Генератор высокопроизводительной NoSQL базы данных

Выпускная квалификационная работа

Владимир Барашко
Руководитель: Коновалов А. В.

Московский государственный технический университет им. Н.Э.Баумана

29 июня 2017 г.

Постановка задачи

- разработать высокоэффективную базу данных, позволяющую записывать и считывать данные (пренебречь поддержкой удаления и модификации данных);
- разработать генератор базы данных по заранее известной схеме данных.

Описание

`sample.sql` $\rightarrow$ `sample.h sample.c`

- вход — спецификация на собственном диалекте SQL, описывающая набор таблиц и процедуры чтения и записи;
- программа генерирует исходный код `.c` и заголовочный файл `.h` библиотеки;
- в сгенерированной библиотеке — методы и структуры для записи, чтения и хранения данных;
- эффективность+удобство сопровождения – база данных поддерживает только чтение, но не модификацию, что позволяет отказаться от интерпретации запросов и заложить их структуру в библиотеку, отсутствие средств обновления;

Спецификация базы данных

```
CREATE TABLE Person (id int pk, name char(32));
CREATE PROCEDURE add_person
        (@id int in, @name char(32) in)
BEGIN
        INSERT TABLE Person VALUES (@id, @name);
END;
CREATE PROCEDURE get_person
        (@id int in, @name char(32) out)
BEGIN
        SELECT name SET @name FROM Person WHERE id = @id;
END;
```

- реализация на flex+bison;
- поддерживаемые типы данных: `int`, `float`, `double`, `char(int)`.

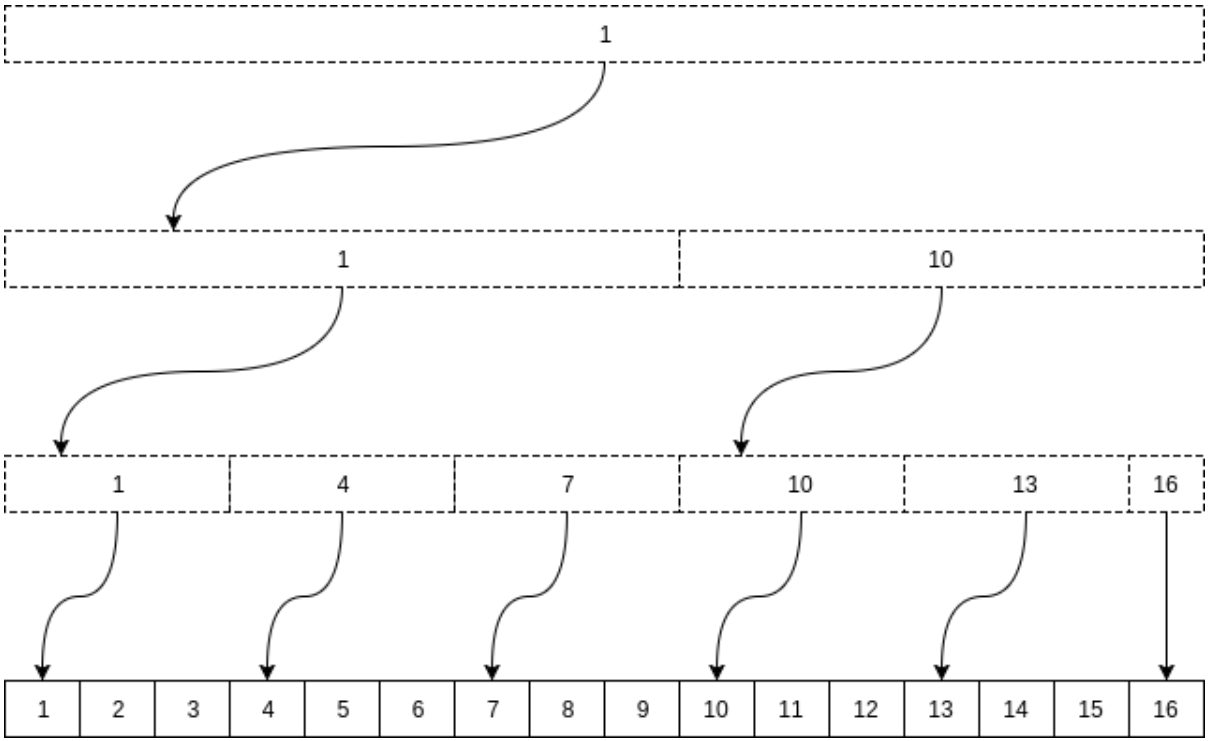
Заголовок файла

```
void add_person(int32_t id, char* name);

struct get_person_out_data { char name[33]; };
struct get_person_out {
        struct get_person_out_service service;
        struct get_person_out_data data;
};
void get_person_init(struct get_person_out* iter,
        struct demo_handle* handle, int32_t id);
int get_person_next(struct get_person_out* iter);

void demo_open_write(const char* filename);
void demo_close_write(void);
struct demo_handle* demo_open_read(const char* filename);
void demo_close_read(struct demo_handle* handle);
```

Пример использования

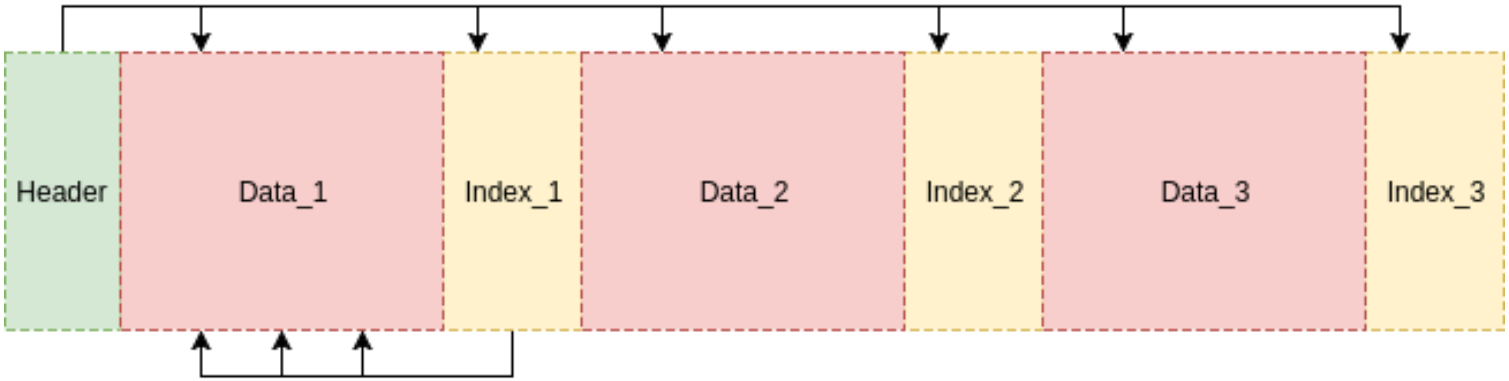
```
#include "demo.h"
int main() {
        demo_open_write("FILE");
        add_person(0, "Sidney Crosby");
        demo_close_write();

        struct demo_handle* handle = demo_open_read("FILE");
        struct get_person_out iter;
        get_person_init(&iter, handle, 0);
        while (get_person_next(&iter))
                printf("%s\n", iter.data.name);
        demo_close_read();
}
```

Индекс (B-tree)



- можно не хранить все данные в оперативной памяти, минимизация количества обращений к диску;
- при вставке данные пишутся в буфер;
- при закрытии базы на запись формируется индексное дерево, записывается файл.

Хранение данных



- в заголовке файла — количество таблиц, количество записей в них, смещения в файле каждого раздела;
- разделы индекса — узлы индексного дерева, узел включает информацию о первичном ключе и смещении в разделе данных;
- раздел данных — собственно данные;

Тестирование

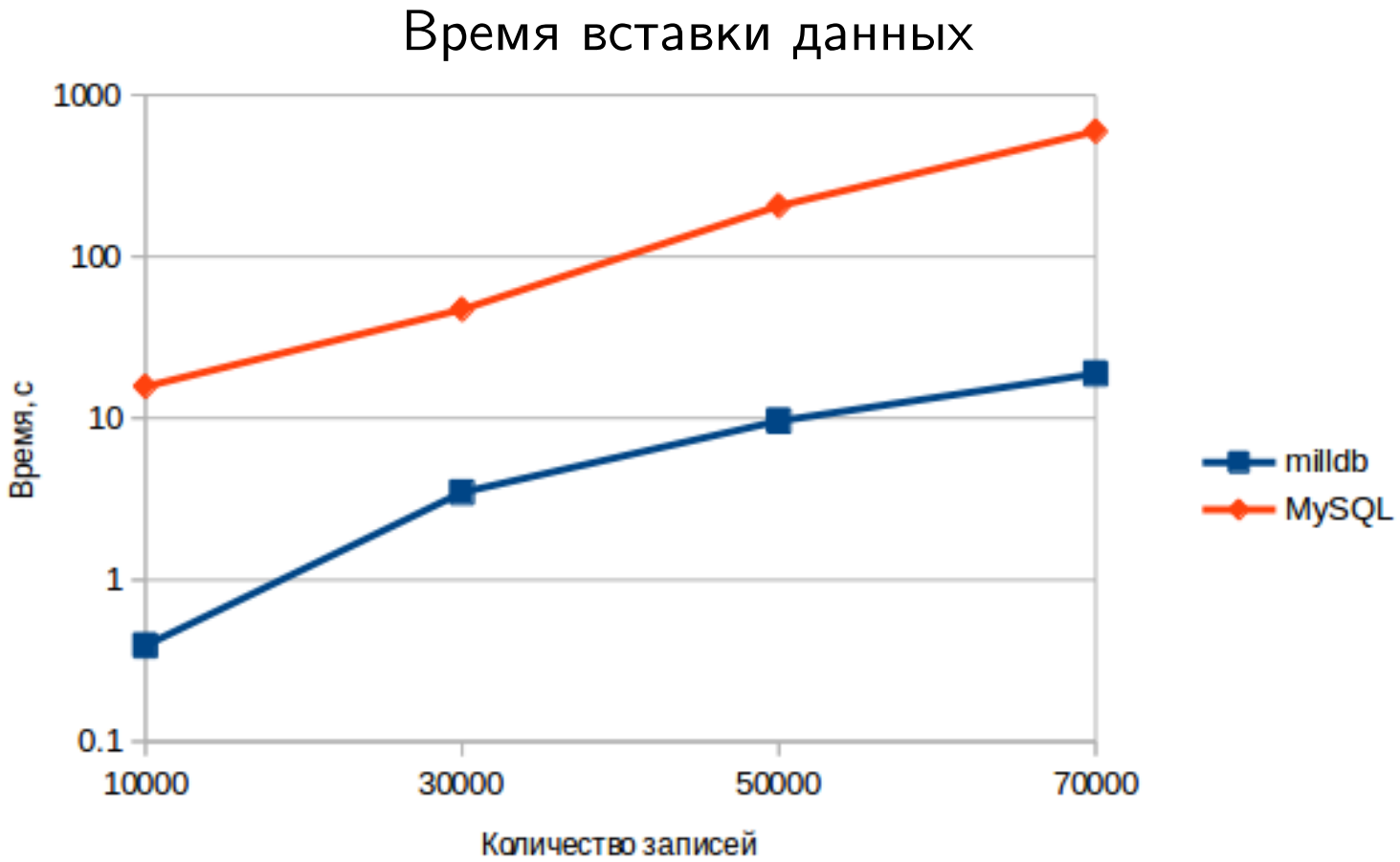
- сравнение с аналогичной разработкой (без генератора);
- параметры: таблица с 8 полями типа `int`, $\sim$9 млн записей.

Таблица: Сравнение с аналогичной разработкой

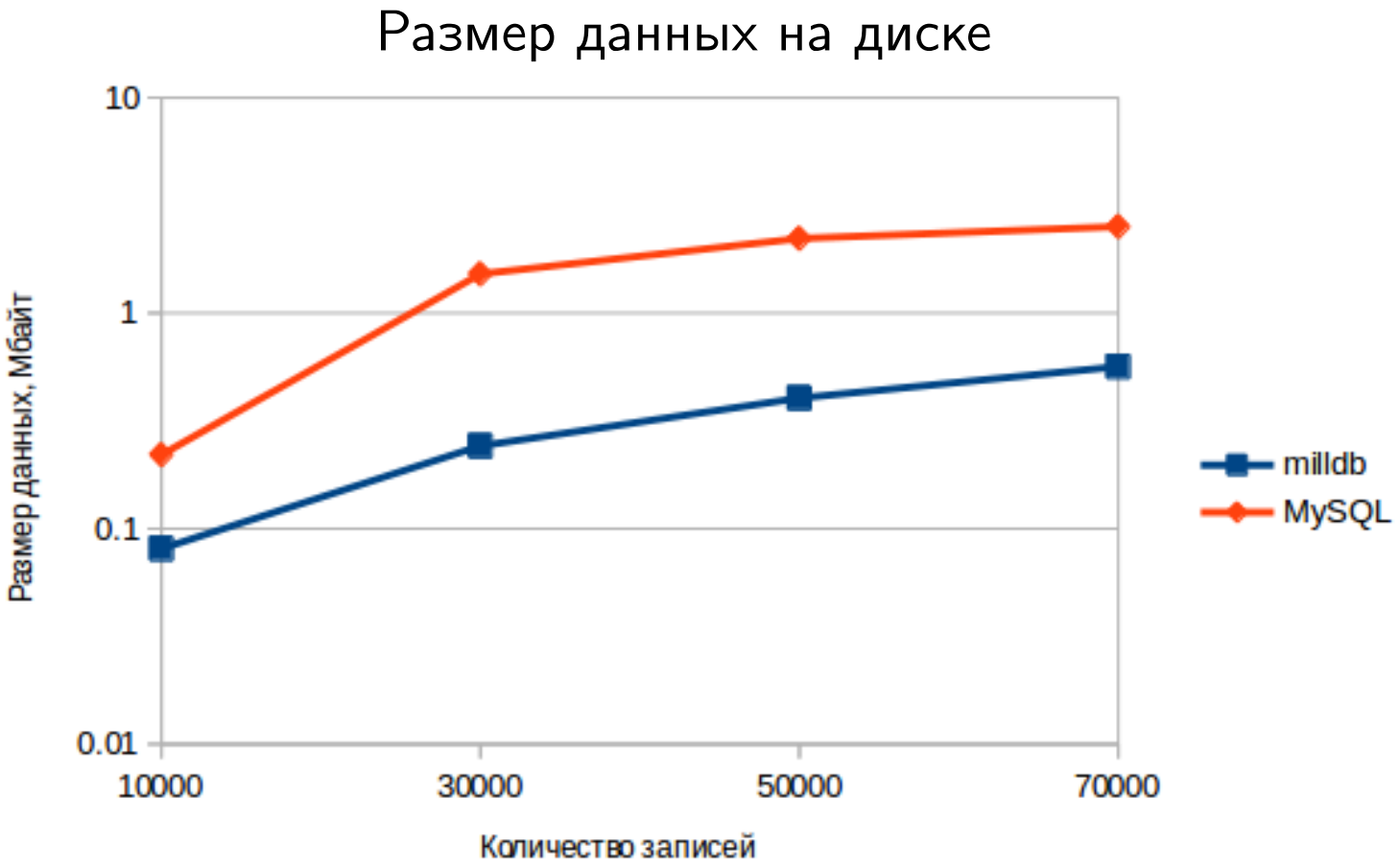
Характеристика	**Аналог**	**Разработанная БД**
Скорость операции вставки, с	18.52	11.27
Размер данных, Мбайт	95.12	350.12

Тестирование

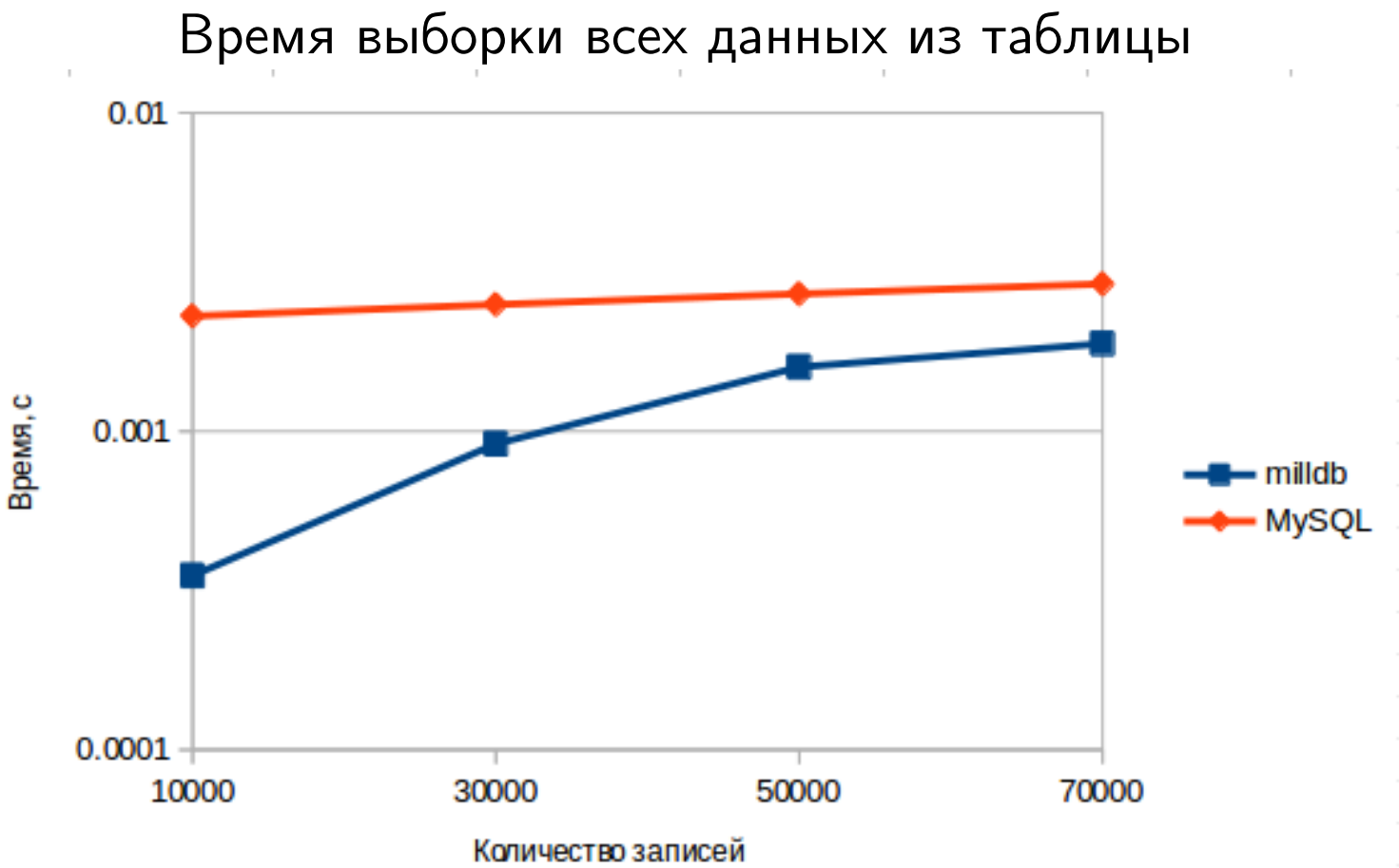
- сравнение с MySQL;
- параметры: таблица с 2 полями типов `int` и `char(4)`.



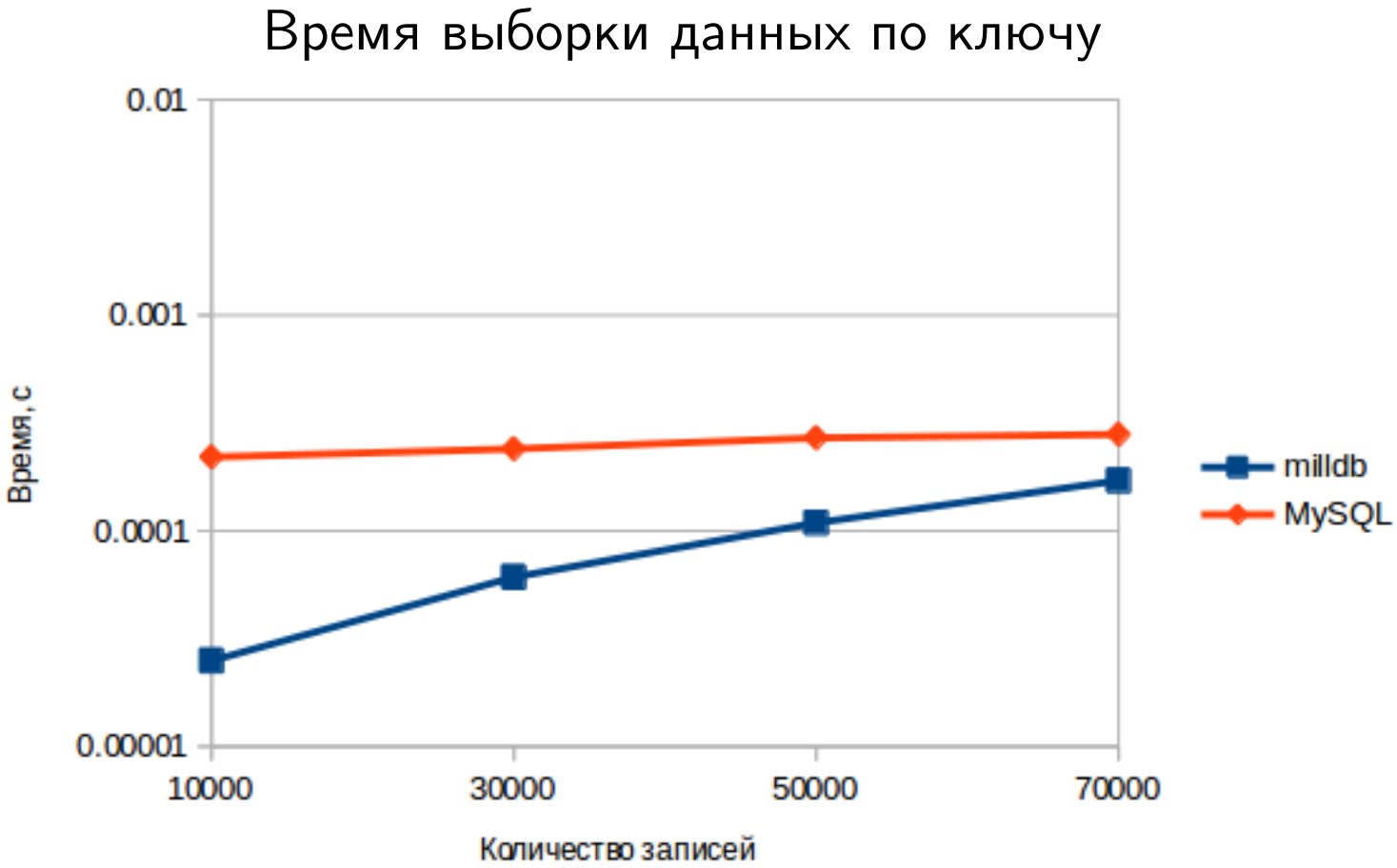
Тестирование



Тестирование



Тестирование



Благодарю за внимание