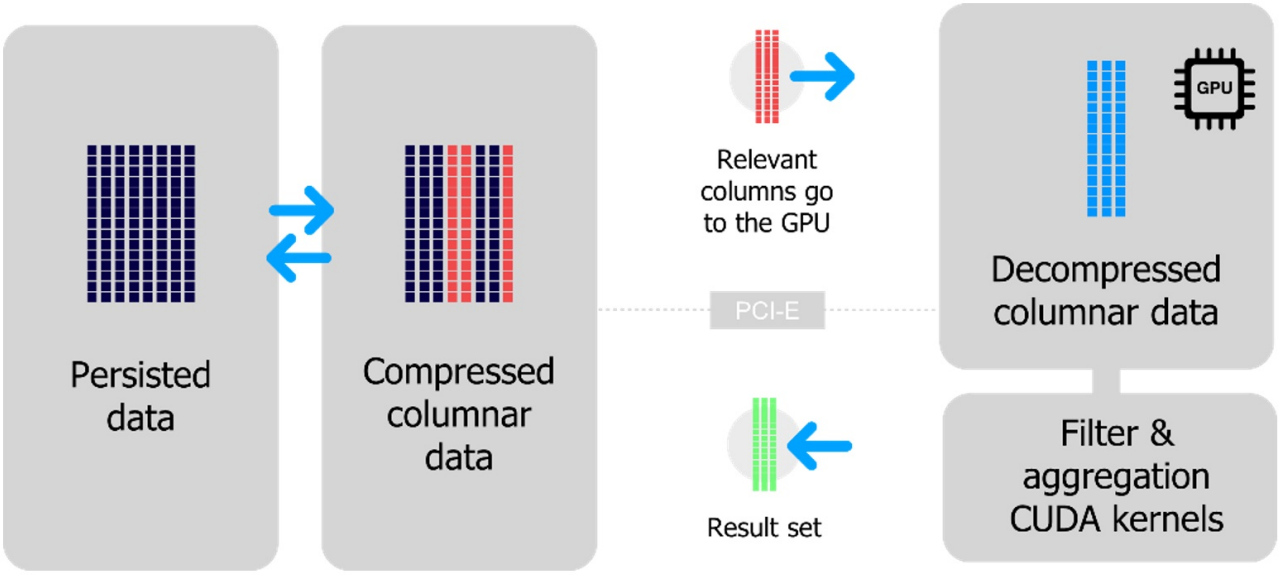


Figure 2. Data flow from disc storage, through main memory RAM to GPU VRAM.

GPU Caching

Especially, in analytics workloads it is very common to use one dataset more than once in a short period of time. Once data are transferred to GPU VRAM it should remain there for repeated use. This is caching at GPU. Of course, GPU VRAM has not unlimited capacity (usually ~16 GB) and we need to wisely manage memory when using various datasets. Thanks to caching, results could be expected in 10x shorter time when queried the same dataset.

Data Compression [Under Development]

As mentioned above, data are internally organized in columnar-wise format. One great advantage of columnar storage is that columns could be very efficiently compressed. QikkDB uses compression to reduce data volume on disk, but also across the whole hierarchy of memory. A compression scheme that is used is rather lightweight in order to provide maximum decompression speed. Therefore, data could be kept compressed even in GPU memory sparing space in GPU RAM that is naturally much smaller. And thus columns are decompressed only when necessary.

Software Toolkit Used

The database layer was implemented in C++ 17 and extension to this are CUDA kernels for computation at NVIDIA graphics cards. The Console is written in C# and therefor it requires .NET runtime environment. We use CMAKE for building and testing the code base. For parsing queries, we use ANTLR v4 library. We also use Google Protocol Buffers in networking. Libraries YAML and Boost are also used. Unit and integration tests are written using GTest library.

Key Capabilities

SQL Language

In the world of databases, SQL is a common interface that almost every data developer or analyst knows and also many tools use it to communicate with persistence layer. QikkDB provides basic SQL statements to select and manipulate data.

Geospatial Functions

Based on our experience with telco data we added geospatial functionality to work with points and polygons. Currently, qikkDB supports inputs in WKT format and provides three main functions – CONTAINS (check whether point is inside polygon), INTERSECTION (new polygon as intersection of two polygons), UNION (new polygon as union of two polygons).

In-Memory

When speed is the most important thing, the whole dataset could be preloaded into RAM. This way is latency decreased to minimum.

Indexing

QikkDB currently provides single clustered index. Similarly, to traditional databases, data are sorted according to chosen columns. Slower insert in turn results in magnitude of order

faster querying.

Multiplatform

Big data world happens on Unix systems. We know that, but we also do not want to forget about companies that run Windows.

Connectors to Other Tools

In order to integrate the database into the processing pipeline we provide currently C# and C++ connectors and other connectors (including ODBC) are planned in the future.

Scalability

Multiple GPUs within a single server node are detected automatically and the load can be balanced. Adding new nodes is a must when working with terabytes and petabytes of data what could be easily configured. It is possible to use up to 8 Tesla GPUs in a single machine. Such machine is comparable to cluster of 20 high-end CPU nodes.

Benchmark & Results

Test Queries

This benchmark was inspired by Mark Litwintschik's benchmark in which four SQL queries of different complexity were used for testing. The queries were defined as follows:

```
--Query #1:
SELECT cab_type, COUNT(*) FROM trips GROUP BY cab_type;
```

```
--Query #2:
SELECT passenger_count, AVG(total_amount) FROM trips GROUP BY passenger_co
```

```
--Query #3:
SELECT passenger_count, YEAR(pickup_datetime) AS pickup_year,
    COUNT(*) FROM trips GROUP BY passenger_count, pickup_year;
```

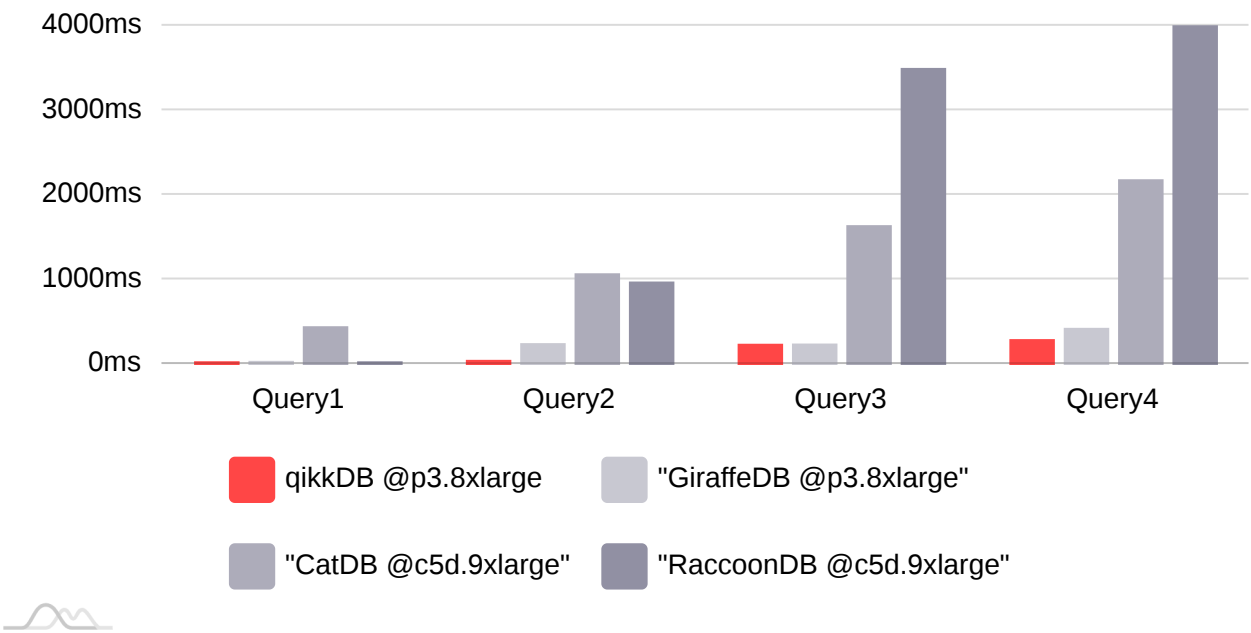
```
--Query #4:
SELECT passenger_count, YEAR(pickup_datetime) AS pickup_year,
    CAST(trip_distance AS INT) AS distance,
    COUNT(*) AS the_count
    FROM trips GROUP BY passenger_count, pickup_year, distance
    ORDER BY pickup_year, the_count DESC;
```

Execution Times Results (Single Server Instance)

We compared our database qikkDB with three leading databases aimed at analytics workloads of different types:

- **GPU database** (referred as "GiraffeDB"),
- **columnar database** (referred as "CatDB"),
- **relational database** (referred as "RaccoonDB").

The results in the table show execution times in **milliseconds**. For each of the 4 queries **average execution time** was obtained based on **200 query runs**, while the **first runs were not counted** towards the average. The results were obtained using the **latest versions of the databases** which were available on the **2th of October 2019**. The benchmarking dataset contained **1.2B data rows in 7 columns** with various data types in a single table. In our database we did not use indexing. The results have been tested on qikkDB **version 1.4.1**.



Speed comparison of qikkDB against competition based on 4 SQL queries.

Query	qikkDB@ p3.8xlarge	qikkDB@ g4dn.12xlarge	GiraffeDB@ p3.8xlarge	CatDB@ c5d.9xlarge	RaccoonDB@ c5d.9xlarge
#1	22	37	25	435	22
#2	37	82	235	1061	964
#3	228	925	231	1630	3491
#4	283	1105	417	2174	3996

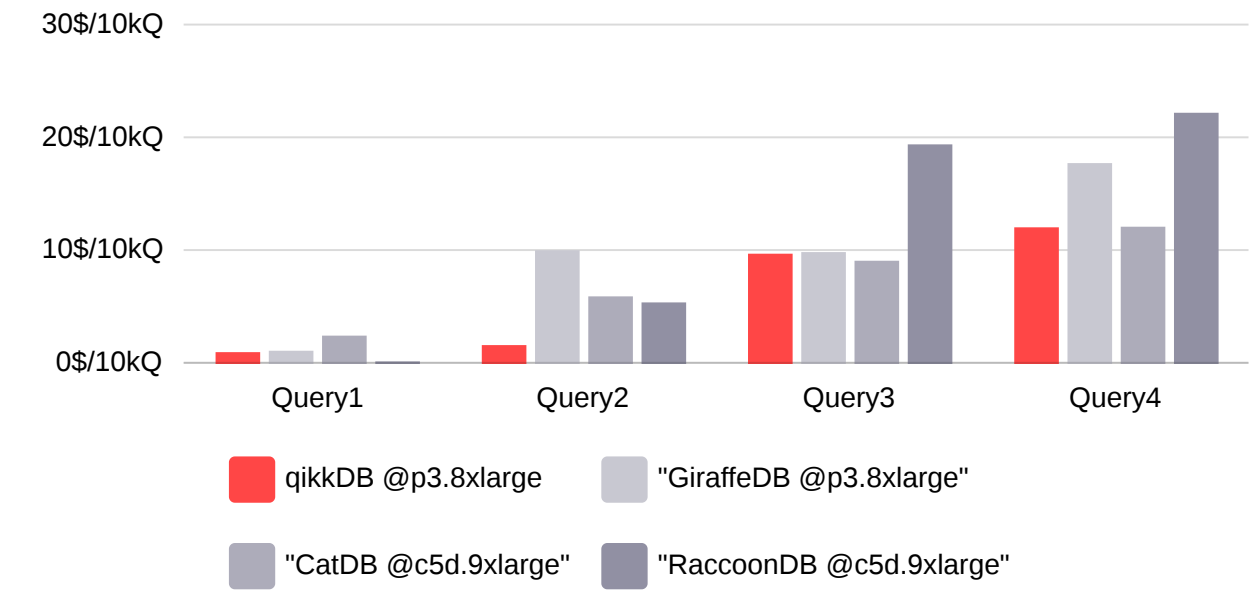
Average	**143**	**537**	**227**	**1325**	**2118**

Price per Query

As the databases require different hardware we also provide a **comparison from effectiveness perspective** what brings normalized view on performance.

We have estimated price per 10 000 query runs ($/10kQ) based on Amazon's on-demand EC2 pricelist (region: **EU Frankfurt**) for GPU optimized instances and compute optimized (CPU) instances. These prices were calculated as follows: we cumulated ten thousand query execution times and calculated the price for the total time according to Amazon's on-demand EC2 pricelist. The prices include hardware running costs only excluding any licence costs.

Query	qikkDB@ p3.8xlarge	qikkDB@ g4dn.12xlarge	GiraffeDB@ p3.8xlarge	CatDB@ c5d.9xlarge	RaccoonDB@ c5d.9xlarge
#1	0.935 $/10kQ	0.503 $/10kQ	1.062 $/10kQ	2.414 $/10kQ	0.122 $/10kQ
#2	1.572 $/10kQ	1.114 $/10kQ	9.982 $/10kQ	5.889 $/10kQ	5.350 $/10kQ
#3	9.685 $/10kQ	12.565 $/10kQ	9.812 $/10kQ	9.047 $/10kQ	19.375 $/10kQ
#4	12.021 $/10kQ	15.010 $/10kQ	17.713 $/10kQ	12.066 $/10kQ	22.178 $/10kQ
Average	**6.053 $/10kQ**	**7.298 $/10kQ**	**9.642 $/10kQ**	**7.354 $/10kQ**	**11.756 $/10kQ**



Effectiveness comparison of qikkDB against competition based on 4 SQL queries.

Hardware & Software Used for Testing

We used Amazon Web Services for testing our database qikkDB as well as databases of our competitors. We tested the same queries on the same dataset. The GPU based databases were tested on the same GPU optimized instance. The CPU based databases were tested on the same CPU optimized instance. We used **single instance** for testing.

GPU Optimized Instance	p3.8xlarge	g4dn.12xlarge
vCPU	32x Intel® Xeon® Scalable (Skylake)	48x Intel® Xeon® Scalable (Cascade Lake)
GPU	4x NVIDIA Tesla V100 16GB	4x NVIDIA Tesla T4 16GB
Memory (RAM)	244 GB	192 GB
OS	Ubuntu 18.04 LTS	Ubuntu 18.04 LTS
NVIDIA CUDA Toolkit	10.1.168	10.1.243

NVIDIA Driver	430.26	418.87

CPU Optimized Instance	c5d.9xlarge
vCPU	36x 3.0 GHz Intel® Xeon® Platinum 8000 (3.5 GHz Intel Turbo Boost)
Memory (RAM)	72 GB
OS	Ubuntu 18.04 LTS

Similar Hardware Prices

We were not able to find out the exact prices for above mentioned GPU and CPU optimized instaces, but prices for servers according to Dell's PowerEdge R840 Rack Server configurator would be as follows (updated on 8th of September 2019):

Similar GPU Optimized Instance:
GPUs: 4x NVIDIA Tesla™ V100 16G Passive GPU
CPU: 2x Intel® Xeon® Platinum 8253 2.2G, 16C/32T, 10.4GT/s, 22M Cache, Turbo, HT (125W) DDR4-2933
Memory: 30x 8GB (total: 240GB) RDIMM, 2666MT/s, Single Rank
Total cost: 62,850.16 $

Similar CPU Optimized Instance:
CPU: 2x Intel® Xeon® Gold 6254 3.1G, 18C/36T, 10.4GT/s, 24.75M Cache, Turbo, HT (200W) DDR4-2933
Memory: 9x 8GB (total: 72GB) RDIMM, 2666MT/s, Single Rank
Total cost: $14,445.47 $

Script for Reproducing the Dataset

We used subset of the TLC Trip Record Data which contained 1.2B rows of data. You can follow these steps to download and preprocess data by yourself or you can **download preprocessed dataset** here:

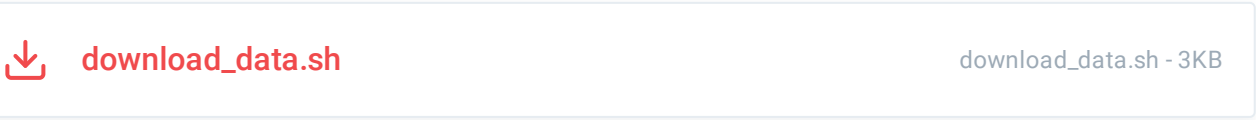
1. Create or download the file named '**urls_data.txt**' which will contain the following URLs:

urls_data.txt — urls_data.txt - 4KB

```
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2010-01.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2010-02.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2010-03.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2010-04.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2010-05.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2010-06.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2010-07.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2010-08.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2010-09.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2010-10.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2010-11.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2010-12.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2011-01.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2011-02.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2011-03.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2011-04.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2011-05.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2011-06.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2011-07.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2011-08.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2011-09.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2011-10.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2011-11.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2011-12.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2012-01.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2012-02.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2012-03.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2012-04.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2012-05.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2012-06.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2012-07.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2012-08.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2012-09.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2012-10.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2012-11.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2012-12.csv
```

```
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2013-01.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2013-02.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2013-03.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2013-04.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2013-05.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2013-06.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2013-07.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2013-08.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2013-09.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2013-10.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2013-11.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2013-12.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2014-01.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2014-02.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2014-03.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2014-04.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2014-05.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2014-06.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2014-07.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2014-08.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2014-09.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2014-10.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2014-11.csv
https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2014-12.csv
```

2. Create or download the bash script named '**download_data.sh**' which will contain the following commands:




```
#!/bin/bash

readonly YELLOW_RESULT_PATH=aggregated_yellow_tripdata.csv
readonly GREEN_RESULT_PATH=aggregated_green_tripdata.csv
readonly PART_ONE=trips-part1.csv
readonly PART_TWO=trips-part2.csv
readonly PART_ONE_RESULT=preprocessed-trips-part1.csv
readonly PART_TWO_RESULT=preprocessed-trips-part2.csv

echo "The process has started."
echo "Downloading data. Please wait. Step [1/4]"
wget -i urls_data.txt
rm urls_data.txt
```

```
14 echo "Downloading data has finished. Removing bad rows. Step [2/4]"
15
16 # concat yellow taxi files into a single csv file
17 head -1 yellow_tripdata_2010-01.csv | cut -d, -f1,2,3,4,5,18,19 >> $YELLOW
18 head -1 yellow_tripdata_2010-01.csv | cut -d, -f1,2,3,4,5,18,19 >> $GREEN_
19
20 for f in yellow_tripdata_2010*.csv
21 do
22   cut -d, -f1,2,3,4,5,18,19 $f | tail -n +2 -q >> $YELLOW_RESULT_PATH
23   rm $f
24   echo "File $f has been concated and removed."
25 done
26
27 # remove the second line, because it is broken
28 sed -i '2d' $YELLOW_RESULT_PATH
29
30 for f in yellow_tripdata_2011*.csv
31 do
32   cut -d, -f1,2,3,4,5,18,19 $f | tail -n +2 -q >> $YELLOW_RESULT_PATH
33   rm $f
34   echo "File $f has been concated and removed."
35 done
36
37 for f in yellow_tripdata_2012*.csv
38 do
39   cut -d, -f1,2,3,4,5,18,19 $f | tail -n +2 -q >> $YELLOW_RESULT_PATH
40   rm $f
41   echo "File $f has been concated and removed."
42 done
43
44 for f in yellow_tripdata_2013*.csv
45 do
46   cut -d, -f1,2,3,4,5,18,19 $f | tail -n +2 -q >> $YELLOW_RESULT_PATH
47   cut -d, -f1,2,3,4,5,18,19 $f | tail -n +2 -q >> $GREEN_RESULT_PATH
48   echo "File $f has been concated."
49 done
50
51 # remove the second line, because it is broken
52 sed -i '2d' $GREEN_RESULT_PATH
53
54 for f in yellow_tripdata_2014*.csv
55 do
56   cut -d, -f1,2,3,4,5,18,19 $f | tail -n +2 -q >> $YELLOW_RESULT_PATH
57   cut -d, -f1,2,3,4,5,18,19 $f | tail -n +2 -q >> $GREEN_RESULT_PATH
58   echo "File $f has been concated."
59 done
60
61 echo "Concating has finished. Adding column 'cab_type' with value '1' to d
62
63 # remove Windows end lines
64 tr -d '\r' < $YELLOW_RESULT_PATH >> $PART_ONE
```

```
65
66  # add new column with data '1'
67  gawk -i inplace '{ printf("%s,1\n", $0); }' $PART_ONE
68
69  # rename column name in header from '1' to 'cab_type'
70  sed -i '1s/1/cab_type/' $PART_ONE
71
72  echo "Concating has finished. Adding column 'cab_type' with value '2' to d
73
74  # remove Windows end lines
75  tr -d '\r' < $GREEN_RESULT_PATH >> $PART_TWO
76
77  # add new column with data '2'
78  gawk -i inplace '{ printf("%s,2\n", $0); }' $PART_TWO
79
80  # rename column name in header from '2' to 'cab_type'
81  sed -i '1s/2/cab_type/' $PART_TWO
82
83  rm yellow_tripdata_20*
84  rm $YELLOW_RESULT_PATH
85  rm $GREEN_RESULT_PATH
86
87  # remove bad rows
88  awk -F',' 'NF==7' $PART_ONE >> $PART_ONE_RESULT
89  awk -F',' 'NF==7' $PART_TWO >> $PART_TWO_RESULT
90
91  echo "Finished. Result is saved in 'preprocessed-trips-part1.csv' and 'pre
92  rm $PART_ONE
93  rm $PART_TWO
94  rm $0
```

3. Run the script. We recommend to run the script in background, because it will take a lot of time to finnish.

List of Competitors

We consider our greatest competitors to be:

- **Blazing**
  Distributed and ACID-compliant GPU-accelerated SQL engine with data lake integration.

- **Brylyt**
  Scalable GPU-accelerated RDBMS for fast analytic and streaming workloads, leveraging PostgreSQL.

- **ClickHouse**
  Column-oriented RDBMS powering Yandex.

- **OmniSci**
  A high performance, in-memory, column-oriented RDBMS, designed to run on GPUs.

- **Kinetica**
  GPU-accelerated database for real-time analysis of large and streaming datasets.

- **Microsoft SQL Server**
  Well known RDBMS developed by Microsoft.

- **SQream**
  Scalable GPU-accelerated RDBMS for fast analytic and streaming workloads, leveraging PostgreSQL.

- **Vertica**
  Columnar RDBMS designed to handle modern analytic workloads, enabling fast query performance.

We encourage you to test our database and our competitors products with your own queries simulating your use cases.

Installation & Getting Started

Hardware Limitations

You will need an NVIDIA graphic card with **Compute Capability 6.0** or higher. You can find a list of NVIDIA graphic cards and their compute capabilities here.

Software Dependencies

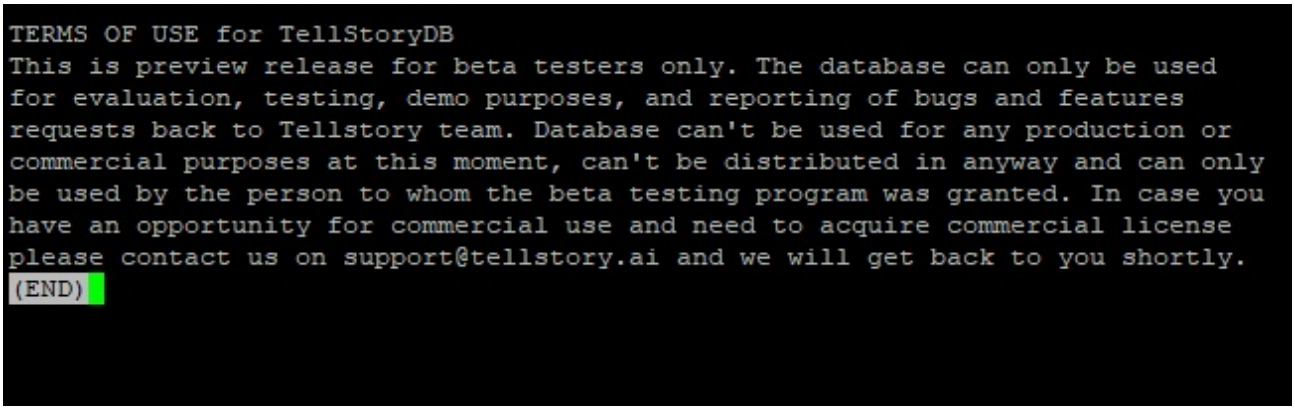
In both Linux and Windows enviroments, you need to have GPU driver installed. We recommend to download the newest NVIDIA GPU driver. If you install whole NVIDIA CUDA package, this will install also suitable drivers.

Windows Only

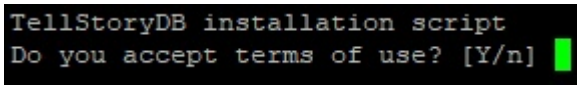
You need to have installed **.NET Core Runtime 2.2** before running the installation wizard. You can download it from here. It is needed for Console Client, which is included.

Linux: Installation Script

Make sure you can execute the installation script (if not, use command `chmod +x qikkDB_Linux_Installer.sh`) and that you are running the script as root.

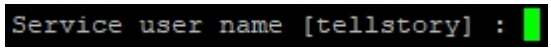


After reading Terms of Use, write `:q` to continue.



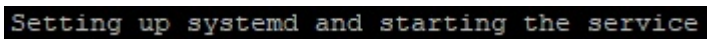
Hit ENTER on keyboard if you accept Term of Use.

Choose destination folder and hit enter on keyboard to continue. It cannot be the current folder, you have to create a new folder. A relative path is supported.



Choose a user name (user will own the installed files and the service will be run under this user) and hit enter on keyboard to continue.

Chose a user group (user will own the installed files and the service will be run under this user) and hit enter on keyboard to continue.



Now the service is running and **if the database crashes, it will be restarted automatically**.

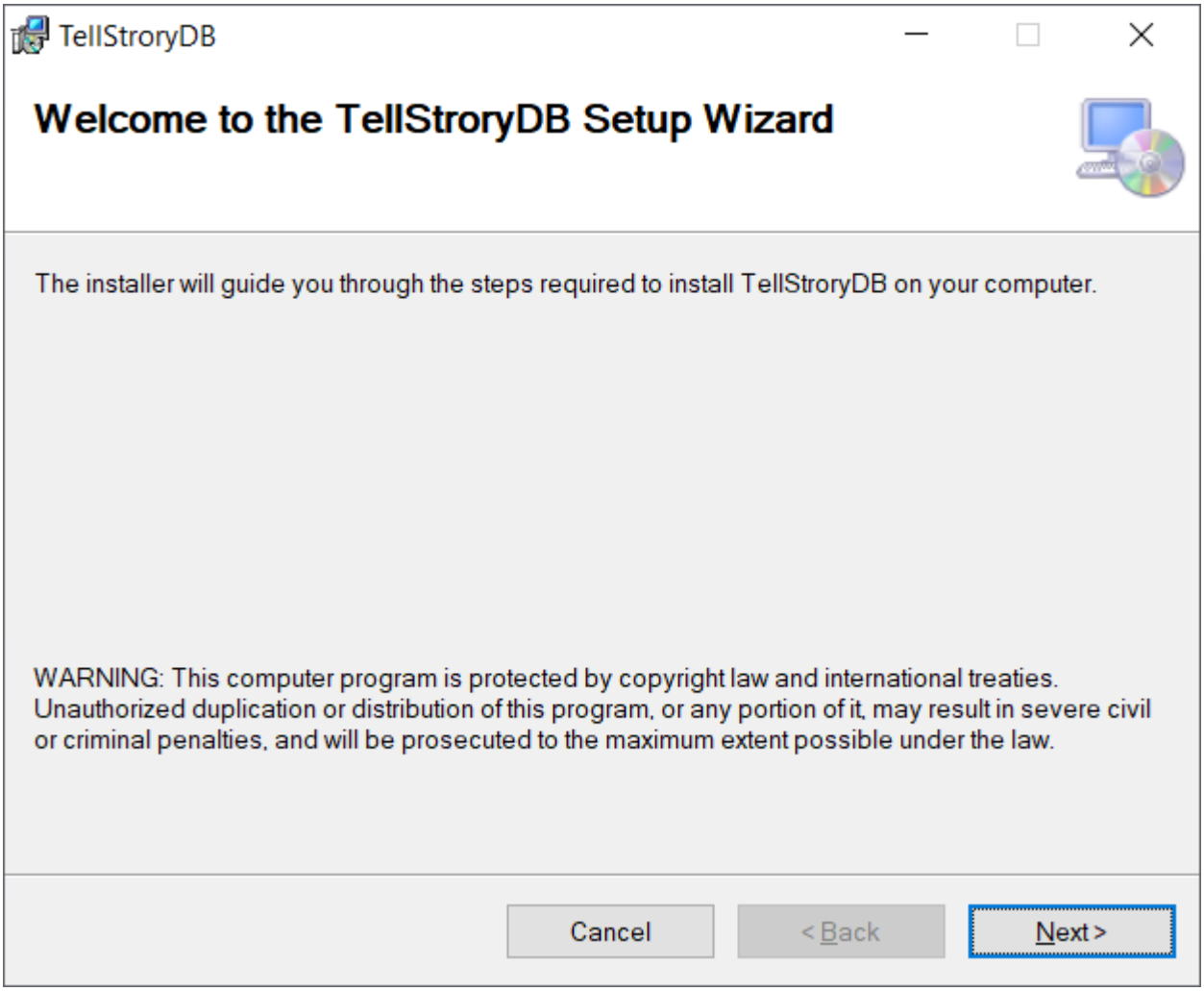
Linux: Deployment via Docker

The docker image is available at Docker Hub as qikkdb/qikkdb-community:latest. Requirements for using this image are drivers at your Linux host machine as well as Docker.io and NVIDIA Docker tools.

Detailed reference for running qikkDB in Docker is at our Docker hub repository.

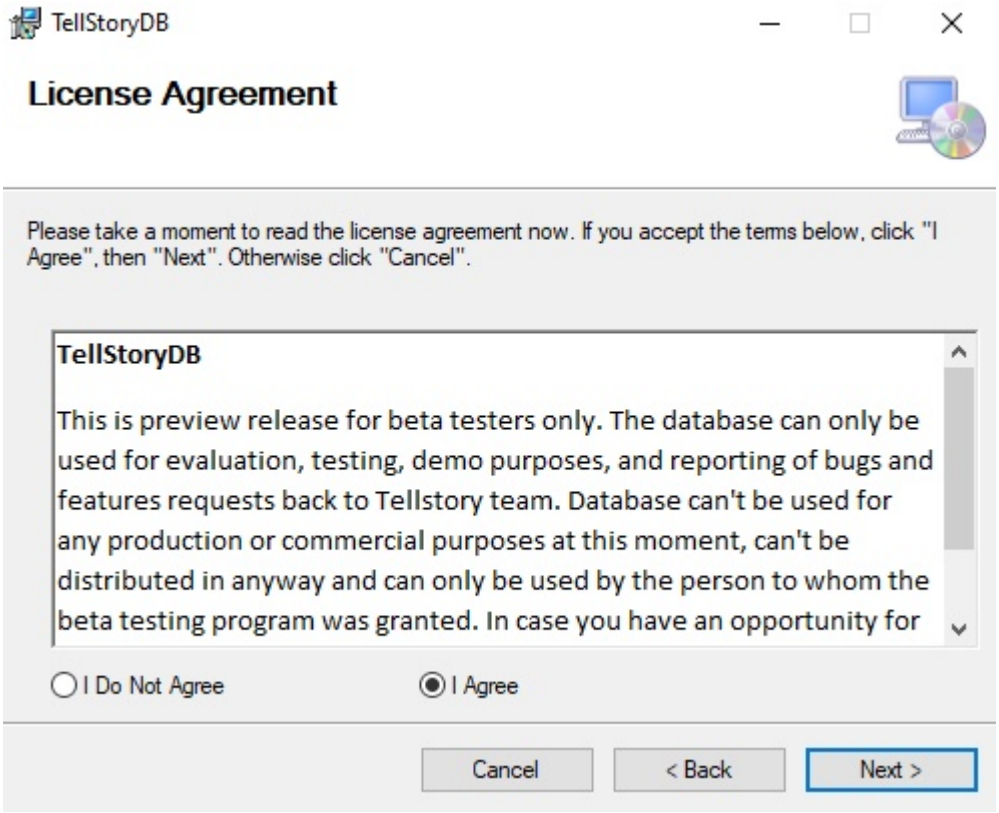
Windows: Installation Wizard

First step is to read general information and warning about copyright and penalties resulting from its violation. For the next step press “Next” button.

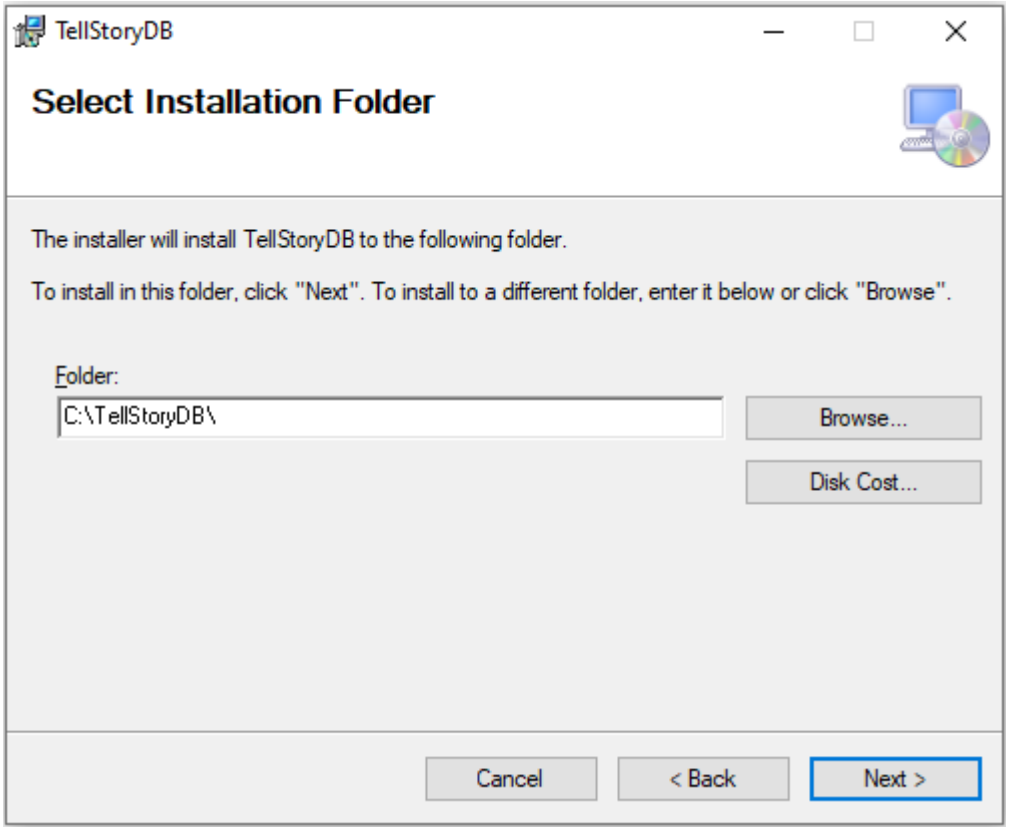


Now you have to read and accept the Terms of Use to continue the installation process.

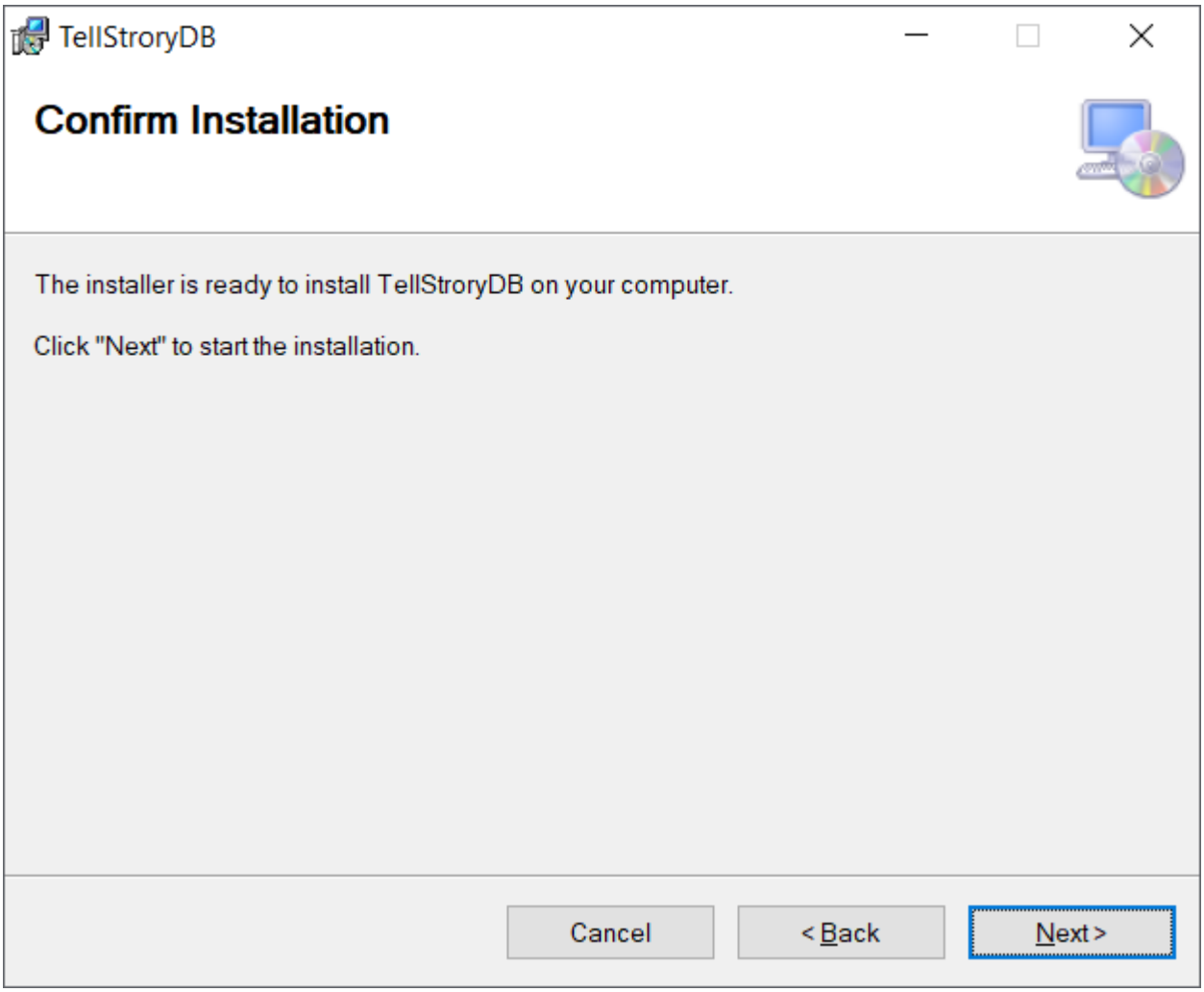
Console client is located in .\console\QikkDB.ConsoleClient.exe. Just double click on it or run it using powershell or cmd (`.\QikkDB.ConsoleClient.exe`). Console have to be started **after** the successful start of database core. We recommend you to run command 'help' to see all the options that console supports. You can see the list of supported console commands here as well.



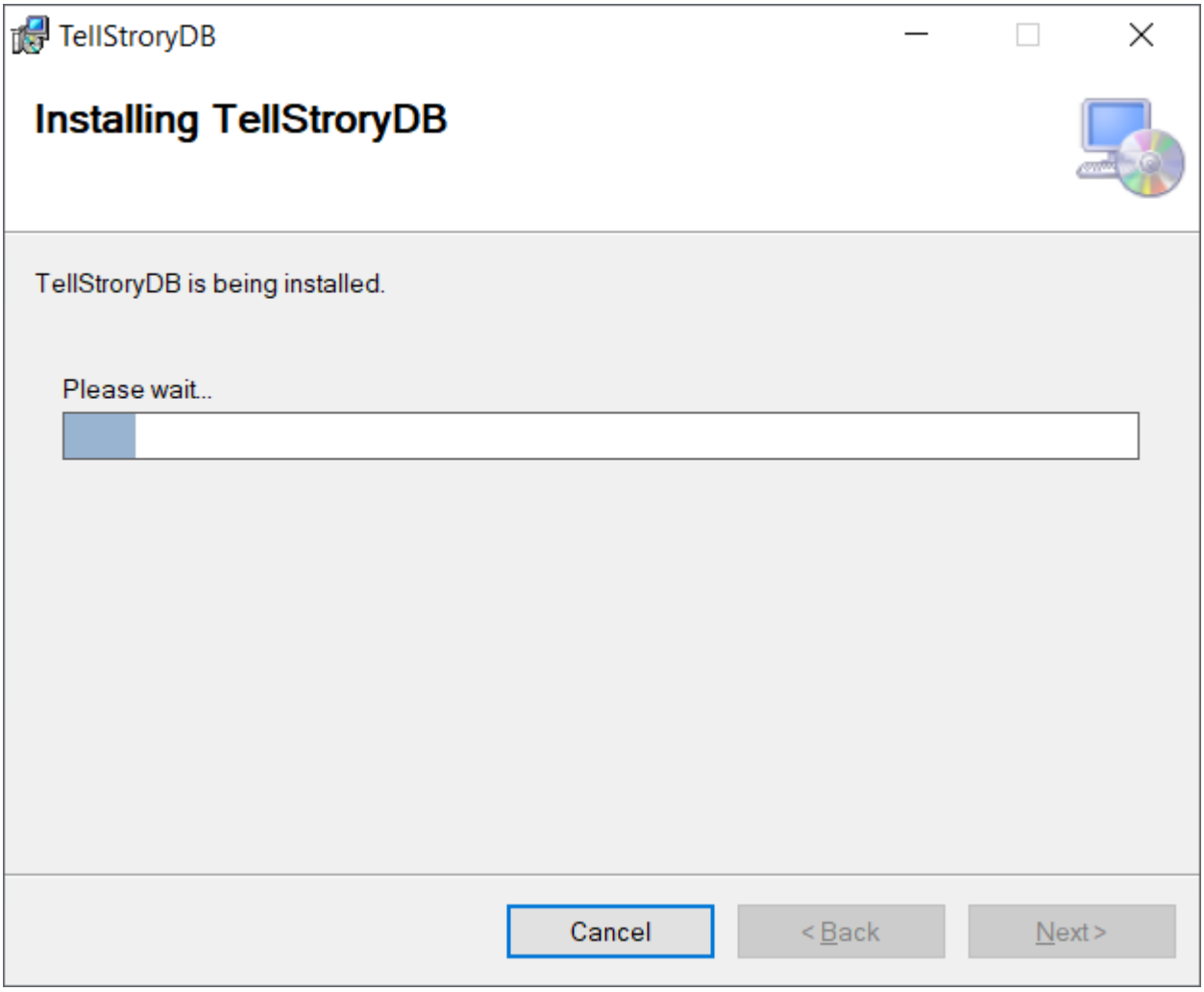
The next step is to choose the folder where the installer will install qikkDB. Bear in mind, that **you need to have read & write access to the selected folder**. For the next step press “Next” button.



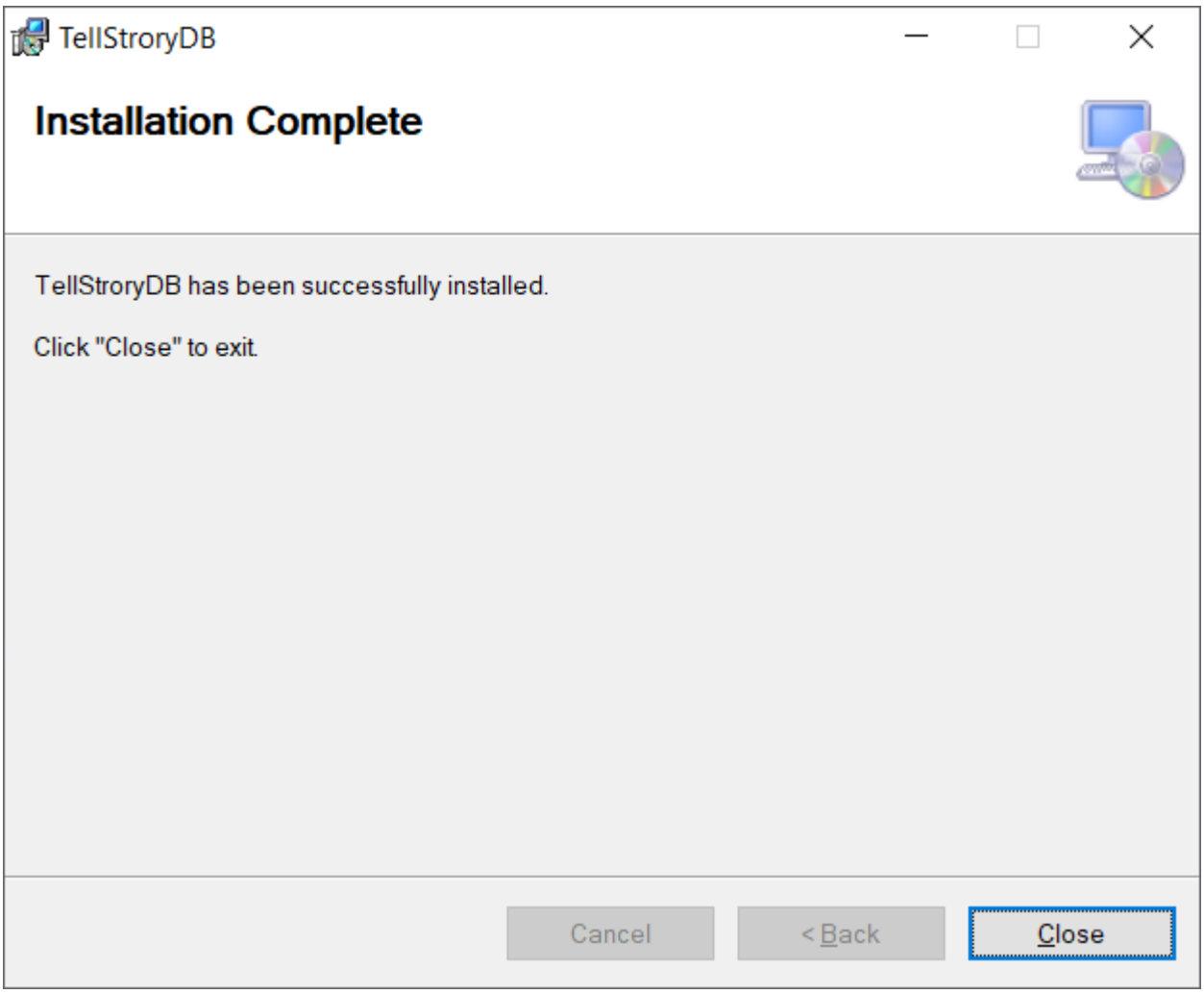
In this step you need to confirm the installation. For the next step press “Next” button.



After that, you may need to allow changes in your computer and after this confirmation a loading bar will inform you about the state of installation process.



Last screen is just to inform you about the success of the installation. After pressing “Close” button, qikkDB is ready to use.



Starting Database

Go to the directory where you have installed the program.

Configuration

The configuration files are in directory 'configuration'. There is a configuration file for logging ('log_config.default') and main configuration file for starting database core ('main_config.default'). If you want to change default settings, you **have to create a copy from configuration file and remove '.default' extension.** Now you can edit this new file and the changes will be reflected after starting / restarting the database core.

Databases

Databases will be saved (if not changed in main_config file) in directory 'databases'. After starting / restarting database, all databases which have extension '.db' will be loaded. If there is a database which you do not want to load, just change the extension '.db' to something else, e.g. '.dbx'.

Logs

Logs will be saved in directory 'logs' if not changed in configuration file ('log_config').

Linux

If not deployed via Docker, you can start database core as follows:

After the sucessful installation, database server is running as service. If the database server crashes when running as service, it will be automatically restarted. If you want to run the database server using executable file, it can be found in directory ./bin/qikkdb, just run it using command `./qikkdb` (preferably run it as background process - e.g. using `tmux`). If you want to import a single csv file locally, run it with arguments (more information here).

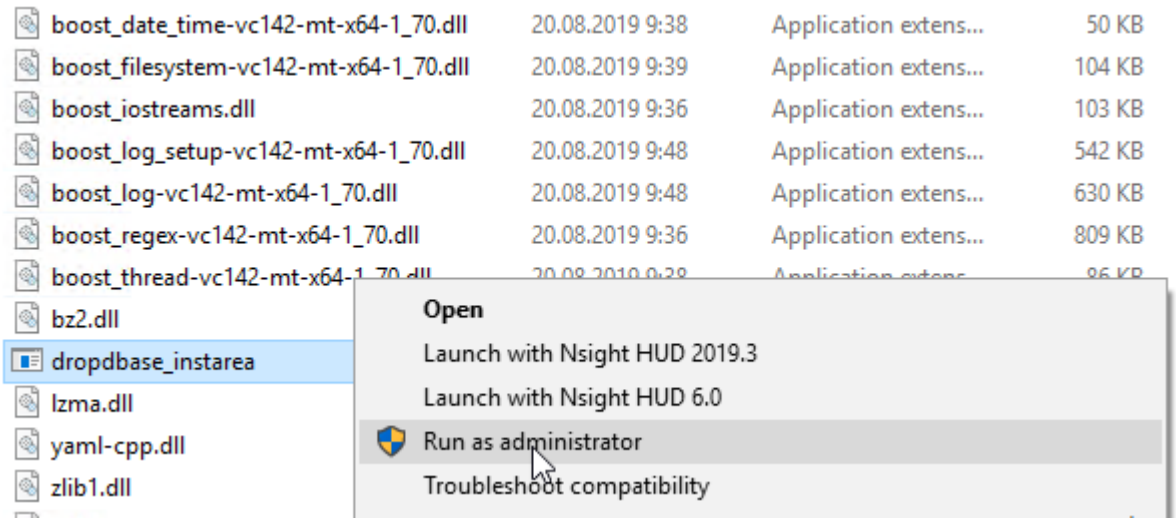
The docker image is available at Docker Hub as qikkdb/qikkdb-community:latest. Requirements for using this image are drivers at your Linux host machine as well as Docker.io and NVIDIA Docker tools.

Requirements for using this image are drivers at your Linux host machine as well as Docker.io and NVIDIA Docker tools. Detailed reference for running qikkDB in Docker is at our Docker hub repository.

Console client is located in ./console/QikkDB.ConsoleClient. Run it using command `./QikkDB.ConsoleClient` if you are located in directory ./console. Console have to be started **after** the successful start of database server. We recommend you to run command 'help' to see all the options that console supports. You can see the list of supported console commands here as well.

Windows

First step is to start database server. After the database server (database core) has successfully started, you can connect console or web client to this running database server. After successful installation, the executable file is in .\bin\dropdbase_instarea.exe. To execute properly, **run as administrator**. If you want to import a single csv file locally, run it using powershell or cmd (also in administrator mode) and add needed arguments (more information here). Do not start the database server more than once concurrently.



Console client is located in ./console/QikkDB.ConsoleClient. Double click on it or run it using powershell or cmd (`./QikkDB.ConsoleClient`). Console have to be started **after** the successful start of database server. We recommend you to run command 'help' to see all the options that console supports. You can see the list of supported console commands here as well.

First Query

Importing file using console

Let's import some data to start querying. But first, a new database has to be created:

```
CREATE DATABASE first_database
```

Locate a CSV file with your data on harddrive and import using console:

```
IMPORT first_database "C:\temp\sample.csv"
```

The console will be showing progress of importing process and will print message once done. A new table `sample` will be created in the database `first_database`.

Getting number of records in table

Before you run a SQL query, you need to select previously created database:

```
USE first_database
```

And now you can query the table for number of records:

```
SELECT COUNT(*) FROM sample
```

Naming Rules for Database / Table / Column Names

Windows filesystem is case insensitive, so naming your first database as 'MyDatabase' and the second one as 'mydatabase' will result in rewritting one of the database files. Bear this in mind and **do not differ your database names just in lower/upper case**, if you are running database core on Windows.

Linux filesystem is case sensitive, so you can name your database as you want and **if a database core is running on Linux, it will work just fine**.

Names **cannot** contain characters '/' (slash), '\' (backslash) or '@' (at).

What's next

To learn more about:

- creating and altering databases and tables - SQL Data Definition
- querying tables and supported SQL functionality - SQL Data Manipulation
- using console for querying and importing - Console Client
- what are the hardware limitations for the database - Hardware Requirements
- what needs to be improved and we know about that - Known Issues
- how you can improve performance - Tips & Tricks

Community Edition

Limitations of the qikkDB Community Edition

qikkDB in the Community Edition is limited by the following factors. Should you need to use qikkDB beyond these factors get in touch with us at hello@qikk.ly for custom pricing.

Max factors in Community Edition

- 1 GPU card
- 2 databases
- 4 tables in a database
- 8 columns in a table
- 1.000.000.000 rows in a table

AWS AMI image

qikkDB is available as a Community Edition free of license costs in Amazon's AWS. You will find it in the Marketplace here: https://aws.amazon.com/marketplace/pp/B089Q9S55J

Docker image

The docker image is available at Docker Hub as qikkdb/qikkdb-community:latest. Detailed reference for running qikkDB in Docker is at our Docker hub repository.

SQL Data Definition

Syntax is case insensitive following this template:

```
KEYWORDS required_input [optional_input];
```

Database basics

Allows you to create, modify or delete a database. You do not have to use database before writing these commands.

CREATE DATABASE

Since version: 1.4.0
Creates database with specified name and block size. Database name cannot contain characters '**/**' (slash), '****' (backslash) or '@' (at). All tables within the same database has the same block size. For more information about block size see Hardware limitations section. If no block size is specified, the value specified in configuration file will be used.

```
CREATE DATABASE database_name [block_size];
```

ALTER DATABASE

Allows you to modify existing database.

Rename Database

Since version: 1.4.1
Allows you to modify existing database. You can rename database using the following command:

```
ALTER DATABASE database_name RENAME TO new_database_name;
```

Change Block Size

Since version: 1.4.2
Change block size for all tables of a particular database and also change block size of the database. This may take a while. Column data needs to be copied to tables with a different block size. Changing block size of a column that can have NULL values (is nullable) takes about 500x longer than changing the block size of a column that is marked as UNIQUE (and therefore cannot have NULL values).

```
ALTER DATABASE database_name ALTER BLOCK SIZE new_block_size;
```

DROP DATABASE

Since version: 1.4.1
Deletes the chosen database.

```
DROP DATABASE database_name;
```

Table Basics

Allows you to create, modify or delete a table. You have to use database, to which the table belongs to, before you write any of the following commands.

Constraints Types

Each constraint is binded with constraint type, constraint name and one or more column names. So we can have for example a constraint UNIQUE which will be named "uniq_const"

and it will be binded with two columns "colA" and "colB". If we decide to drop the constraint "uniq_const", it will drop this constraint on all columns which it is binded with (so in this case "colA" and "colB").

Adding duplicit constraint of the same type results in error. For example, you cannot create constraint NOT NULL with constraint name "cons_not_null_1", which would be binded with "ColA" and "ColB" and then create another constraint NOT NULL, let's name it "cons_not_null_2", which would be binded just with "ColB" (because the "ColB" already has NOT NULL constraint which is named "cons_not_null_1"). **If you want to drop a constraint, you have to use it's name.**

UNIQUE

Since version: 1.4.2
The UNIQUE constraint ensures that all values in a constrained column are different and in this column cannot be a NULL VALUE, not even a single one. To add UNIQUE constraint on a column (using ALTER TABLE command), this column must have NOT NULL constrained already applied on it, otherwise it results in error. When trying to add UNIQUE constraint on a column which have duplicate values or a NULL value/s, it results in error. Dropping UNIQUE constraints does not drop also NOT NULL constraint, NOT NULL constraint will remain. The suffix in default constraint name is: "_UC" (the default constraint name is made up of the column name and constraint suffix, for more information see create table command). If the default name of the constraint has been used (because the constraint name has not been explicitly defined), this default name must be used when dropping constraint which name was not explicitly defined.

NOT NULL

Since version: 1.4.2
The NOT NULL constraint ensures that the constrained column cannot contains NULL values. When trying to add NOT NULL constraint (using ALTER TABLE command) on a column which has one or more NULL values, it results in error. You cannot drop NOT NULL constraint if a column has UNIQUE constraint - you need to first drop UNIQUE constraint and then you can drop NOT NULL constraint. The suffix in default constraint name is: "_NC" (the default constraint name is made up of the column name and constraint suffix, for more information see create table command). If the default name of the constraint has been

used (because the constraint name has not been explicitly defined), this default name must be used when dropping constraint which name was not explicitly defined.

INDEX

Since version: 1.4.0
The index constraint creates clustered index on the particular column or multiple columns. The suffix in default constraint name is: "_IC" (the default constraint name is made up of the column name and constraint suffix, for more information see create table command). If the default name of the constraint has been used (because the constraint name has not been explicitly defined), this default name must be used when dropping constraint which name was not explicitly defined.

CREATE TABLE

Since version: 1.4.0
Creates a new table along with its columns in the database. Table name cannot contain characters '**/**' (slash), '****' (backslash) or '@' (at). The block size can be specified for particular tables, but if the block size is not eplicitly defined, the value of the database block size will be used also as the table block size for this new table. You can also add constraint on one or more columns. Each constrained has it's name, type and list of columns that it aplies to. Adding constraints is optional and CONSTRAINT_TYPE can be chosen from this list. Optional parameter allows you to make an index. In order to run this command you need to select database first (see Console).

```
CREATE TABLE table_name [block_size]
  (
    column1_name COLUMN_DATA_TYPE
    [, column2_name COLUMN_DATA_TYPE, ...]
    [, CONSTRAINT_TYPE constraint1_name (column1_name [, column2_name, ...
     [, CONSTRAINT_TYPE constraint2_name (column2_name [, column1_name, ..
    ]
  );
```

We also support another syntax to create table with contraints, withou explicitly specifying their names. The default values for the constraint names would be used, which are made up of the column name and suffix according to constraint type (e.g. unique constraint for

column named "colA" would look like this: "colA_UC"). The constraint suffixes are mentioned here under each constraint type. The syntax is following:

```
CREATE TABLE table_name [block_size]
  (column1_name COLUMN_DATA_TYPE CONSTRAINT_TYPE
    [, column2_name COLUMN_DATA_TYPE CONSTRAINT_TYPE, ...]
  );
```

ALTER TABLE

Allows you to modify existing table.

Add Column

Since version: 1.4.0
You can add column with specified datatype. Column name cannot contain characters '**/**' (slash), '****' (backslash) or '@' (at). When adding new column to the table with data already inserted, new column will be fill with NULL data values to equalize the number of data in columns.

```
ALTER TABLE table_name ADD column_name datatype;
```

Change Column Data Type

Since version: 1.4.0
You can also change data type of a specified column in a specified table. When changing to to a data type with a smaller byte length, there is a **possible lost of data**. Also when changing data type is not possible for some elements of a particular column (e.g. cast string 'DHC-05' to integer), the result of changed data type for these elements is **NULL value**. We recommend you to check the values in a column before changing data type so the result is not the column with all elements as a NULL value. To see, what data types we support see Supported Data Types. The possible combinations for changing data types are:

- Bool → Bool, Double, Float, Int, Long, String

- Double → Bool, Double, Float, Int, Long, String
- Float → Bool, Double, Float, Int, Long, String
- Int→ Bool, Double, Float, Int, Long, String
- Long → Bool, Double, Float, Int, Long, String
- String → Bool, Double, Float, Geo_Point, Geo_Polygon, Int, Long, String
- Geo_Point → String
- Geo_Polygon → String

Since version: 1.4.2
When corventing String to Bool, any non zero number (as string value, e.g. "42") will be converted as true value. String value "0" will be converted as false. String value "true" (case insensitive) will be converted as bool value true. String value "false" (case insensitive) will be converted as bool value false. Anything else will be converted as NULL value.

The syntax of a command to change data type is as follows:

```
ALTER TABLE table_name ALTER COLUMN column_name data_type;
```

Rename Table or Column

Since version: 1.4.1
You can rename a table using the following command:

```
ALTER TABLE table_name RENAME TO new_table_name;
```

Since version: 1.4.1
Or you can rename a specific column of a particular table.

```
ALTER TABLE table_name RENAME COLUMN column_name TO new_column_name;
```

Change Block Size

Since version: 1.4.2
Change block size for a particular table. This may take a while. Column data needs to be copied to tables with a different block size. Changing block size of a column that can have NULL values (is nullable) takes about 500x longer than changing the block size of a column that is marked as UNIQUE (and therefore cannot have NULL values).

```
ALTER TABLE table_name ALTER BLOCK SIZE new_block_size;
```

Drop Column

Since version: 1.4.1
There is also an option to drop column from a specified table.

```
ALTER TABLE table_name DROP COLUMN column_name;
```

Add Constraint

Since version: 1.4.2
You can add constraint on one or multiple columns of a table. You can add one or multiple constraints. Keyword CONSTRAINT_TYPE has to be replaced with one of the constraint from this list. For example, if we want to add NOT NULL constraint on just one column, the command would look like this: "ALTER TABLE table_name ADD NOT NULL constraint_name (column_name);".

```
1  ALTER TABLE table_name ADD CONSTRAINT_TYPE constraint1_name
2    (column1_name [, column2_name, ...])
3    [, ADD CONSTRAINT_TYPE constraint2_name (column1_name [, column2_name, .
```

Drop Constraint

Since version: 1.4.2
You can drop constraint using constraint name. If the constraint name was not explicitly

defined when creating constraint, the default name of the constraint was made up of the column name and constraint suffix, which depends on the constraint type (for more information see chapter create table). It will affect all columns that the constraint applied to. Note that dropping one constraints can be dependent (e.g. you cannot drop NOT NULL constraint when the UNIQUE constraint is still applied on that particular column).

```
ALTER TABLE table_name DROP CONSTRAINT_TYPE constraint1_name
  [, DROP CONSTRAINT_TYPE constraint2_name, ...];
```

DROP TABLE

Since version: 1.4.1
Deletes chosen table. In order to run this command you need to select database first (see Console).

```
DROP TABLE table_name;
```

SQL Data Manipulation

Syntax is case insensitive following this template:

```
KEYWORDS required_input [optional_input];
```

Escaping Keywords

Since version: 1.4.0
If you need to escape database or table or column, because it is named the same as our keyword, (e.g. if you have column named 'count' in your table and COUNT() is also a function), use brackets `[]` to escape it. See the example below:

```
SELECT t.[count] FROM My_Table AS t LIMIT 10;
```

SHOW Statement

SHOW DATABASES

Since version: 1.4.0
Shows loaded databases in memory.

```
SHOW DATABASES;
```

SHOW TABLES

Since version: 1.4.0
Shows tables from database. It is **not** needed to use database to execute this query, but it is possible.

```
SHOW TABLES FROM my_database;
```

Since version: 1.4.0
Also this syntax is valid:

```
SHOW TABLES IN my_database;
```

SHOW COLUMNS

Since version: 1.4.0
Shows columns from table. It is needed to use database to execute this query.

```
SHOW COLUMN FROM my_table;
```

Since version: 1.4.0
Also this syntax is valid:

```
SHOW COLUMNS IN my_table;
```

SHOW CONSTRAINTS

Since version: 1.4.2
Shows constraints - their type and name (id) and also columns to witch they apply.

```
```

```
SHOW CONSTRAINTS FROM table_name;
```

SHOW QUERY COLUMN TYPES

Since version: 1.4.2
Shows columns and column types which will be returned as result when executing that particular query.

```
SHOW QUERY COLUMN TYPES query;
```

SELECT Statement

Since version: 1.4.0

```
SELECT expression [AS alias] [, expression [AS alias] ...]
FROM table [AS alias]
[WHERE expression]
[GROUP BY expression, ...]
[ORDER BY <expression> [ ASC | DESC ] , ...]
[LIMIT number [OFFSET number]];
```

AS

Since version: 1.4.0
Allows you to tag column or table with alias - custom string that you can use from that point instead of original name.

GROUP BY

Since version: 1.4.0
Allows you to group result rows into groups with the same values of the columns defined

after GROUP BY keywords. Often used with aggregate functions.

LIMIT

Since version: 1.4.0
Allows you to get the specified number of records from the whole result.

OFFSET

Since version: 1.4.0
Allows you to skip the specified number of records from the result.

ORDER BY

Since version: 1.4.0
Sorts the result of a query according to the columns defined after ORDER BY keyword in ascending or descending order (can be specified). Default is ascending order.

WHERE

Since version: 1.4.0
Allows you to filter the records according to the specified conditions.

INNER JOIN [Desgined for Star Schema]

Since version: 1.4.0
The INNER JOIN keyword selects records that have matching values in both tables. This is the default type of JOIN so it is not needed to write keyword 'INNER'. This type of JOIN was designed for Star Schema (data mart schema). That means, it was optimized for joining tables with 1:N realtion. It can be used for joining tables with N:M relation as well, but it was not designed for this purpose, and therefore, the performance is weak.

```
SELECT column_1 [, column_2, ...]
FROM table1
[INNER] JOIN table2
```

```
4  ON table_1.column_name = table_2.column_name;
```

INSERT INTO Statement

Since version: 1.4.0
Insert records of data into columns of a table.

```
1  INSERT INTO my_table (column_1 [, column_2, ...])
2  VALUES (value_1 [, value_2, ...]);
```

INSERT INTO String Values

Since version: 1.4.0
When inserting string values via INSERT INTO, use double quotation marks.

```
INSERT INTO my_table (string_column) VALUES ("string value");
```

INSERT INTO Geo Point or Geo Polygon Values

Since version: 1.4.0
When inserting geo point or geo polygon values via INSERT INTO, use double quotation marks.

```
1  INSERT INTO my_table (point_column, polygon_column) VALUES (
2      POINT(15 88.8695),
3      POLYGON((15 10, 22 89.85, 15 10), (89 115, 23 23, 89 115))
4  );
```

Operators

Since version: 1.4.0

Operator	Functionality
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
=	Equal
>	Greater
<	Less
>=	Greater Or Equal
<=	Less Or Equal
<>	Not Equal
!=	Not Equal
>>	Left Shift
<<	Right Shift
AND	Logical AND
OR	Logical OR

NOT	Logical NOT

Geospatial Functions

Since version: 1.4.0
Geospatial functions currently work in planar (Euclidean) mode. Supported types are Point and Polygon. When geo column is selected, it is returned and displayed in well known text format, e.g. `POINT(10.8 14)`,
`POLYGON((30.89 10, 40 40.2, 20 40, 10 20, 30.89 10))`.

Note: Point and Polygon are keywords in SQL so tables or columns have to be named differently.

GEO CONTAINS

Since version: 1.4.0
Returns bool value whether point is in polygon. Both combinations are valid:
`GEO_CONTAINS(wkt_point_or_column, wkt_polygon_or_column)`
`GEO_CONTAINS(wkt_polygon_or_column, wkt_point_or_column)`.

Example:

```
SELECT geo_point FROM geo_table
  WHERE GEO_CONTAINS(
    POLYGON((17.0779 48.1303, 17.0912 48.1303, 17 48, 17.0779 48.1303)),
    geo_point);
```

GEO INTERSECT

Since version: 1.4.0
Calculates intersection of two polygons. Meaning that every point in the intersection appears in both first and second polygon. Returning type is Polygon.

```
GEO_INTERSECT(wkt_polygon_or_column, wkt_polygon_or_column)
```

Example:

```
SELECT GEO_INTERSECT(
  POLYGON((17.0779 48.1303, 17.0912 48.1303, 17 48, 17.0779 48.1303)),
  geo_polygon)
  FROM geo_table;
```

GEO UNION

Since version: 1.4.0
Calculates union of two polygons. Returning type is polygon.

```
GEO_UNION(wkt_polygon_or_column, wkt_polygon_or_column)
```

Typical usage:

```
SELECT GEO_UNION(
  POLYGON((17.0779 48.1303, 17.0912 48.1303, 17 48, 17.0779 48.1303)),
  geo_polygon)
  FROM geo_table;
```

Numeric Functions

ABS

Since version: 1.4.0
Calculates the absolute value of the numeric expression or number.

```
ABS(numeric_expression)
```

ACOS

Since version: 1.4.0
Calculates arc cosine of the numeric expression or number. This numeric expression or number needs to be greater than -1 and less than 1.

```
ACOS(numeric_expression)
```

ASIN

Since version: 1.4.0
Calculates arc sine of the numeric expression or number. This numeric expression or number needs to be greater than -1 and less than 1.

```
ASIN(numeric_expression)
```

ATAN

Since version: 1.4.0
Calculates arc tangent of the numeric expression or number.

```
ATAN(numeric_expression)
```

ATN2

Since version: 1.4.0
Calculates arc tangent of two numeric expressions or numbers.

```
ATN2(numeric_expression_1, numeric_expression_2)
```

AVG

Since version: 1.4.0
Calculates average value of the numeric expression.

```
AVG(numeric_expression)
```

CEIL

Since version: 1.4.0
Calculates an integer value from the numeric expression or number, so the new value is the smallest integer value that is greater or equal to the input numeric expression or number.

```
CEIL(numeric_expression)
```

COUNT

Since version: 1.4.0
Counts the number of records.

```
COUNT(expression)
```

COS

Since version: 1.4.0
Calculates cosine of the numeric expression or number.

```
COS(numeric_expression)
```

COT

Since version: 1.4.0
Calculates cotangent of the numeric expression or number.

```
COT(numeric_expression)
```

EXP

Since version: 1.4.0
Calculates Euler's number raised to the power of the input numeric expression or number.

```
EXP(numeric_expression)
```

FLOOR

Since version: 1.4.0
Calculates an integer value from the numeric expression or number, so the new value is the greatest integer value that is less or equal to input numeric expression or number.

```
FLOOR(numeric_expression)
```

LOG

Since version: 1.4.0
Calculates the natural logarithm of the input numeric expression or number or if there are two input numeric expressions or numbers, calculates logarithm of the first numeric

expression or number to the base defined by the second numeric expression or number. The input numeric expression or number needs to be greater than 0 and base numeric expression or number needs to be greater than 1.

```
LOG(numeric_expression[, base])
```

LOG10

Since version: 1.4.0
Calculates the logarithm of the input numeric expression or number to the base of 10. The input numeric expression or number needs to be greater than 0.

```
LOG10(numeric_expression)
```

MAX

Since version: 1.4.0
Returns the maximum value.

```
MAX(numeric_expression)
```

MIN

Since version: 1.4.0
Returns the minimum value.

```
MIN(numeric_expression)
```

PI

Since version: 1.4.0
Provides the value of π (3.141592F).

```
PI()
```

POW

Since version: 1.4.0
Calculates the first input numeric expression or number raised to the power of the second numeric expression or number.

```
POW(base, exponent)
```

ROUND

Since version: 1.4.0
Rounds the input numeric expression or number to the number of decimal input.

```
ROUND(numeric_expression, decimal)
```

ROOT

Since version: 1.4.0
Calculates the n-th root from the input numeric expression or number.

```
ROOT(numeric_expression, n)
```

SIGN

Since version: 1.4.0
Returns the sign of the numeric expression or number.

```
SIGN(numeric_expression)
```

SIN

Since version: 1.4.0
Calculates sine of the numeric expression or number.

```
SIN(numeric_expression)
```

SQRT

Since version: 1.4.0
Calculates square root of the input numeric expression or number.

```
SQRT(numeric_expression)
```

SQUARE

Since version: 1.4.0
Calculates square of the input numeric expression or number.

```
SQUARE(numeric_expression)
```

SUM

Since version: 1.4.0
Calculates the sum of the input numeric expression.

```
SUM(numeric_expression)
```

TAN

Since version: 1.4.0
Calculates tangent of the input numeric expression or number.

```
TAN(numeric_expression)
```

Advanced Functionality

CAST

Since version: 1.4.0
Converts a value to a specified data type. The possible casting combinations are:

- Bool → Bool, Double, Float, Int, Long, String
- Double → Bool, Double, Float, Int, Long, String
- Float → Bool, Double, Float, Int, Long, String
- Int→ Bool, Double, Float, Int, Long, String
- Long → Bool, Double, Float, Int, Long, String
- String → Bool (any string is casted to false except for string "true" - case insensitive), Double, Float, Geo_Point, Int, Long, String
- Geo_Point → String
- Geo_Polygon → String

One more casting combinations will be added in the near future (string to geo_polygon).

```
CAST(some_value AS data_type)
```

IS NOT NULL

Since version: 1.4.0
Allows you to get the not null values.

```
SELECT column_1 [, column_2, ...]
FROM table
WHERE column IS NOT NULL;
```

IS NULL

Since version: 1.4.0
Allows you to get the null values.

```
SELECT column_1 [, column_2, ...]
FROM table
WHERE column IS NULL;
```

Date Functions

FORMAT OF THE DATE TYPE:
year-month-day hour:minute:second

DATE

Since version: 1.4.0
Returns the datetime in human readable format (*year-month-day hour:minute:second*) from Integer64 data type.

```
DATE(datetime_represented_as_integer64)
```

Also addition of two datetimes represented as Integer64 would work inside this function like this:

```
DATE(datetime1_represented_as_integer64 + datetime2_represented_as_integer64
```

DAY

Since version: 1.4.0
Returns the day from the input date, which is in the range from 1 to 31.

```
DAY(date)
```

DAYOFWEEK

Since version: 1.5.0
Returns integer 1-7 according to daytime (NOTE: returned values are different from function WEEKDAY).

Day	Value
Monday	2
Tuesday	3
Wednesday	4

Thursday	5
Friday	6
Saturday	7
Sunday	1

It works with both long or string, e.g.:

```
SELECT DAYOFWEEK(4564545);
SELECT DAYOFWEEK("2020-04-26");
```

HOUR

Since version: 1.4.0
Returns the hour from the input date, which is in the range from 1 to 24.

```
HOUR(date)
```

MINUTE

Since version: 1.4.0
Returns the minute from the input date, which is in the range from 1 to 60.

```
MINUTE(date)
```

MONTH

Since version: 1.4.0
Returns the month from the input date, which is in the range from 1 to 12.

```
MONTH(date)
```

NOW

Since version: 1.4.0
Returns the current date and time.

```
NOW()
```

SECOND

Since version: 1.4.0
Returns the seconds from the input date, which is in the range from 1 to 60.

```
SECOND(date)
```

WEEKDAY

Since version: 1.5.0
Returns integer 0-6 according to daytime (NOTE: returned values are different from function DAYOFWEEK).

Day	Value
Monday	0
Tuesday	1
Wednesday	2
Thursday	3

Friday	4
Saturday	5
Sunday	6

It works with both long or string, e.g.:

```
SELECT WEEKDAY(4564545);
SELECT WEEKDAY("2020-04-26");
```

String Functions

Syntax consists from:

```
KEYWORDS required_input [optional input];
```

CONCAT

Since version: 1.4.0
Merges two strings together.

```
CONCAT(string_1, string_2)
```

LEFT

Since version: 1.4.0
Extracts specified number of chars from left side from the input string.

```
LEFT(string, charCount)
```

LEN

Since version: 1.4.0
Returns the length of the input string.

```
LEN(string)
```

LOWER

Since version: 1.4.0
Transforms the input string to lower case.

```
LOWER(string)
```

LTRIM

Since version: 1.4.0
Removes leading spaces from the input string.

```
LTRIM(string)
```

REVERSE

Since version: 1.4.0
Reverses the input string.

```
REVERSE(string)
```

RIGHT

Since version: 1.4.0
Extracts the specified number of characters from the right side from the input string.

```
RIGHT(string, characters_count)
```

RTRIM

Since version: 1.4.0
Removes trailing spaces from the input string.

```
RTRIM(string)
```

UPPER

Since version: 1.4.0
Transforms the input string to upper case.

```
UPPER(string)
```

Data Types

Since version: 1.4.0

Each column of a table has to have assigned data type. The qikkDB supports boolean, numeric types, strings, date type and geometry types.

Data Type	Column Type Code	Size	Description
Bool	15	1 byte	Representing TRUE or FALSE values
Int	8	4 bytes	Representing integer in range -2,147,483,648 to 2,147,483,647
Long	9	8 bytes	Representing integer in range -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Float	10	4 bytes	Representing floating-point numbers (precision 7 digits)
Double	11	8 bytes	Representing floating-point numbers (precision 15 digits)
Geo_Point	12	8 bytes	Representing geometry point with latitude (Float - 4 bytes) and longitude (Float - 4 bytes)
Geo_Polygon	13	variable	Representing polygon as a series of points with x and y. It can contain also complex polygons that are composed of multiple polygons.
String	14	variable	Representing text values with length up to 2,147,483,647 characters
Date	9	8 bytes	Representing date and time. Format: year-month-day hour:minute:second

Console Client

Running the Console Client

Linux

Console client is located in ./console/QikkDB.ConsoleClient. Go to directory ./console. Run Console Client using command `dotnet ./QikkDB.ConsoleClient.dll`. Console have to be started **after** the successful start of database core. Start by typing command `help` to see all the options that console supports (the list of supported console commands is documented here as well).

Windows

Console client is located in .\console\QikkDB.ConsoleClient.exe. Just double click on it or run it using powershell or cmd (`.\QikkDB.ConsoleClient.exe`). Console have to be started **after** the successful start of database core. Start by typing command `help` to see all the options that console supports (the list of supported console commands is documented here as well).

Specify host

Default host is 127.0.0.1:12345 (localhost). You can specify host using argument *-h ip:port* when starting the console client. For example (Windows) like this: `.\QikkDB.ConsoleClient.exe -h 86.110.229.2:8999`.

Specify timeout

When the conection with database server is lost, the console client will try to established this connection again. The timeout represents, how long (ms) shall the console client be trying to establish connection with database server again. If the connection is still not established when the timeout is reached, the console client will close itself. Default value is 30,000 ms. You can specify timeout using argument *-t value* when starting the console client. For example (Windows) like this. `.\QikkDB.ConsoleClient.exe -t 50000`.

Commands

Console supports following commands or directly written SQL queries. Syntax of commands is case insensitive.

Use, U

```
USE database_name, U database_name
```

Since version: 1.4.0
Prior to running the query, it is necessary to specify database, by Use command. This changes context to all subsequent queries.

If you are not sure about the names of imported databases, enter `SHOW DATABASES` command.

If there is no database, it should be created by running query `CREATE DATABASE` .

Query

```
[QUERY] sql_query
```

Since version: 1.4.0
Query itself written according to currently supported SQL syntax for data definition or data modification. Query command is optional and SQL query can be directly written to the console client.

If you are not sure about tables and columns in the database in use, enter `SHOW TABLES` and `SHOW COLUMNS FROM table_name` .

Script, S

```
SCRIPT file_path, S file_path
```

Since version: 1.4.2
Runs SQL queries from a file path (also supports console client command USE and IMPORT). Commands in script file are seprated by new line, so every command has to start on a new line and end on the line it started. If you want to skip a command or a query in a script file, just make that command a single line comment (in sql, single line comments starts with double dash '–').

Import, I

Since version: 1.4.2
Import currently support files in CSV and TSV format or text files in tabular structure with columns separated by a character and rows separated by a newline.

```
IMPORT database_name filename_with_path
  [table-name=string_value]
  [block-size=integer_value]
  [has-header=boolean_value]
  [column-separator=character_value]
  [threads-count=integer_value]
  [batch-size=integer_value]
```

The database has to exists before importing into it, but it does not need to be in use (command 'use').

If the database contains the table which has the same name as the input file (without extension) or the same name as specified by tablename argument, the imported data will be appended into this existing table. Data values parsed from the input data file will be casted into types specified by definition of the table.

Note: When importing POLYGON data, it needs to be escaped using double quotes, because WKT of POLYGON uses commas as delimiter.

If the table does not exists, the table will be created and it's name will be the name of the imported data file (without extension).The types will be automatically guessed.

If optional arguments are not provided, default or automatically guessed values will be used.

Argument	Description	Default value
`table-name`	Name of the table into which the data will be imported. If the table does not exists, the new table will be created. If table already exists, the data will be appended.	The name of the data file, which is being imported without the file extension
`block-size`	Block size of the imported table (see Block size setting). Input value is an interger.	Value inherited from the database in which the data will be imported
`has-header`	Whether the first line of the input file is header. Input value is `true` or `false`.	`true`
`column-separator`	While parsing the input file, this single character determines how rows are splitted (columns of tabular data). Most common values are `,` or `;`	Automatically guessed
`threads-count`	Number of threads parsing the input file and importing the parsed data. Input value is an integer.	Number of CPU cores of the client machine
`batch-size`	Number of lines of the input file parsed and imported in a single batch.	100 000

Example of usage:

```
IMPORT test_database "C:\temp\sample.csv"
```

```
IMPORT test_database "C:\temp\sample.csv" block-size=200000 has-header=fal
  column-separator=, batch-size=10000
```

Timing, T

Since version: 1.4.1
Runs a query 201 times and prints the first run execution time and average execution time from 200 cached query runs.

Docs, Man

Since version: 1.4.0
Prints URL link where this documentation can be found.

Help, H

Since version: 1.4.0
Provides information about supported commands and their syntax.

Exit, Quit, Q

Since version: 1.4.0
Exits the console program

Database Shutdown

Windows & Linux

Since version: 1.4.0
We recommend to kill the database using **CTRL + C**, because that way, the modified columns will be saved on disk.

Tips & Tricks

Very Fast Importing of a Single CSV File (Single Table)

When starting server side (database core), you can specify arguments to import local CSV file. This way of importing is **faster than web client's importer** (because there is no networking to slow it down), but can **only be performed when starting server** side and is limited to **import only single CSV file (database with only one table)**. The reason for this is, that this feature is there for **backup purposes only**, when web client CSV importer is not available for some reason. So, how to do it? You need to open command line interface (powershell or cmd on Windows) and write the following command:

Linux

```
./dropdbase_instarea path_to_csv_file [database_name_optional] [block_size_o
```

Windows

```
.\dropdbase_instarea.exe path_to_csv_file [database_name_optional] [block_si
```

You do not have to specify database name and block size. Database block size will be used as table block size as well. If database name is not specified, the default value '**TestDb**' will be used. If block size is not specified, the default value '**1048576**' will be used.

Faster query execution time

Query

Use only those columns, that you truly need. It takes more memory and takes more time to execute `SELECT * FROM Table;` if a table has many columns than `SELECT ColumnA, ColumnB FROM Table;`

Data Types

If there are string enumerates, that can be internally represented as integer in database, save this enumeration in integer format for much better query performance.

Block Size

If there are minimum cache misses, the optimal block size (in terms of performance) is the largest one, that can be stored in the single graphic card (read Hardware Limitations - Block Size). Otherwise it depends on each query and data and the only way to find optimal block size is to try different block sizes on the same set of queries and choose the best one. Generally speaking - smaller block sizes improve stability (it is less likely that the query fails due to not having enough GPU VRAM), but sacrifies a bit of speed (we are talking in terms of roughly 5% - 15%). Larger blocks are better for performance, but it is more likely, that the query fails due to not having enough GPU VRAM.

GPU VRAM Cache

In the main configuration file, there is entry called "GPUCachePercent". Set it's value to the highest possible, but still leave as many GPU VRAM percent as needed for intermediate results for the most difficult query you expect to use. You can calculate this roughly like number of rows in temporary columns (which are created, e.g. when you extract year from datetime, then the year column is Int32 type) times the number of bytes needed for one value. So if we have 1 million rows of data and we create a new column which is Int32 type, then we need for intermediate results 1 000 000 rows x 4 bytes = 3.81 MB of memory space. Aliases do not create new columns, so they do not need extra memory. That being said, usually you need about 500 MB of GPU VRAM per 1 billion rows of data in 7 columns for intermediate results. So we usually the value for GPUCachePercent is in range 80 - 95 (percent).

Hardware

Better hardware means better performance, see chapter Hardware Impact on Performance to see, what hardware components are the most important in terms of speed.

Stop Auto-Loading a Database

If you do not want to load a specific database from your databases directory, just change **.db** extension of the main database file to something else (e.g. **.dbx**).

Hardware Requirements

Hardware Limitations

Graphics Processing Unit

You will need an NVIDIA graphic card with **Compute Capability 6.0** or higher. You can find a list of NVIDIA graphic cards and their compute capabilities here.

Model example of GPU Optimized Instance:
GPUs: 4x NVIDIA Tesla™ V100 16G Passive GPU
CPU: 2x Intel® Xeon® Platinum 8253 2.2G, 16C/32T, 10.4GT/s, 22M Cache, Turbo, HT (125W) DDR4-2933
Memory: 30x 8GB (total: 240GB) RDIMM, 2666MT/s, Single Rank
Total cost: 62,850.16 $

Block Size

The default block size is defined in configuration file, but it can be also specified for each database and table using CREATE or ALTER commands. The data is divided into blocks of data. The block size represents how many rows of the column will be stored in one block of data. The theoretical maximum block size is 2,147,483,647. The block size is limited by graphic card's VRAM. The single block cannot be divided into multiple GPUs, so **the single graphic card has to have enough VRAM to store at least one block of data**. The data type of a block has to be taken into account as well. The Integer32 block of data naturally needs a smaller amount of VRAM than block of data with Integer64 data type. It is possible to calculate optimal block size, but there is a lot of things that affect it. In our experience, it is faster to find the maximum block size empirically than to calculate it. If you want to know, how to setup the optimal block size according to your data and queries, read Tip & Tricks - Block Size.

Memory Requirements

Operation Memory (RAM) Limitations

Currently there are some RAM capacity limitations. The first one can be spotted when importing large databases from **.csv** files. The second one is during the operation of database.

Importing Large .CSV Files

When you want to import a large database, it is needed to divide the single large .csv file into multiple smaller parts and import them sequentially. We will support splitting the large .csv file into smaller ones and importing them sequentially on our side (so that the user will not have to do this) in the near future. For the reason of copying data, you will need **the amount of free memory equivalent to about 2x - 2.5x the size of the CSV file that is being imported**. The GPU VRAM memory can also cause an issue (fail of the import), especially if the CSV file, which is being imported, has a lot of columns. Operation persistance currently needs even more RAM than import and therefore when importing large CSV file, we strongly recommend to split it to multiple smaller parts. Sometimes importing of large CSV file can be successful, but persisting this amount of new rows of data can fail on not having enough free RAM.

Operation of Database

Currently we do not support swapping data into disk, so there has to be enough free space of RAM to handle all of the data. For the correct operation of the database core (server side), you will need **the amount of free memory equivalent to 1.5x - 2x the size of saved database files (.db, .col) on disk or to simplify it even more - about 1.5x - 2x the size of the CSV file that was imported**. In the future, we will support the functionality to load data from disk on demand and we will reduce the memory usage. For example, 1.2B rows of Integer_32 data in one column takes 4.7 GB of disk space and if loaded into memory, it takes about 7.05 GB of memory (RAM) capacity. We are currently working on reducing memory (RAM) usage when database is loaded.

How Block Size Affects Memory Usage

The block size is set per database (this is will be used as the default value for tables which do not have specified their block sizes) **and per each table**. If the block size for a particular table is specified, this specified value will be used as block size per that table. The block size cannot be set per columns. It is ideal to have **large block size for tables with huge**

amounts of data and **small block size for tables with small amounts of data**. If you set a huge block size, e.g. 300M, for a table which has small amount of data, e.g. 10K rows, there will be allocated a block with size of 300M for those 10K rows of data, which means it will have a **huge comsuption of memory (RAM) pointlessly**. So how to set a block size? If you have columns in a table which have a similar amounts of data and you do not use indexing, the best block size is set as splitting the data into as many blocks as there are GPU cards. Let show it on an exaple: If we have database with just one table with 3 columns, each column have 500M rows of data and you have 2 graphic cards available for querying and the indexing is turned off, the best block size is the largest one, which means 500M/2 GPUs = 250M block size. There is a limitation that **the whole block has to fit into VRAM** (GPU memory). If it does not fit, you have to choose a smaller block size. **If you use indexing**, we recommend you to **choose block size empirically based on experimenting with different block sizes**, because data has a huge impact on performace when indexes are used.

Hardware Impact on Performance

The following hardware components are ordered from the most important to the least important component in terms of speed impact.

GPU

It is very important to have a graphic processing unit with the following specifications (order from the highest important to the lowest):

- a lot of stream multiprocessors,
- a large bus width,
- high throughput,
- the architecture is important, we recommend to have at least a Pascal architecture or newer, but it will work with the older architectures as well,
- a lot of VRAM.

RAM

The volume of RAM is more important than its speed.

CPU

The most important in terms of speed is to have a CPU with at least as many cores as there are graphic cards, because each core handles one graphic card. Once you meet this condition, then the frequency of the cores becomes the second important parameter.

STORAGE

When there is not enough memory (RAM) for handling all the data, the database starts swapping data to disk and that is when a speed of a disk starts to be important in terms of speed of the queries. The disk speed is always important when data are loading from disk to memory and saving from memory into disk.

Known Issues

- **INNER JOIN can join only two tables**

  We will support joining multiple tables in the future,

- **Multiple aggregation functions when using GROUP BY does not work**

  We will support it in the next release.

- **Casting cannot be done in JOIN clause**

  We are aware of this but we do not plan to add it soon. It is possible to support it but it is difficult to implement in our solution.

- **Aggregations without GROUP BY do not work with NULL values**

  We will fix this in the next release.

- **The order of table columns after importing a database from a CSV file may be different from CSV file**

  This would require too much work to fix and we do not consider it as an important issue, so we do not plan to fix this. However, if you consider it as a major issue, let us know and we will reconsider.

- **The function IS NULL does not work together with other conditions in WHERE clause (e.g. colA = 0 OR colA IS NULL)**

  We will fix this in the next release.

- **All data has to be in the memory (RAM), if data cannot fit in there, it will fail**

  Currently, we do not support lazy loading.

TellStory Visualization Tool

What is TellStory?

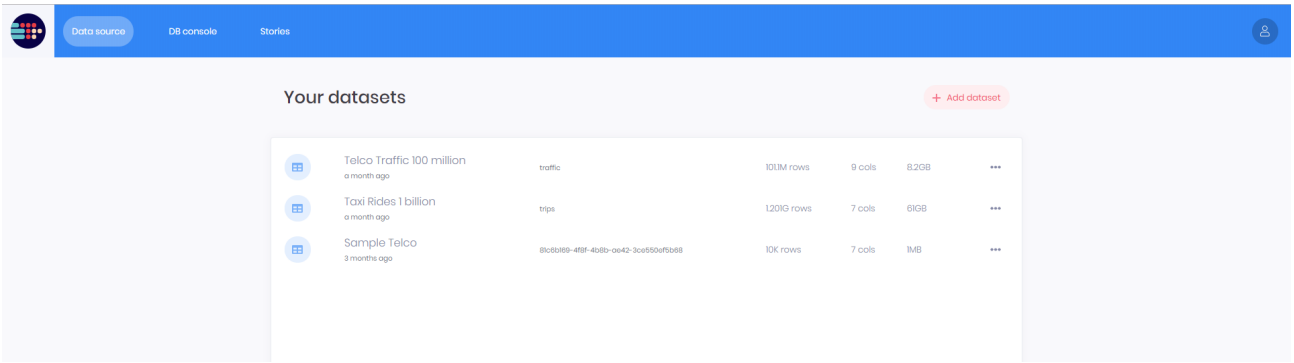
TellStory is a web application that helps you to create interactive stories over massive datasets. Thanks to underlying ultra fast database qikkDB it enables to present charts and maps based on billions of rows in realtime.

Currently, **TellStory is running publicly for demonstration purposes of the underlyng qikkDB**. Our vision is to make a presentation tool which will help users without programming background to simply create visually stunning stories and share with others. It has two parts - one for managing datasets and another for creating story presentation.

Visit TellStory here. To request a demo, you need to go through a short registration process.

Datasets Management

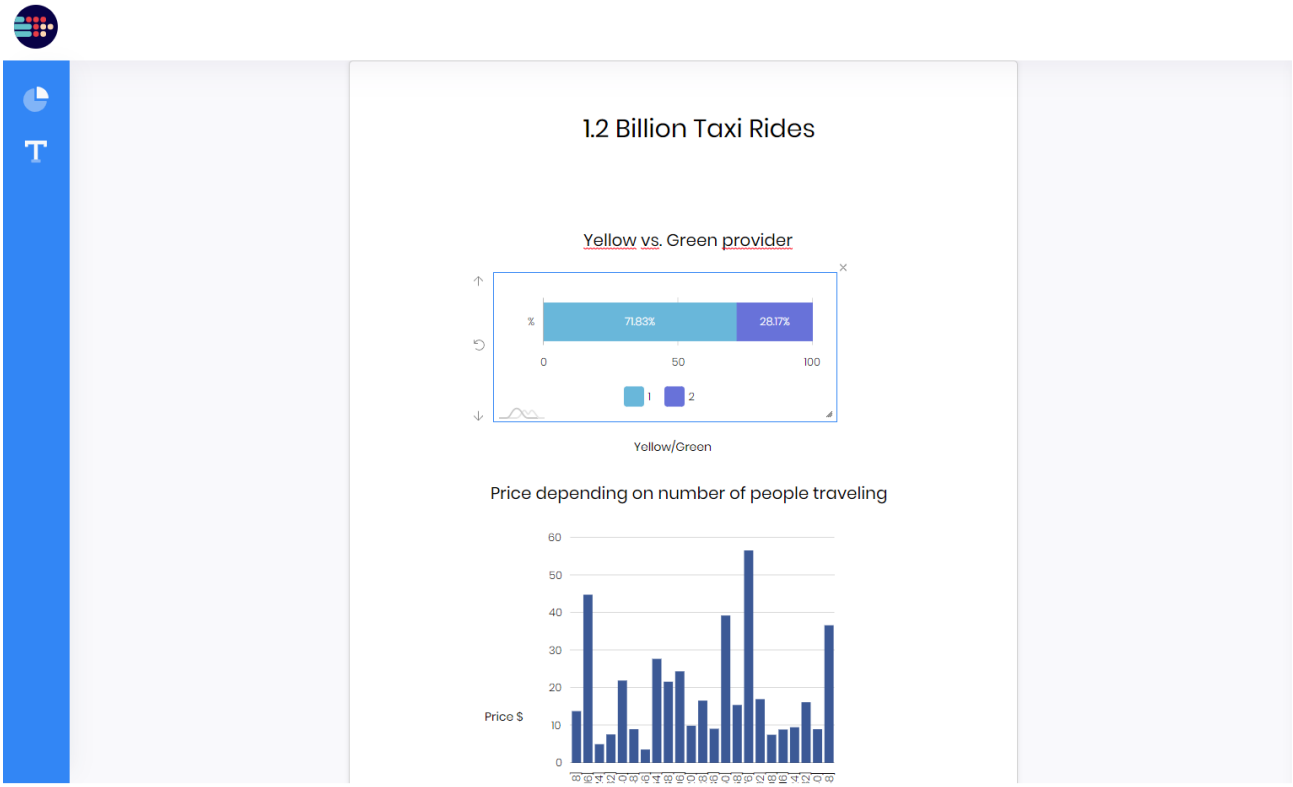
This part allows you to import dataset from various sources and to manage imported datasets. Currently, you can import CSV file from URL link or upload from local computer. Imported datasets are in tabular format - structure is defined by columns and data are stored as rows.



Datasets management part of TellStory.

Story Desinger

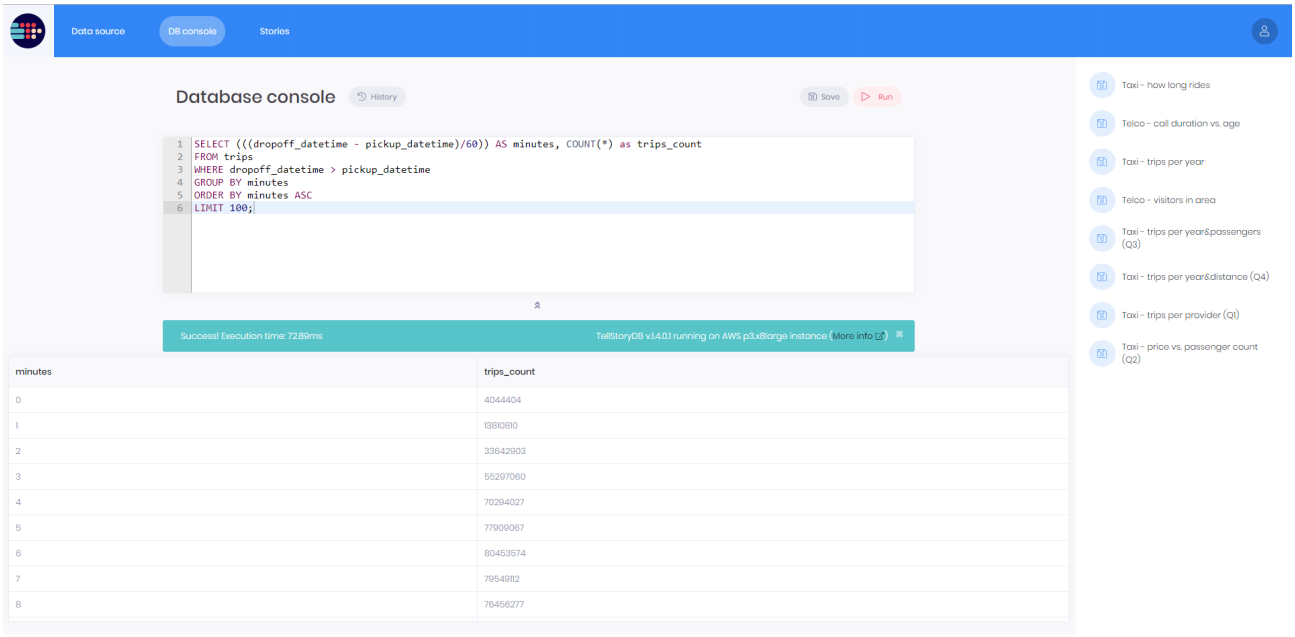
Once you have imported a dataset you can start creating a story. Insert chart widgets to provide insight into data by configuring source dataset and columns used for visualizations. Or insert text widgets to describe the chart widgets.



Story designer part of TellStory.

Web Console

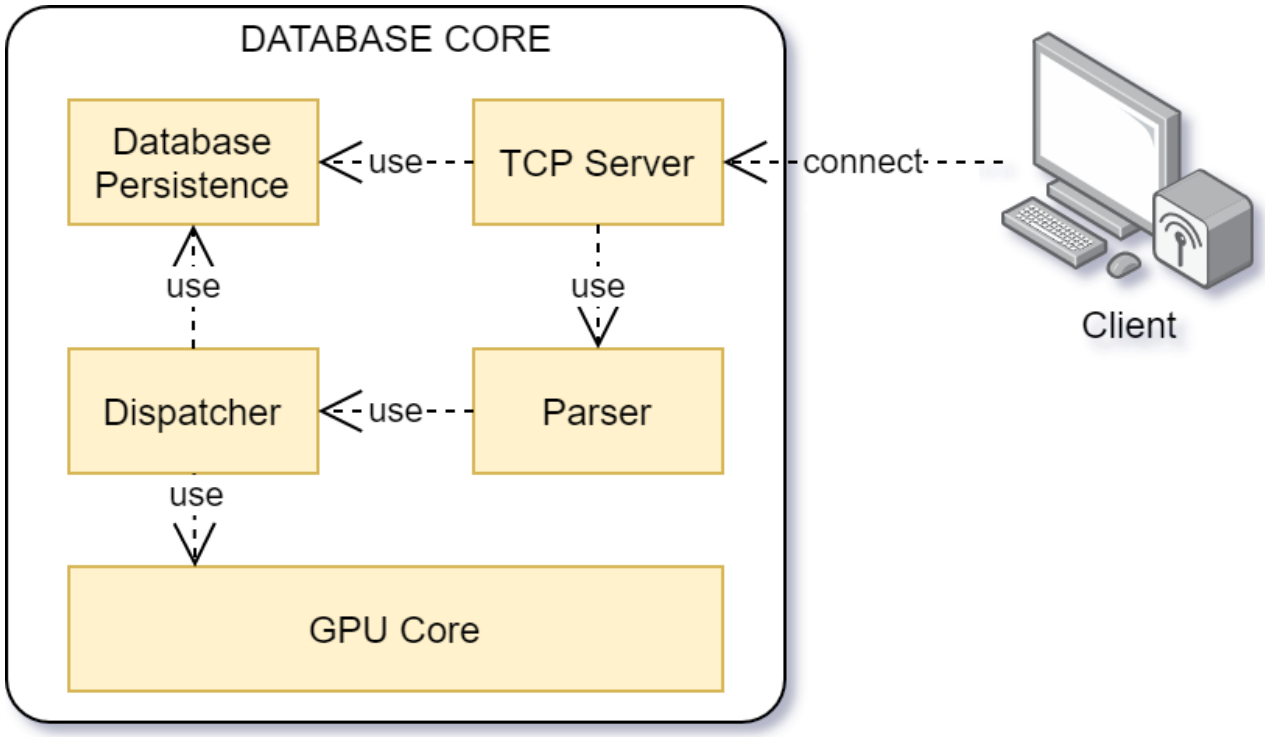
This is part for more advanced users who want to run SQL queries over their datasets. It allows SELECT queries while other types of queries are blocked for security reasons.



Web console part of TellStory.

Developer Guide

Overview



Architecture of a system that uses our database

CMake configuration

QikkDB is compiled using the **CMake** build system.

The build process is described using three **CMakeList** configuration files:

- *CMakeList.txt* - Main CMake configuration file
- *qikkDB/CMakeList.txt* - Configuration file used to build qikkDB library and qikkDB executable
- *qikkDB_test/CMakeList.txt* - Configuration file used to build qikkDB_test executable linked with the qikkDB library

All **QikkDB dependacies** except Boost 1.66 are built using **CMake external projects**. The definitions and configurations of these external projects are located in the main CMakeList file.

The *qikkDB/CMakeList.txt* file contains list of the C++ and CUDA source files used to build **qikkDB** executable. The build of **qikkDB** executable consists of these steps:

1. Compilation of the source files to *qikkDB_lib* OBJECT library target dependant on the: *protobuf-external, yaml-cpp-external, antlr4_static, jsoncpp-external* targets
2. Creation of *qikkDB_obj* STATIC library target from the OBJECT library
3. Creation of *qikkDB* executable target
4. Linking of *qikkDB_obj* STATIC library to the qikkDB executable
5. Linking of external project libraries to the *qikkDB_obj* STATIC library

The qikkDB_test/CMakeList.txt file contains list of the C++ and CUDA test source files used to build qikkDB_test executable. The build of qikkDB executable consists of these steps:

1. Compilation of the test source files to *qikkDB_test* executable
2. Linking of the *GTest* external project library to the *qikkDB_test* executable
3. Linking of *qikkDB_obj* STATIC library to the *qikkDB_test* executable

All external project libraries are build and linked statically.

Networking Protocol

Google Protocol Buffers

Networking uses Google Protocol Buffers messages (https://developers.google.com/protocol-buffers) to communicate with the database, which are used to serialize data. They allow you to define the structure of .proto messages that are then compiled by Protoc (Protobuf Compiler) into the selected language. You can download Protoc here - https://developers.google.com/protocol-buffers/docs/downloads, version used in QikkDB is protoc 3.10.0 (https://github.com/protocolbuffers/protobuf/releases/tag/v3.10.0).

Protobuf Compiling

The compilation of protobufs differs depending on the language required, but in general the command applies:

```
protoc -I=$SRC_DIR --language=$DST_DIR $SRC_DIR/addressbook.proto
```

Language	Output
C++	cpp_out
C#	csharp_out
Dart	dart_out
Go	go_out
Java	java_out
Python	python_out

In case of C# this compilation for QueryResponseMessage.proto could look like this:

```
protoc -I=..\GPU-DB\QikkDB.NetworkClient
--csharp_out=..\GPU-DB\QikkDB.NetworkClient\Message
..\GPU-DB\QikkDB.NetworkClient\Message\QueryResponseMessage.proto
```

For more information about compiling protobufs and tutorials see - https://developers.google.com/protocol-buffers/docs/tutorials

Structure of Protobuf Messages

Syntax used for all messages is proto3.

BulkImportMessage

This type of message is used during importing data into the database. The data are sent in multiple smaller batch parts.

```
message BulkImportMessage {
    string TableName = 1;
    string ColumnName = 2;
    DataType ColumnType = 3;
    int32 ElemCount = 4;
    int32 nullMaskLen = 5;
    int32 dataLength = 6;
}
```

CSVImportMessage

This type of message is used during importing data into the database from a .csv file.

```
message CSVImportMessage {
    string DatabaseName = 1;
    string CSVName = 2;
    string Payload = 3;
    repeated DataType ColumnTypes = 4;
}
```

This .proto file also defines enum DataType, which defines a constant value for each data type that we support - type for constants and columns, error flag and size of datatype.

```
enum DataType {
    CONST_INT = 0;
    CONST_ERROR = -1;
    CONST_LONG = 1;
    CONST_FLOAT = 2;
    CONST_DOUBLE = 3;
    CONST_POINT = 4;
    CONST_POLYGON = 5;
    CONST_STRING = 6;
    CONST_INT8_T = 7;
    COLUMN_INT = 8;
    COLUMN_LONG = 9;
    COLUMN_FLOAT = 10;
    COLUMN_DOUBLE = 11;
    COLUMN_POINT = 12;
    COLUMN_POLYGON = 13;
    COLUMN_STRING = 14;
    COLUMN_INT8_T = 15;
    DATA_TYPE_SIZE = 16;
}
```

InfoMessage

This type of message is used for getting status information.

```
message InfoMessage {
    enum StatusCode
    {
        OK = 0;
        WAIT = 1;
        GET_NEXT_RESULT = 6;
        QUERY_ERROR = 2;
        IMPORT_ERROR = 3;
        CONN_ESTABLISH = 4;
        CONN_END = 5;
        HEARTBEAT = 7;
    }
    StatusCode Code = 1;
    string Message = 2;
}
```

QueryMessage

This type of message is used to transmit the specified query.

```
message QueryMessage {
    string Query = 1;
}
```

QueryResposeMessage

This type of message is used to transmit the result of the specified query.

```
message QueryResponseIntPayload {
    repeated int32 intData = 1;
}

message QueryResponseInt64Payload {
    repeated int64 int64Data = 1;
}

message QueryResponseDateTimePayload {
    repeated int64 dateTimeData = 1;
}

message QueryResponseFloatPayload {
    repeated float floatData = 1;
}

message QueryResponseDoublePayload {
    repeated double doubleData = 1;
}

message QueryResponsePolygonPayload {
    repeated QikkDB.Types.ComplexPolygon polygonData = 1;
}

message QueryResponsePointPayload {
    repeated QikkDB.Types.Point pointData = 1;
}

message QueryResponseStringPayload {
```

```
    repeated string stringData = 1;
}

message QueryNullmaskPayload {
    repeated uint64 nullMask = 1;
}

message QueryResponsePayload {
    oneof payload {
        QueryResponseIntPayload intPayload = 2;
        QueryResponseFloatPayload floatPayload = 3;
        QueryResponseInt64Payload int64Payload = 4;
        QueryResponseDoublePayload doublePayload = 5;
        QueryResponsePointPayload pointPayload = 6;
        QueryResponsePolygonPayload polygonPayload = 7;
        QueryResponseStringPayload stringPayload = 8;
        QueryResponseDateTimePayload dateTimePayload = 9;
    }
}

message QueryResponseMessage {
    map<string, QueryResponsePayload> payloads = 1;
    map<string, QueryNullmaskPayload> nullBitMasks = 3;
    map<string, float> timing = 2;
    repeated string columnOrder = 4;
}
```

SetDatabaseMessage

This type of message is used to transmit the name of the database, which we want to work with.

```
message SetDatabaseMessage {
    string DatabaseName = 1;
}
```

Custom Data Types

To define our custom type such as Point a Polygon, four messages of structure below are needed.

```
message GeoPoint {
    float latitude = 1;
    float longitude = 2;
}

message Point {
    GeoPoint geoPoint = 1;
}

message Polygon {
    repeated GeoPoint geoPoints = 1;
}

message ComplexPolygon {
    repeated Polygon polygons = 1;
}
```

Exceptions

The custom exception in the networking client is **QueryException**, which represents an error that occurs during query execution or during data import

The exception parameter is a message of type string to be displayed.

Networking Client

After defining and compiling the necessary Protobuf messages, the implementation of communication with the database using these messages follows. Communication takes place on the basis of a TCP connection (ip Address = 127.0.0.1, port = 12345).

CONNECTION TO THE DATABASE

The first step in connecting to a database requires opening a TCP connection. The next step is to send an **InfoMessage**, with the parameters: *Code = ConnectionEstablish* and an empty string representing the *Message = “” “* being sent.

The expected answer from the database is **InfoMessage** with Code = Ok. If the returned message type does not match, or the *Code* for the correct message type does not, then the expected behavior is to throw an **IOException** with *"Invalid response received from server"* message.

If anything fails during this process, the TCP connection should be closed.

USE DATABASE

After creating a connection with the database, another functionality is selecting a specific database to work with (adding a table, changing the contents of the table, alter columns of the table, making queries, ...)

You need to create and send through the open connection a **SetDatabaseMessage** with the *DatabaseName* parameter set to the name of the selected database.

The expected answer from the database is **InfoMessage** with *Code = Ok*. If the returned message type does not match, then the expected behavior is to throw an **IOException** with *"Invalid response received from server"* message. If the *Code* of the returned **InfoMessage** message does not match *Ok*, then the expected behaviour is to throw a **QueryException** with a message set to the message of returned **InfoMessage**.

If anything fails during this process, the TCP connection should be closed.

QUERY

To create a query over the selected database, it is necessary to create a **QueryMessage** with the *Query* parameter set on the desired query in the form of a string and send it through the open connection.

There are two expected messages that are considered as a response to **QueryMessage**.

First expected answer from the database is **InfoMessage** with *Code = Wait*. If the returned message type does not match,then the expected behavior is to throw an **IOException** with *"Invalid response received from server"* message.

If the type and Code parameter of the first **InfoMessage** is correct, then we expect a second **InfoMessage**. If the returned message type does not match, then the expected behavior is to throw an **IOException** with *"Invalid response received from server"* message. If the Code of the returned **InfoMessage** message does not match *Get Next Result*, then the expected behaviour is to throw a **QueryException** with a message set to the message of returned **InfoMessage**.

If anything fails during this process, the TCP connection should be closed.

GET NEXT QUERY RESULT

After successfully sending a query and subsequent database responses, it is necessary to obtain data corresponding to the query.

The initial step is to send through the open connection **InfoMessage** with parameters *Code = Get Next Result* and an empty string as parameter *Message = ' '*.

The expected answer is **QueryResponseMessage** with the result of the selected query from the chosen database. This message has information about correct Column Order to display, queried data in *Payload* parameter and its nullmasks in *Null Bit Mask* parameter. Last parameter *Timing* tells about the time duration of individual components of the executed query.

If the returned message is **InfoMessage** with different *Code* as *Ok*, then the expected behaviour is to throw a **QueryException** with a message set to the message of returned **InfoMessage**. If the *Code* of **InfoMessage** is *Ok*, there are no other results of this query.

If anything fails during this process, the TCP connection should be closed.

IMPORT CSV

For importing a csv file there is a **CSVImportMessage**, with necessary parameters like *Database Name* which represents chosen name of the database, *CSV Name* which represents chosen csv file name and the data itself in *Payload* parameter.

Payload is a string of data that is retrieved from a csv file by loading individual lines that are separated by a line break character *"\n"*. (Keep in mind string length limits and load csv in sections, eg 1000 lines or use Bulk Import)

The expected answer from the database is **InfoMessage** with *Code = Ok*. If the returned message type does not match, then the expected behavior is to throw an **IOException** with *"Invalid response received from server"* message. If the *Code* of the returned **InfoMessage** message does not match *Ok*, then the expected behaviour is to throw a **QueryException** with a message set to the message of returned **InfoMessage**.

If anything fails during this process, the TCP connection should be closed.

BULK IMPORT

During bulk import, **BulkImportMessage** is used, which does not carry data by itself, it only describes the data, which is then sent in bulks in the form of bytes via the TCP connection. One bulk has defined size.

The parameter *Table Name* as a name of chosen table, *Element Count* as a sum of data of concrete type in one bulk, *Column Name* as a name of chosen column, *Column Type* as datatype of chosen column, *Null Mask Length* in Bytes (if there is no nullmask this parameter is set to null) *Data Length* in Bytes is added to the message. Subsequently, raw Bytes are sent for the data and a nullmask, if a nullmask exists.

The expected answer from the database is **InfoMessage** with *Code = Ok*. If the returned message type does not match, then the expected behavior is to throw an **IOException** with *"Invalid response received from server"* message. If the *Code* of the returned **InfoMessage** message does not match *Ok*, then the expected behaviour is to throw a **QueryException** with a message set to the message of returned **InfoMessage**.

If anything fails during this process, the TCP connection should be closed.

HEARTHBEAT

InfoMessage with *Code = Hearthbeat* and empty *Message = " "* is used to send it regularly and maintain a connection to the database.

The expected answer from the database is **InfoMessage** with Code = Ok. If the returned message type does not match, or the Code for the correct message type does not, then the expected behavior is to throw an **IOException** with "Invalid response received from server" message.

If anything fails during this process, the TCP connection should be closed.

CLOSE

InfoMessage with Code = CnnectionEnd and an empty string as Message = “ ” is used to close the connection. It is recommended to use a delay before sending the message itself for this operation. Subsequently, it is necessary to close the TCP connection.

If anything fails during this process, the TCP connection should be closed.

TCP Server

TCP Server is implemented in *TCPServer* class and can be created using constructor that have three arguments:

- IP address,
- port,
- flag if database should be saved automatically.

After constructing an instance, the server can be started by calling its *Run* function. This functions asynchronously runs TCP acceptor (*boost::asio::ip::tcp::acceptor*). The acceptor uses *ClientPoolWorker* and its function *HandleMessage* that distinguishes between these message types:

- InfoMessage,
- QueryMessage,
- CSVImportMessage,
- SetDatabaseMessage,
- BulkImportMessage.

If user connects to the server via Console Client and enters a query (e.g. select or create), QueryMessage is received and *ClientPoolWorker* calls *TCPClientHandler::HandleQuery* function which asynchronously calls *RunQuery* function.

RunQuery function

At the beginning of this function, there is a mutex that locks used database (we do not support multi-query parallelization on one database). After locking the mutex, a *GpuSqlCustomParser* instance is created and its *Parse* function is called. Return value of this function, that contains all result payloads in form of protobuf message, is returned from *RunQuery* function as well.

Database Persistence

Version 1.5.0

Main database file (.db)

Order of values saved on disk (json):

```
{
  „persistence_format_version“:
  • Petsistence format version (unsigned int32_t) ,
  „database_name“:
  • Database name (string) ,
  „database_default_block_size“:
  • Database default block size (unsigned int32_t) ,
  „tables“:
  [
    • For each table:
    {
      „table_name“:
      o Table name (string) ,
      „table_block_size“:
      o Table block size (unsigned int32_t) ,
      „start_up_loading“:
      o Start-up loading (bool) ,
      „save_interval_ms“:
      o Save interval in miliseconds (unsigned int32_t) ,
      „index_columns“:
      [
        o For each sorting column:
        {
          „index_column_name“:
          ▪ Index column name (string) ,
        }
      ]
      o For each column:
      „columns“:
      [
        {
          „column_name“:
          ▪ Column name (string) ,
          „column_type“:
          ▪ Type of column (unsigned int32_t) ,
          „file_path_address_file“:
```

```
            ▪ File path to .adrs file (string) ,
            „file_path_data_file“:
            ▪ File path to .data file (string) ,
            ▪ If column type is polygon or string:
            „file_path_string_address_file“:
            ▪ File path to .stradrs file (string) ,
            „file_path_string_data_file“:
            ▪ File path to .strdata file (string) ,
            „encoding“:
            ▪ Encoding of the string data (string) ,
            „default_entry_value“:
            ▪ Default entry value used if column is not nullable
            (int8_t, int32_t, int64_t, float, double, string) ,
            „nullable“:
            ▪ IsNullable flag (bool) ,
            „unique“:
            ▪ IsUnique (bool) ,
            „hidden“:
            ▪ IsHidden (bool) ,
          }
        ]
      }
  ]
}
```

Column address file (.adrs)

This ensures that blocks in .data file can be in random order. Value of entries of never persisted blocks into disk is UINT32_MAX for columnType: Polygon, String. Value of entries of never persisted blocks into disk is UINT64_MAX for columnType: Int8_t, Int32_t, Int64_t, Float, Double, Point. When we are saving a block of such index, that it will create an empty space between entries, then to fill this empty space, we will save there this value of never persisted blocks.

Order of values saved on disk (std::ios::binary):

```
Switch columnType:
  o ColumnType: Polygon, String
    ▪ Block index to which the fragment belongs to (unsigned int32_t)
      If the value is UINT32_MAX, that means, the fragment data are marked
      as invalid and may be deleted when the cleaning job is triggered
  o ColumnType: Int8_t, Int32_t, Int64_t, Float, Double, Point
    ▪ For each block:
```

```
        • Position (in bytes) of the start of the block data in .data file
        (unsigned int64_t)
```

Column data file (.data)

Order of values saved on disk (std::ios::binary):

```
Switch columnType:
  o ColumnType: Polygon, String
      ▪ Block index (unsigned int32_t)
      ▪ Group Id for binary indexing (int32_t)
      ▪ Length of NullBitMask (unsigned int32_t)
      ▪ NullBitMask (Array of chars)
      ▪ Number of actual entries (unsigned int64_t)
  o ColumnType: Point
      ▪ Block index (unsigned int32_t)
      ▪ Group Id for binary indexing (int32_t)
      ▪ Length of NullBitMask (unsigned int32_t)
      ▪ NullBitMask (Array of chars)
      ▪ Number of actual entries (unsigned int64_t)
      ▪ IsCompressed flag (bool)
      ▪ For i = 0; i < Block size; i++
      • Latitude (float)
      • Longitude (float)
  o ColumnType: Int8_t, Int32_t, Int64_t, Float, Double
      ▪ Block index (unsigned int32_t)
      ▪ Group Id for binary indexing (int32_t)
      ▪ Length of NullBitMask (unsigned int32_t)
      ▪ NullBitMask (Array of chars)
      ▪ Number of actual entries (unsigned int64_t)
      ▪ IsCompressed flag (bool)
      ▪ Minimal (smallest) value in this block - statistics
        (int8_t, int32_t, int64_t, float, double)
      ▪ Maximal (largest) value in this block - statistics
        (int8_t, int32_t, int64_t, float, double)
      ▪ Average value in this block - statistics (float)
      ▪ Sumarrization of values in this block - statistics
        (int8_t, int32_t, int64_t, float, double)
      ▪ Data (Block size × [int8_t, int32_t, int64_t, float, double])
```

Column fragment data file (.fragdata)

This is used for POLYGON and STRING data. Data are saved as 1MB fragments, whole fragments is for single block, does not have to be full. This file may consists of multiple 1MB fragments which are saved one by one. If the length is negative number, that means the string is never referenced and can be deleted. If we want to shorten or make longer the string data, we change the length of the current string to be negative number and we create a new string at the end of fragment or even create a new fragment at the end of .strdata file.

Order of values saved on disk (std::ios::binary):

```
For each entry:
  o String length with '\0' in bytes (int32_t)
  o String entry data (Array of chars in encoding UNICODE)
```

Version 1.4.3

Main database file (.db)

Order of values saved on disk (std::ios::binary):

```
• Petsistence format version (int32_t)
• Length of database name including symbol '\0' (int32_t)
• Database name (string)
• Database block size (int32_t)
• Number of table (int32_t)
• For each table:
  o Length of table name including symbol '\0' (int32_t)
  o Table name (string)
  o Table block size (int32_t)
  o Number of column (int32_t)
  o Number of sorting columns used when indexing (int32_t)
  o If number of sorting columns > 0
    ▪ Length of sorting column name including symbol '\0' (int32_t)
    ▪ Sorting column name (string)
  o For each column:
    ▪ Length of the column name including symbol '\0' (int32_t)
    ▪ Column name (string)
```

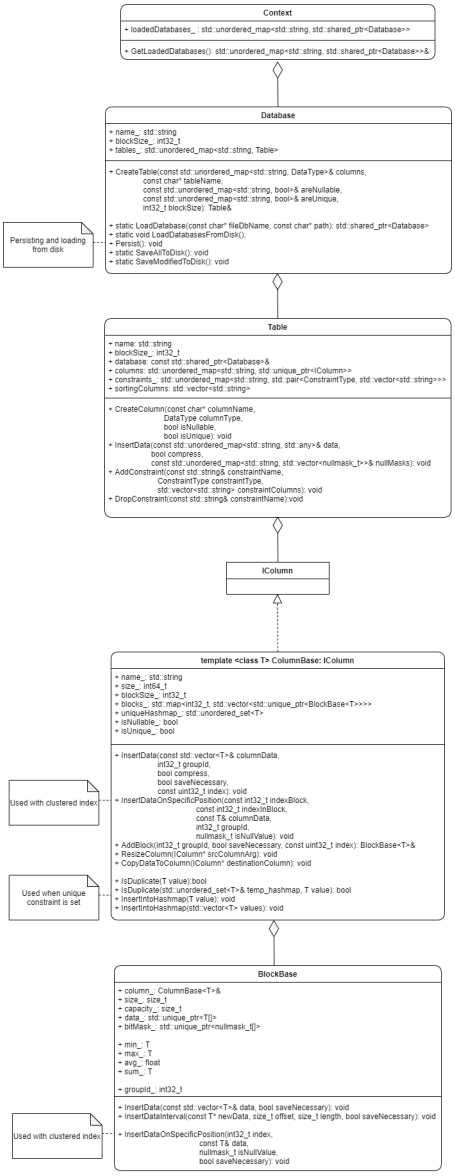
Column file (.col)

Order of values saved on disk (std::ios::binary):

```
• Type of column (int32_t)
• IsNullable flag (bool)
• IsUnique (bool)
• Switch columnType:
  o ColumnType: Polygon, Point, String
    ▪ If number of entries > 0:
      • Block index (int32_t)
      • Group Id for binary indexing (int32_t)
      • If IsNullable is true:
        o Length of NullBitMask (int32_t)
        o NullBitMask (Array of chars)
      • Block length in bytes (int64_t)
      • While currentBlockLength < BlockLength:
        o Entry length in bytes (int32_t)
        o Entry data (Array of chars)
  o ColumnType: Int8_t, Int32_t, Int64_t, Float, Double
    ▪ If number of entries > 0:
      • Block index (int32_t)
      • Group Id for binary indexing (int32_t)
      • If IsNullable is true:
        o Length of NullBitMask (int32_t)
        o NullBitMask (Array of chars)
      • Block length - number of entries (int32_t)
      • IsCompressed flag (int8_t)
      • Minimal (smallest) value in this block - statistics
        (int8_t, int32_t, int64_t, float, double)
      • Maximal (largest) value in this block - statistics
        (int8_t, int32_t, int64_t, float, double)
      • Average value in this block - statistics (float)
      • Sumarrization of values in this block - statistics
        (int8_t, int32_t, int64_t, float, double)
      • Data (Number of entries × [int8_t, int32_t, int64_t, float, double
```

Classes

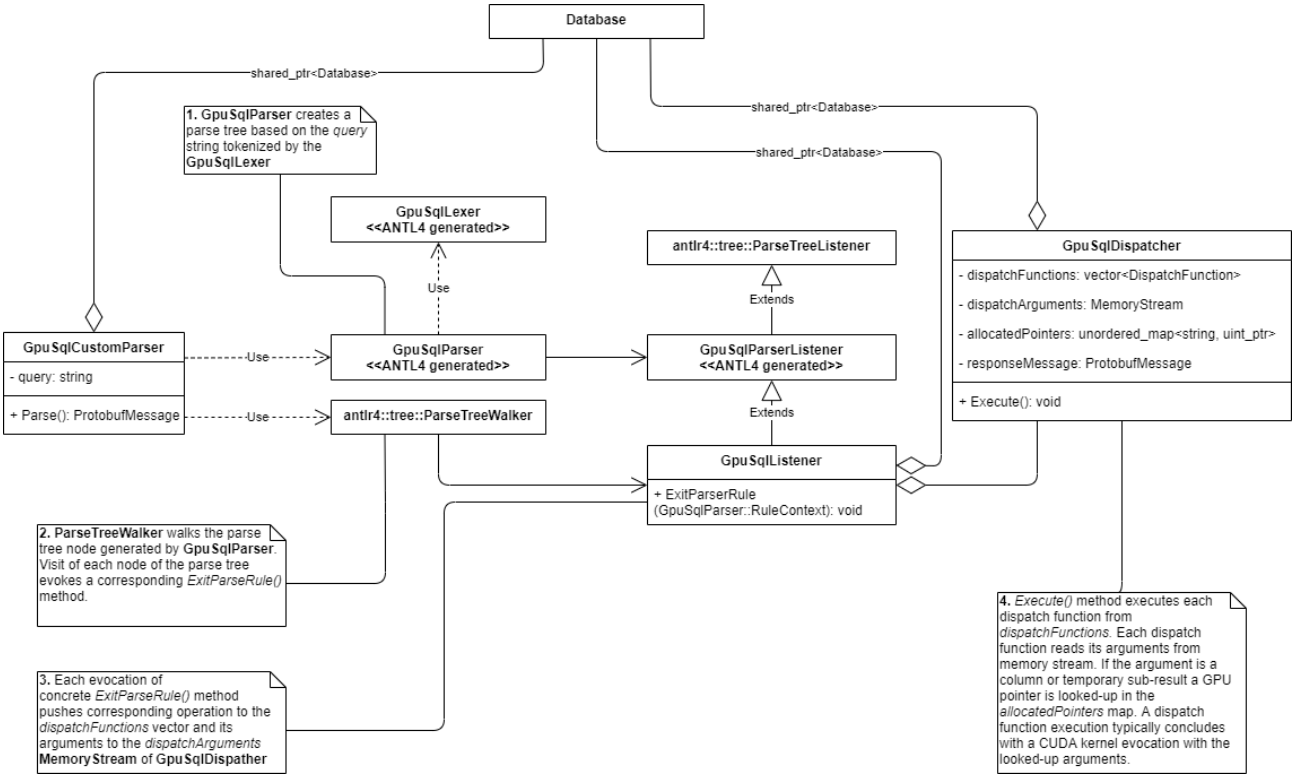
Classes representing whole database management system are shown in the class diagram.



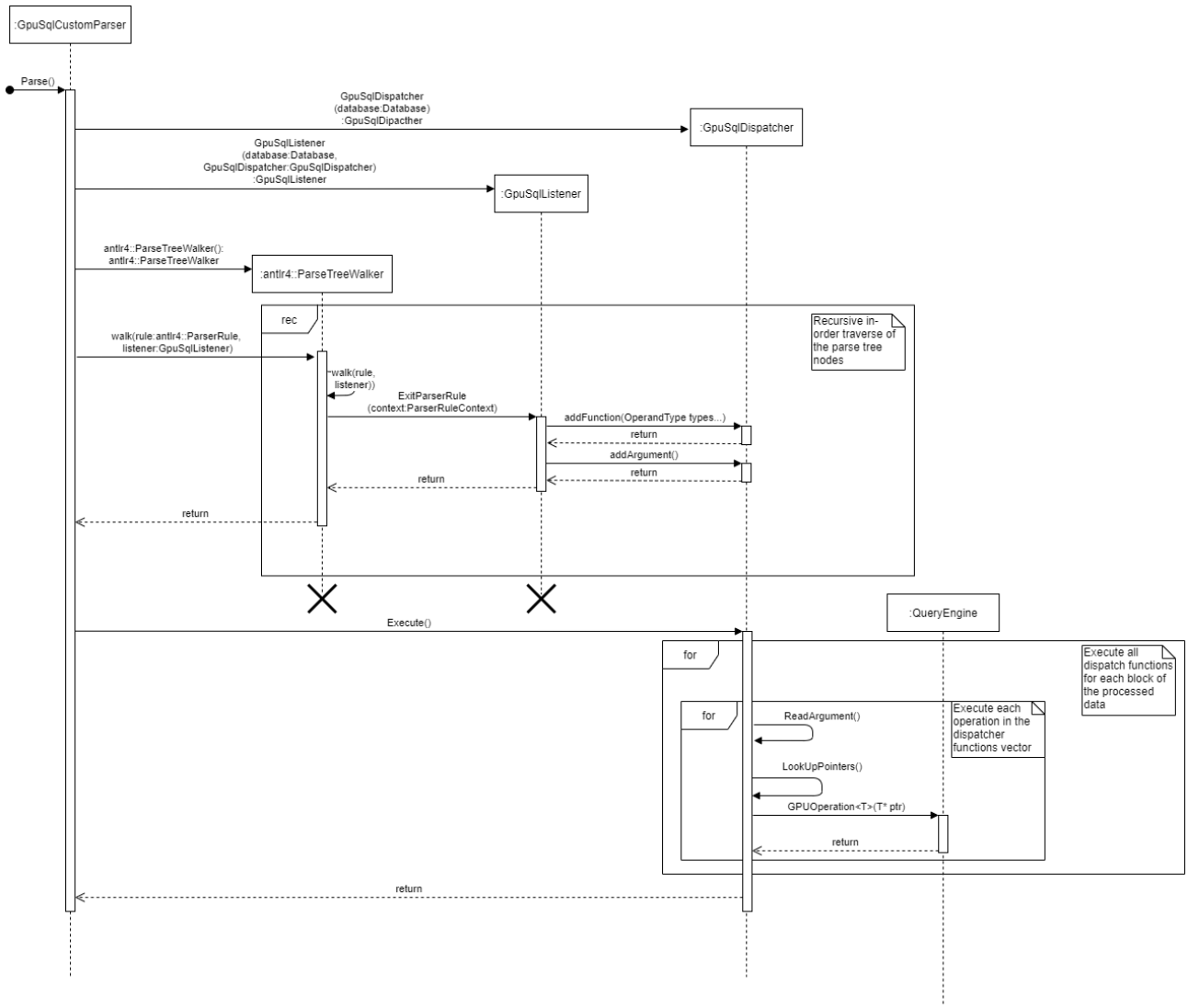
Class diagram with Database, Table, Column and Block hierarchy

Parser and Dispatcher

Based on a custom defined grammar file the ANTLR4 library generates a parse tree and provides means for its in-order traversal. The **GpuSqlListener** class implements listener methods evoked by entering/exiting the nodes of the parse tree during its traversal. This class also contains a typical parse stack. Based on the type of an operation in a particular parse tree node its corresponding enter/exit method also push a concrete dispatch function with correct template type parameters (given by the types of operands on the parse stack) to the *dispatchFunctions* vector of an **GpuSqlDispatcher** instance. The values of the operation operands are pushed to the *dispatchArguments* memory stream of the same instance. When the *execute()* method of this instance is called all dispatch functions from *dispatchFunctions* vector are executed. Each dispatch function reads its arguments from the *dispatchArguments* and uses them to evoke a specific (wrapped) **CUDA** kernel. All the dispatch functions are executed for each of the database storage blocks.



Class diagram of Dispatcher



Sequence diagram of dispatch process

GPU Core

GPU Memory Cache

Since user of the database often writes similar queries repeatedly, it would be appropriate to preserve all used columns in the GPU memory after they are loaded here within the first query processing. However, if the GPU memory is full and next query contains a column that is not loaded yet, some of the loaded columns needs to be deleted to free space for the new one. This functionality is handled by GPU Memory Cache that uses LRU (Least Recently Used) policies. The cache is also related to GPU Memory Allocator.

CacheEntry

CacheEntry is a struct representing one cached **block of one column**. It is defined by a string key consisting of database, table and column name, block index and also load size (number of rows - e.g. when LIMIT is used) and load offset (e.g. when OFFSET is used). With this key, a pointer to GPU, size of allocated memory and LRU queue iterator are associated.

```
std::string key = databaseName + "." + tableAndColumnName + "_"
  + std::to_string(blockIndex) + "_" + std::tostring(loadSize) + "_"
  + std::to_string(loadOffset);
```

Fetching blocks

As "user" of the cache, we usually use just *GetColumn* function that uses template T for defining data type of values in the column. In arguments, we define database, table and column name, block index, size, load size (could be INT64_MAX) and load offset (could be 0). The cache will allocate GPU memory and copy data of wanted block to the GPU memory and **return pointer** to it. If it is already cached, it will not allocate nor copy anything, it will just return the pointer. Within the allocation of new GPU memory block, some blocks might be evicted (if the cache is full).

Eviction

Since we have finite GPU memory, after caching too many blocks, we need to evict the oldest ones which means to delete them and free GPU memory ("un-cache" them). This is performed in a following while cycle:

```
while (!HasFreeSpace(sizeToInsert) || !GetAllocator().CanAllocate(sizeToIn
{
    if (!Evict())
    {
        throw std::length_error("Not enough space left in cache");
    }
}
```

So we evict as many cached block as needed to make enough free space for allocation of the new block. However, if we evict all block and the free space is still not enough, the Evict function returns false and that means there is not enough space left in cache, just as written in the error message.

Cache Size

Within the Configuration, by editing entry *"GPUCachePercent",* the user can set the percentage, how much of the GPU memory will be taken by the cache (however, this value could be automated sometime in the future).

GPU Memory Allocator

Since original CUDA allocator causes certain overhead during allocation a block of GPU memory, we implemented our own simple allocator (*CudaMemAllocator* class) which is approximately 3x faster. It lacks a defragmentation feature, but the fragmentation does not cause any crashes.

BlockInfo

The allocator stores a list of GPU memory blocks that are implemented as *BlockInfo* struct. This struct has a flag if it is allocated (or free block), size of the block, pointer to the block in the GPU memory, and iterator with blocks sorted by its size.

Allocation

At start, there is one free block with size of whole GPU memory. During the allocation, this block is split into two parts - one is allocated (with required size) and the other one (with remaining size) is free. If there is more free block (due to external fragmentation), the smallest suitable block is split and again, the first part is allocated.

Deallocation

During the memory freeing, or deallocation of one block, the block is marked as free and joined with neighboring free blocks.

Group By

Classes

In the implementation, there is a IGroupBy interface which is implemented by:

- generic class *GPUGroupBy<AGG, O, K, V>* for key defined by single numeric column (e.g. int or float) - for queries like "... GROUP BY columnInteger"
- specialization *GPUGroupBy<AGG, O,* **std::string***, V>* for String columns - for queries like "... GROUP BY columnString"
- specialization *GPUGroupBy<AGG, O,* **void***, V>* for multiple columns - for queries like "... GROUP BY columnString, columnInteger"

AGG in templates represents aggregation function (e.g. SUM - *AggregationFunctions::sum*), *O* stands for output data type for aggregated values, *K* for data type of key (if generic class is used) and *V* for input data type for aggregated values. If the COUNT aggregation function is used, *int64_t* must be used as *O*, independent of *V*.

Pipeline

Since our database engine processes the query per blocks in a cycle, *GPUGroupBy* class has function *ProcessBlock*, which is called for each block. After last block is processed, *GetResults* function is called and it returns the final results. In multiple GPU setting, there is one *GPUGroupBy* instance per one Graphics card (as well as *GpuSqlDispatcher* instances). Function *ProcessBlock* of these instances is called in parallel, and the *GetResults* function, after all blocks are processed, is called just on the first of the instances and it merges the results from all Graphics cards.

In case of String column and multiple columns (so-called Group By Multi Key), the *ProcessBlock* function consists of two steps (due to CUDA atomic functions reasons). First part is calling a group by kernel, that fill array with indices to input keys, and the second part is storing keys to group by buffers (implemeted as a hash table) according to the indices.

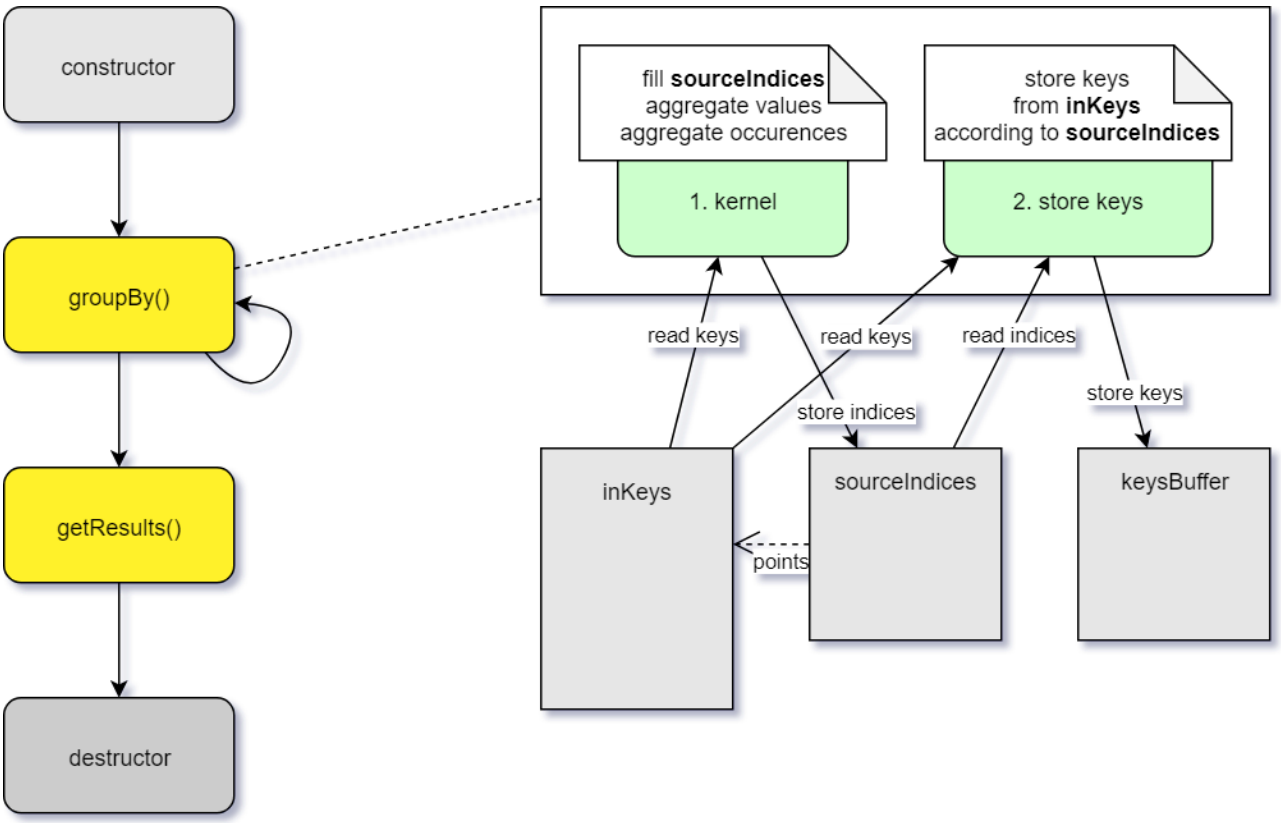


Diagram explaining how Group By works with multiple columns

Kernel

Group By kernels (*group_by_kernel, kernel_group_by_string* and *kernel_group_by_multi_key*) are based on CUDA atomic functions, since multiple threads collect keys and aggregate values. If one thread finds a new key, it tries to insert that into the hash table, using *atomicCAS* function. Value returned from this functions tells if the insertion was successful or other thread inserts its key. In the second case, it checks if the inserted key is the same (ok, done) or different (in this case a hash conflict occurred, so new index to the hash table is computed and the insertion is tried again).

After successful insertion of the key, the value is aggregated in the same index, but different "column" (column with values) of the hash table. In case of average aggregation function (AVG), both values and count of keys are summed. And in the *GetResults* function, the values are divided by the counts (so-called occurrences) to get the average value.

Order By

The basic principle of the Order By module is to generate indices according to columns in ORDER BY clause and reorder result rows (with columns in SELECT clause) according to these indices.

Generating indices

Function *OrderByColumn* of class *GPUOrderBy* is used to generate indices according to order of values in column from ORDER BY clause. Within null values, for each NULL, the lowest possible numeric value is generated to ensure the first place for NULLs. For generating the indices itself, radix sort from CUB library is used. To enable ordering by multiple columns, there are two buffers (*indices1* and *indices2*) where the first one is initialized with ascending numbers (0,1,2,...) and the second one is computed in each *OrderByColumn* function (according to currently processing column from ORDER BY clause) and they are subsequently swapped.

Supported data types for order by columns are just numeric types (int, long, float and double), string is not supported yet. Point and polygon are not even suitable for ordering. However, all data types are supported for columns in SELECT or WHERE clause of query with ORDER BY. These columns need to be reordered according to the indices.

Reordering columns

Reordering columns from SELECT and other clauses are then trivial. Just simple kernel is called (*kernel_reorder_by_idx* that reorders the input column). Little bit more complex are kernels for reordering string and polygon columns, since they are complex data types.

Join

For joins execution, a Join Dispatcher firstly computes join indices, and then, the main Dispatcher reorders all the loaded columns according to them. They are stored in a map where the key contains table name and the value is an array with the indices itself.

If at least one column from the ON clause is set as UNIQUE, the Merge Join implementation is used. But if none of the two columns is UNIQUE, it means they may contain duplicates and therefore, Hash Join needs to be used. However, for now, neither Hash Join can process duplicates in both columns correctly.

Merge Join

Function *MergeJoin::JoinUnique* is used for generating the join indices. Input is the two columns, which must have the same data type T. At least one of the columns have to be unique. If these conditions are met, following steps are performed:

1. sorting input columns using radix sort
2. labeling
3. partitioning
4. finding a merge path (generating a merge indices)
5. evaluating a join predicate on the merge path
6. compressing the join indices (from the merge indices)

The most important part of the Merge Join, finding the merge path - *kernel_find_merge_path*, is based on parallel finding a path on a 2D table from top left to bottom right corner. Rows of this table represent left column of ON clause and columns right column. However, this table is not present in the GPU memory, it is represented by just two 1D arrays: *mergeAIndices* and *mergeBIndices*.

Hash Join

Function *GPUJoin::JoinTableRonS* also computes the join indices, but using Hash Join algorithm. This method calls left table R and right S, respectively, and it consists of following steps:

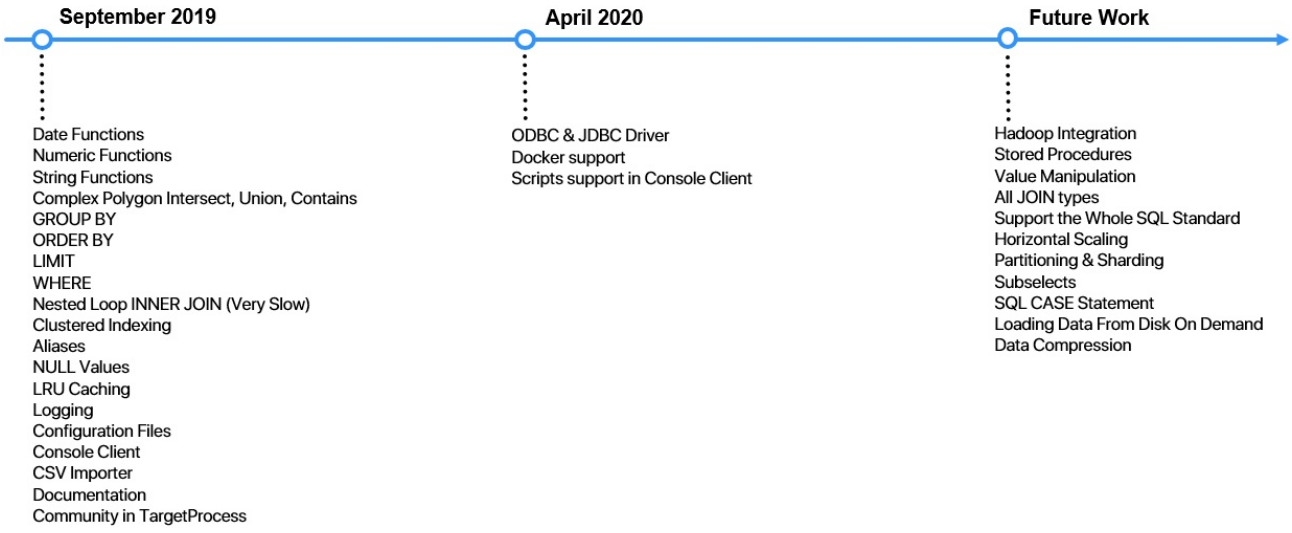
1. hashing R block
2. counting matches between R and S blocks
3. storing the join indices

Reordering columns

The main Dispatcher uses function *CPUJoinReorderer::reorderByJI* with computed *joinIndices_* to reorder columns from different tables. For example, see function *GpuSqlDispatcher::LoadCol*. Note that these reordered values are also cached, like the original columns.

Roadmap

Roadmap: QikkDB (Database Server)



Roadmap: TellStory (Visualization Tool)

Terms of Use

1 Introductory Provisions

1.1 These license terms and rules of use of the qikk.ly software (the “Terms of Use”) provided by Instarea, s.r.o., with its registered office at 29. augusta 36/A, 811 09 Bratislava, Slovak Republic, Identification No.: 43 866 239, registered in the Commercial Register of the District Court Bratislava I, section: Sro, insert No.: 49904/B (“Instarea”), provide for the mutual rights and obligations associated with the provision of a work in copyright and/or other intellectual property, computing services and database services upon the User’s accession to these Terms of Use, which at the same time shall constitute the Agreement concluded between these Parties.

1.2 These Terms of Use shall solely provide for the regulation of use of the qikk.ly software during the implementation of the temporally limited testing phase. Once the testing phase has ended, the rights of Users under these Terms of Use shall expire.

1.3 Any legal relationships not provided for by these Terms of Use shall be governed by the applicable laws of the Slovak Republic, unless stipulated otherwise.

---

2 Interpretation

2.1 Data means, for example, information, records, images, data, communications or any other information provided or stored within the Software.

2.2 Software means, in particular, the computer program named qikk.ly that represents the data concerned, their structured datasets and the individual software computing functions designed to process that data, including individual, even separate, modules or components, particularly referred to here: https://docs.qikk.ly/.

2.3 Account means the user interface of the Website. At the same time, the Account allows access to certain functions associated with the Software, facilitating communication with the Provider and sending comments on the course of the testing phase.

2.4 Provider means Instarea. The Provider is the owner and/or sole holder of all rights to the work in copyright or other intellectual property rights, in particular to the Data and the Software.

2.5 User means the natural person or a person acting on behalf of the legal person that is a Party and uses the Software.

2.6 Website means the websites https://qikk.ly/ and https://tellstory.cloud/.

2.7 Agreement means the agreement in electronic form, which takes the form of accession to these Terms of Use, and any related legal documents, information and instructions for use of the Software, as amended at the time of their use.

2.8 Party means the Provider and the User who have concluded the Agreement; a Party also means the legal person whose agent has concluded the Agreement under these Terms of Use if it is obvious that it is using the Software for activities related to the business activities of that legal person, but in accordance with these Terms of Use.

---

3 Subject-Matter of the Agreement

3.1 The subject-matter of the Agreement shall be the provision of the Software in the form of obtaining access to and use of its computing functions and data. The subject-matter of the Agreement shall also include the rights and obligations of both Parties and other relevant facts stipulated by the Agreement.

3.2 The Agreement shall be concluded at the moment of approval of the User's request to create the Account and gaining access to the Software upon the Provider's approval. The request pursuant to the previous sentence shall take the form of registration of the User's e-mail address by means of the function dedicated therefor on the Website.

3.3 The Agreement shall be entered into for an indefinite period of time. By concluding the Agreement, the User agrees that the moment of its termination will be unilaterally determined by the Provider. The termination of the testing phase by the Provider gives rise to legal effects within the meaning of clause 3.6. of these Terms of Use.

3.4 The Provider shall make the Software available during the testing phase free of charge. At the same time, the User declares that by providing any data, information, communication, feedback or reviews of the Software, the User will not claim any financial or any other form consideration against the Provider or any legal successor of the Provider or a third party.

3.5 The User shall be entitled to use the Software solely to the extent set forth in these Terms of Use or the Provider‘s instructions, and only for the purposes of its testing, without any commercial use. The Software may be used during the testing phase solely for the purpose of learning about the functions of the Software and providing feedback and opinions on the operation of the Software to the Provider. The User shall not be entitled to use the Software for commercial purposes and for its own or third party’s business purposes. The User shall not be entitled to use the Software in a way that would result in the creation of any financial, proprietary, intellectual or property assets.

3.6 The Provider may interrupt or terminate the testing phase of the Software at any time, even without giving any reason therefor. Upon termination of the testing phase, the User shall cease to be entitled to any use or retention of the Software or any part thereof. The use of the Software during the testing phase shall not afford the User any legal entitlements for its continuous use, and the Provider shall be not liable for any loss of User’s data or for any obligations assumed by the User against third parties that the User can no longer honour due to the termination of access to the use of the Software by the Provider.

3.7 Within the testing phase, the Software is provided in electronic form by making available its functions and database, which the User acquires by downloading then to the User’s terminal. The Software is or will be available to the User on the Website for the purpose of its downloading.

3.8 Upon the registration of the User’s e-mail pursuant to clause 3.2 and upon successful approval of the User’s request for access to the Software, the Agreement shall be entered into and the Account shall be created.

3.9 If the Software is used in breach of these Terms of Use, including any form of its misuse or breach by the User of its contractual obligations, the Provider shall become entitled to terminate the Agreement; this shall be without prejudice to the Provider’s right to damages and other claims under these Terms of Use.

3.10 The User shall properly protect the login data for the Account from being disclosed and provided to a person other than a Party.

3.11 For the purpose of checking the fulfilment of the Terms of Use and protection of rights to the Software, the Provider reserves the right to audit the User. The User shall tolerate the performance of the audit, even without giving a reason by the Provider, immediately after receiving a written request from the Provider. The written notice shall be delivered by e-mail sent to the User's address.

3.12 In connection with the provision of the Software, the User is fully aware that the Provider will be able to use, without any limitation, the information concerning the manner in which the User uses the Software as well as the information provided by the User to the Provider in connection with the use of the Software and information about the activity of the Software for testing purposes.

---

4 Termination of Agreement

4.1 The Contract may be terminated by the Provider's withdrawal due to a breach of the Terms of Use by the User or by the expiry of its term, which the Provider may unilaterally set at any time during the term of the Contract in accordance with clause 3.3.

4.2 If a withdrawal takes place within the meaning of the preceding clause, by acting in breach of these Terms of Use, the User waives any financial compensation, in kind or in any other form, for the consequences associated with the termination of the provision of the Software.

4.3 The User may terminate the Agreement by withdrawing from it with immediate effect, at any time during its term and for convenience.

4.4 In the case of the procedure under clause 4.1 or clause 4.3, the User shall have the right to obtain data stored by the User in the Software for 7 calendar days after the termination of the Agreement. Any data and information produced by the User's activities, including reviews and feedback on the use of the Software, may be retained and used by the Provider for the purposes specified by the Provider.

---

5 Copyright and Other Intellectual Property Rights

5.1 The User acknowledges that the Website and the Software, including, without limitation, databases, Data, methods of calculation and acquisition of Data and source codes, functions, products, images, user interface, text content, logo, designs, manuals and other documentation materials (the “Intellectual Property”) shall be protected by applicable intellectual property rights laws, including, without limitation, copyright laws, owned or held exclusively by Instarea. The User undertakes to use such Intellectual Property objects solely for the purpose of using the Software in accordance with these Terms of Use and to the extent of the provided Software.

5.2 The User undertakes to not copy, reproduce, modify, lend, rent, borrow, republish, sell, distribute, download the Intellectual Property objects or create any derivative works from them in any way and shall not use the Website and the Software or any parts thereof subject to intellectual property protection in any unauthorised manner, including, without limitation, an unauthorised access or congestion of the network capacity of the Website. Any use of the protected information and materials, except for the use of the Website and the Software in accordance with these Terms of Use, shall require the prior written consent of Instarea.

5.3 By downloading the Software, the Provider grants the User a non-exclusive territorially unlimited license, limited in time to the term of the Agreement under these Terms of Use, in accordance with these Terms of Use, in particular in accordance with clause 3 and clause 5 (the “License”). The User may not provide a sublicense to any third party or assign the rights under this License to any third party.

5.4 The User may not damage the operation and structure of the Website and the software by circumventing or breaching the security measures, uploading files containing viruses or harmful programs, entering user accounts of other Users or performing other unfair or unlawful acts violating the laws or infringing on the rights of the Provider.

5.5 The Provider shall not be liable for the content included in the Software, the information and the Data it provides within the Software.

5.6 The User may not use the Software for purposes related to the so-called high-risk activities, such as the use of the Software for medical activities, medical research and records, nuclear-related activities, provision of navigation services, military activities, etc.

6 Warranty Claim

6.1 Because the use of the Software during the testing phase is free of charge, the Provider shall not incur any liability to the User arising from liability for defects and a warranty claim made.

6.2 However, you may address your suggestions regarding the provision of the Software to our support centre by sending an e-mail to support@qikk.ly or by contacting us through the helpdesk at support.qikk.ly.

---

7 No Liability

7.1 We strive to ensure continuous provision of our services and their uninterrupted availability on the Website and the functionality of the Software. We use the highest standards of security and serviceability to make our services available whenever you need them. However, we shall not be liable for the failures that we cannot influence in cases of objective and unavoidable technical obstacles or other facts that make it impossible for us to perform. However, we use our best efforts to promptly resolve any existing obstacles.

7.2 We also reserve the right to be not liable for any damage or harm suffered as a result of the use of the Software or any interruption or termination of the use of the Software.

7.3 The Provider reserves the right to terminate its business, the provision of the Software and operation of the Website or its part at any time and to not enter into new Agreements with additional Users.

7.4 The User shall bear full responsibility for the data and information stored by it in the Software, including the fact that it has the data and information legally in its possession.

7.5 The Provider shall not be liable for violating the integrity of the User's data, leaking or disclosing it to unauthorised persons, in case of a breach of security measures in the User's terminals or as a result of the User acting in a way that could lead to a breach of data protection, including the omission of customary precautions by the User.

7.6 Any information associated with the use of the Software shall be subject to trade secrecy protection. The User undertakes to maintain confidentiality in relation to this information and to protect it against publication or disclosure to a third party.

7.7 The Provider reserves the right to modify the data stored by the User in the Software during the performance of tasks associated with the service of the provision of the Software. The Provider shall not be liable for any damage or injury caused by this action.

7.8 The Provider reserves the right to provide the data stored by the User in the Software to a third party, in particular a public authority acting in accordance with the law for the purpose of exercising its powers.

7.9 The Provider is not obliged to ensure the storage of data stored by the User in the Software.

7.10 The Provider shall in no way be responsible for compliance with the so-called lawfulness of processing of data within the meaning of Article 6 of Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation), provided by the User, which may be in the nature of personal data, as well as for the fulfilment of any legal obligations imposed on the controller of personal data by the legislation governing the protection of personal data. For the purposes of the Agreement, the use of the Software in a manner that would lead to the processing of personal data, including the processing of personal data in which the Provider acts as a so-called processor of personal data within the meaning of Article 28 of the GDPR, shall be prohibited. Upon its accession to the Agreement, the User declares that it will bear all possible sanctions that may be imposed on the Provider as a result of a breach of this provision.

---

8 Sanctions for Breaches of Terms of Use

8.1 The Provider shall be entitled to a contractual penalty of EUR 500 000 (in words: five hundred thousand euro) if the User breaches these Terms of Use.

8.2 If the User breaches these Terms of Use, the Provider shall also be entitled to compensation of damage (including lost profits) directly or indirectly incurred by the Provider in connection with their breach, and the Provider shall be entitled to full compensation, even in excess of the contractual penalty.

---

9 Communication, Notices and Delivery

9.1 The Parties agree that in case of delivery of mutual correspondence, delivery under these Terms of Use means delivery of a written content to the e-mail address support@qikk.ly or by registered mail or courier to: Instarea, s.r.o., with its registered office at 29. augusta 36/A, 811 09 Bratislava, Slovak Republic. The date of delivery also means the date on which a Party refuses to accept the document being delivered or the day of return of the parcel containing the words “the addressee has not taken over the delivery within the take-over deadline”, “the addressee has moved”, “the addressee is unknown” or any other note of similar meaning. In case of electronic delivery of documents by e-mail, a document shall be deemed to have been delivered on the day following the day on which it was dispatched. For the purposes of delivery by post, the addresses of the Parties known to the Parties or, in case of a Party – entrepreneur, the registered office address registered in the commercial, trade or special register shall be used.

---

10 Resolution of Disputes and Jurisdiction of Courts

10.1 The Parties agree that any disputes arising in connection with the fulfilment of obligations under the Agreement will be preferably resolved out of court by negotiation or by mutual agreement. Should the Parties fail to resolve their disputes out of court, the Parties may resolve such disputes through the courts in accordance with the applicable laws of the Slovak Republic. The local jurisdiction of the court is given in accordance with Act No. 160/2015 Coll., the Code of Civil Contentious Procedure, as amended.

---

11 Change of Terms of Use

11.1 The Provider reserves the right to change these Terms of Use at any time during the term of the Agreement or the use of the Software. The Provider will announce the change to the Terms of Use through the Website or a notice via electronic communication. The Parties shall always be bound by the provisions of the Terms of Use valid at the time of validity of the Agreement.

11.2 If any provision of the Terms of Use becomes invalid, ineffective or unenforceable to a specified extent, the remaining provisions unaffected by this shall remain fully valid. In such a case, the Provider will replace the provision with a valid, effective and enforceable provision that will differ to the smallest possible degree from the principles agreed in these Terms of Use, while preserving the economic and legal purpose and meaning of the provision being replaced.

Place: Bratislava, Slovak Republic
Date: 23 October 2019

Release Notes

qikkDB

The released versions of qikkDB (database server) are listed below, along with their main changes.

1.5.0

Release Date: the 30th of June 2020

Features & Performance:

- Completely refactored database persistence format
  - Main .db file has been structured from binary form to JSON
  - Persisting only modified blocks of data (in previous versions, whole modified columns we persisted)
  - Persisted blocks of data has different structure and fixed length
- Added function DAYOFWEEK
- Added function WEEKDAY

1.4.3

Release Date: the 15th of April 2020

Features & Performance:

- Handling of corrupted database files
- Improved main configuration file - added options to:
  - Choose if to use multiple graphic cards (if available) or just a single graphic card
  - Choose the maximum overal GPU VRAM usage in percent
- Prepared Docker files
- Added String to Bool casting option
- Improved log messages
- Improved overal stability

1.4.2

Release Date: the 6th of February 2020

Features & Performance:

- Added command SHOW QUERY COLUMN TYPES
- Added option to run SQL scripts from Console
- Added conversion from String to Bool in ALTER TABLE ALTER COLUMN
- Added option to ADD CONSTRAINT and DROP CONSTRAINT
- Added constraint type UNIQUE
- Added option to SHOW CONSTRAINT
- Added checking if the database core has write access into directory where the databases are

1.4.1

Release Date: the 23rd of October 2019

Features & Performance:

- GROUP BY support for unlimited number of unique values in results
- Block size can be set also per tables
- Decreased memory (RAM) usage of loaded databases by 30%
- Decreased memory (RAM) usage when loading databases from disk by 40%
- Added function DATE() which returns datetime in human readable format
- Improved speed of queries like SELECT * FROM table LIMIT 10;
- Added functionality to use GROUP BY without aggregation function
- Added possibility to rename database / table / column
- Added possibility to drop database / table / column

1.4.0

Release Date: the 25th of September 2019

Features & Performance:

- Date Functions
- Numeric Functions
- String Functions
- Geospatial Functions
- GROUP BY
- ORDER BY
- LIMIT
- WHERE
- Nested Loop Inner JOIN (Experimental)
- Clustered Indexing
- Aliases
- NULL Values
- LRU Caching
- Logging
- Configuration File
- Console Client

FAQ

Do I have to fit all the data into memory (RAM) to work with qikkDB?

Yes. We do not support swaping data into disk yet, but we are working on it.

Does qikkDB support modification of data?

Currently we cannot modify (update) a single entry data (e.g. we have a column with book prices, we cannot change a price per book). However, you can rename database / table / column or you can change a column data type which will result in casting the data and therefore possible modification of data.

What happens to the data changes which have not been yet saved into disk if database crashes or a server goes down?

We are saving only modified columns during autosave. How often an autosave is triggered can be changed in configuration file. Currently, any changes which were not persisted into disk will be lost if a database crashes or server goes down. We will look at this when we will support data updates as well.

Contact Us

Submit Issues, Ideas & Questions

Check out the **list** of already submitted issues, ideas and questions and upvote the ones that you need the most. You have to be logged in to upvote (if you do not have an account, you can create one).

If you could not found in the **list** the issue, idea or a question, that you have in mind, **submit a new issue, idea or a question**.

How to submit an issue

1. Visit Instarea Help Desk.
2. Choose request type 'Issue' (default is 'Idea').
3. Write a brief description of the issue.
4. **Provide more details** in the bigger text box below brief description **so we can replicate the issue**. Provide also **error message which appeared on client and/or server side**.
5. **Attach server side logs** - it does not have to be the whole log file, but it has to be **at least those lines which are related to the issue**. We really need them to see what went wrong on the server side.

Email

Instarea:
hello@instarea.com

qikkDB team:
support@qikk.ly

Phone

Instarea:
+421 903 124 356

qikkDB team:
+421 902 946 341

Address

29.Augusta 36/A
811 09 Bratislava
Slovak Republic