

Федеральное государственное автономное образовательное учреждение высшего  
образования "Национальный Исследовательский Университет ИТМО"  
Мегафакультет Компьютерных Технологий и Управления  
Факультет Программной Инженерии и Компьютерной Техники



**Модуль №1**  
по дисциплине  
**'Системы искусственного интеллекта'**

Выполнил Студент группы Р33102  
**Лапин Алексей Александрович**  
Преподаватель:  
**Кореньков Юрий Дмитриевич**

г. Санкт-Петербург  
2023г.

# Содержание

<b>1</b>	<b>Цель:</b>	<b>3</b>
<b>2</b>	<b>Задачи:</b>	<b>3</b>
2.1	Описание структур данных для представления данных в памяти: . . . . .	3
<b>3</b>	<b>Описание работы:</b>	<b>5</b>
3.1	Публичный интерфейс: . . . . .	5
3.1.1	Таблица . . . . .	5
3.1.2	Схема . . . . .	9
3.1.3	База данных . . . . .	10
3.2	Модули: . . . . .	11
<b>4</b>	<b>Аспекты реализации:</b>	<b>12</b>
<b>5</b>	<b>Результаты:</b>	<b>13</b>
5.1	Операция вставки: . . . . .	13
5.2	Операция выборки: . . . . .	13
5.3	Операция удаления: . . . . .	14
5.4	Операция обновления: . . . . .	14
5.5	Размер файла: . . . . .	15
5.6	Использование оперативной памяти: . . . . .	15
<b>6</b>	<b>Выводы:</b>	<b>15</b>

# 1 Цель:

Выданный вариант - 5 (реляционные таблицы, отображение файла) Целью является создать модуль, реализующий хранение в одном файле данных информации общим объёмом от 10GB в виде реляционных таблиц.

# 2 Задачи:

1. Модуль работающий с файлом на низком уровне(маппинг, синхронизация, чтение, запись и т.п) с одной страницей(file);
2. Модуль, который абстрагируется от низкоуровневых операций и предоставляет более удобный интерфейс для записи и чтения, а также осуществление кеширования страниц(caching);
3. Модуль, который еще больше абстрагируется от низкоуровневых операций и добавляет возможность переиспользовать страницы, после их удаления(pager);
4. Модуль, который соединяет страницы в список, для работы не с физическими страницами по 4кб, а с виртуальными условно бесконечными страницами. (linked-pages)
5. Модуль, который строит поверх linked-pages pool страниц, блоки которого можно выдавать за  $O(1)$ . (page-pool)
6. Модуль, который расширяет блоки в page-pool, связывая их, чтобы можно было работать не с физическими блоками фиксированного размера, а с виртуальными условно бесконечными блоками. (linked-blocks)
7. Модули, создания схемы таблицы и самих таблиц
8. Модули, хранения метаданных о существующих таблицах, строках переменной длины и т.п.
9. Операции над таблицами, вставка, удаление, обновление, выборка, соединение, сравнение rows и т.п.

## 2.1 Описание структур данных для представления данных в памяти:

```
1  /* Файл */
2  typedef struct file {
3      char *filename;
4      int fd;
5      void *cur_mmaped_data;
6      off_t cur_page_offset;
7      off_t file_size;
8      int64_t max_page_index;
9  } file_t;
10
11 /* Кеширование */
12 typedef struct caching{
13     file_t file;
```

```

14     size_t size, used, max_used, capacity;
15     uint32_t* usage_count;
16     time_t* last_used;
17     void** cached_page_ptr;
18     char* flags;
19 } caching_t;
20
21 /* Переиспользование страниц */
22 typedef struct pager{
23     caching_t ch;
24     int64_t deleted_pages; // index of parray page with deleted pages
25 } pager_t;
26
27 /* Связанные страницы */
28 typedef struct linked_page{
29     int64_t next_page;
30     int64_t page_index;
31     int64_t mem_start;
32 } linked_page_t;
33
34 /* Пул страниц */
35
36 /* Чанк */
37 typedef struct chunk {
38     linked_page_t lp_header;
39     int64_t page_index;
40     int64_t capacity;
41     int64_t num_of_free_blocks;
42     int64_t num_of_used_blocks;
43     int64_t next;
44     int64_t prev_page;
45     int64_t next_page;
46 } chunk_t;
47
48 /* Менеджер пула */
49 typedef struct page_pool {
50     linked_page_t lp_header;
51     int64_t current_idx;
52     int64_t head;
53     int64_t tail;
54     int64_t block_size;
55     int64_t wait; // parray index
56 } page_pool_t;
57
58 /* Связанные блоки */
59 typedef struct linked_block{
60     chblix_t next_block;
61     chblix_t prev_block;
62     chblix_t chblix;
63     char flag;
64     int64_t mem_start;
65 } linked_block_t;
66

```

```

67  /* Поле схемы */
68  typedef struct field{
69      linked_block_t lb_header;
70      char name[MAX_NAME_LENGTH];
71      datatype_t type;
72      uint64_t size;
73      uint64_t offset;
74  } field_t;
75
76  /* Схема таблицы */
77  typedef struct schema{
78      page_pool_t ppl_header;
79      int64_t slot_size;
80  } schema_t;
81
82  /* Таблица */
83  typedef struct table {
84      page_pool_t ppl_header;
85      int64_t schidx; //schema index
86      char name[MAX_NAME_LENGTH];
87  } table_t;
88
89  /* Типы данных */
90  typedef enum datatype {DT_UNKNOWN = -1,
91                        DT_INT = 0,
92                        DT_FLOAT = 1,
93                        DT_VARCHAR = 2,
94                        DT_CHAR = 3,
95                        DT_BOOL = 4} datatype_t;

```

## 3 Описание работы:

### 3.1 Публичный интерфейс:

#### 3.1.1 Таблица

```

1
2  /**
3   * @brief      Initialize table and add it to the metatable
4   * @param[in]  db: pointer to db
5   * @param[in]  name: name of the table
6   * @param[in]  schema: pointer to schema
7   * @return     index of the table on success, TABLE_FAIL on failure
8   */
9  table_t* tab_init(db_t* db, const char* name, schema_t* schema);
10 /**
11  * @brief      Get row by value in column
12  * @param[in]  db: pointer to db
13  * @param[in]  table: pointer to the table
14  * @param[in]  schema: pointer to the schema
15  * @param[in]  field: pointer to the field
16  * @param[in]  value: pointer to the value

```

```

17 * @param[in]    type: type of the value
18 * @return      chblix_t of row on success, CHBLIX_FAIL on failure
19 */
20 chblix_t tab_get_row(db_t* db,
21                     table_t* table,
22                     schema_t* schema,
23                     field_t* field,
24                     void* value,
25                     datatype_t type);
26 /**
27 * @brief        Print table
28 * @param[in]    db: pointer to db
29 * @param[in]    table: index of the table
30 * @param[in]    schema: pointer to the schema
31 */
32 void tab_print(db_t* db,
33               table_t* table,
34               schema_t* schema);
35 /**
36 * @brief        Inner join two tables
37 * @param[in]    db: pointer to db
38 * @param[in]    left: pointer to the left table
39 * @param[in]    left_schema: pointer to the schema of the left table
40 * @param[in]    right: pointer to the right table
41 * @param[in]    right_schema: pointer to the schema of the right table
42 * @param[in]    join_field_left: join field of the left table
43 * @param[in]    join_field_right: join field of the right table
44 * @param[in]    name: name of the new table
45 * @return      pointer to the new table on success, NULL on failure
46 */
47 int64_t tab_join(
48     db_t* db,
49     int64_t leftidx,
50     int64_t rightidx,
51     const char* join_field_left,
52     const char* join_field_right,
53     const char* name);
54 /**
55 * @brief        Select row form table on condition
56 * @param[in]    db: pointer to db
57 * @param[in]    sel_table: pointer to table from which the selection
58 *                        is made
59 * @param[in]    sel_schema: pointer to schema of the table from which
60 *                        the selection is made
61 * @param[in]    select_field: the field by which the selection is
62 *                        performed
63 * @param[in]    name: name of new table that will be created
64 * @param[in]    condition: comparison condition
65 * @param[in]    value: value to compare with
66 * @param[in]    type: the type of value to compare with
67 * @return      pointer to new table on success, NULL on failure
68 */
69 table_t* tab_select_op(db_t* db,

```

```

67     table_t* sel_table,
68     schema_t* sel_schema,
69     field_t* select_field,
70     const char* name,
71     condition_t condition,
72     void* value,
73     datatype_t type);
74 /**
75  * @brief      Drop a table
76  * @param[in]   db: pointer to db
77  * @param[in]   table: pointer of the table
78  * @return      PPL_SUCCESS on success, PPL_FAIL on failure
79  */
80 int tab_drop(db_t* db, table_t* table);
81
82 /**
83  * @brief      Update row in table
84  * @param[in]   db: pointer to db
85  * @param[in]   table: pointer to table
86  * @param[in]   schema: pointer to schema
87  * @param[in]   field: pointer to field
88  * @param[in]   condition: comparison condition
89  * @param[in]   value: value to compare with
90  * @param[in]   type: the type of value to compare with
91  * @param[in]   row: pointer to new row which will replace the old
92  *                  one
93  * @return
94  */
95 int tab_update_row_op(db_t* db,
96                       table_t* table,
97                       schema_t* schema,
98                       field_t* field,
99                       condition_t condition,
100                      void* value,
101                      datatype_t type,
102                      void* row);
103 /**
104  * @brief      Update element in table
105  * @param[in]   db: pointer to db
106  * @param[in]   tablix: index of the table
107  * @param[in]   element: element to write
108  * @param[in]   field_name: name of the element field
109  * @param[in]   field_comp: name of the field compare with
110  * @param[in]   condition: comparison condition
111  * @param[in]   value: value to compare with
112  * @param[in]   type: the type of value to compare with
113  * @return      TABLE_SUCCESS on success, TABLE_FAIL on failure
114  */
115
116 int tab_update_element_op(db_t* db,
117                          int64_t tablix,
118                          void* element,

```

```

119         const char* field_name,
120         const char* field_comp,
121         condition_t condition,
122         void* value,
123         datatype_t type);
124 /**
125  * @brief      Delete row from table
126  * @param[in]  db: pointer to db
127  * @param[in]  table: pointer to table
128  * @param[in]  schema: pointer to schema
129  * @param[in]  field_comp: pointer to field to compare
130  * @param[in]  condition: comparison condition
131  * @param[in]  value: value to compare with
132  * @return     TABLE_SUCCESS on success, TABLE_FAIL on failure
133  */
134 int tab_delete_op(db_t* db,
135                  table_t* table,
136                  schema_t* schema,
137                  field_t* comp,
138                  condition_t condition,
139                  void* value);
140 /**
141  * @brief      Insert a row
142  * @param[in]  table: pointer to table
143  * @param[in]  schema: pointer to schema
144  * @param[in]  src: source
145  * @return     chblix_t of row on success, CHBLIX_FAIL on failure
146  */
147
148 chblix_t tab_insert(table_t* table, schema_t* schema, void* src)
149
150 /** @brief      Create a table on a subset of fields
151  * @param[in]  db: pointer to db
152  * @param[in]  table: pointer to table
153  * @param[in]  schema: pointer to schema
154  * @param[in]  fields: pointer to fields
155  * @param[in]  num_of_fields: number of fields
156  * @param[in]  name: name of the new table
157  * @return     pointer to new table on success, NULL on failure
158  */
159
160 table_t* tab_projection(db_t* db,
161                       table_t* table,
162                       schema_t* schema,
163                       field_t* fields,
164                       int64_t num_of_fields,
165                       const char* name);
166
167 /**
168  * @brief      For each element specific column in a table
169  * @param[in]  table: pointer to the table
170  * @param[in]  chunk: chunk
171  * @param[in]  chblix: chblix of the row

```



```

172 * @param[in]    element: pointer to the element, must be allocated
    before calling this macro
173 * @param[in]    field: pointer to field of the element
174 */
175
176 #define tab_for_each_element(table, chunk, chblix, element, field)
177 /**
178  * @brief        For each element specific column in a table
179  * @param[in]    table: pointer to the table
180  * @param[in]    chunk: chunk
181  * @param[in]    chblix: chblix of the row
182  * @param[in]    row: pointer to the row, must be allocated before
    calling this macro
183  * @param[in]    schema: pointer to the schema
184  */
185
186 #define tab_for_each_row(table, chunk, chblix, row, schema)

```

### 3.1.2 Cxema

```

1  /**
2  * @brief        Delete a schema
3  * @param[in]    schidx: index of the schema
4  * @return       SCHEMA_SUCCESS on success, SCHEMA_FAIL on failure
5  */
6
7  #define sch_delete(schidx)
8
9  /**
10 * @brief        Load field
11 * @param[in]    schidx: index of the schema
12 * @param[in]    fieldix: index of the field
13 * @param[out]   field: pointer to the field
14 * @return       LB_SUCCESS on success, LB_FAIL on failure
15 */
16
17 #define sch_field_load(schidx, fieldix, field)
18
19 /**
20 * @brief        Update field
21 * @param[in]    schidx: index of the schema
22 * @param[in]    fieldix: index of the field
23 * @param[out]   field: pointer to the field
24 * @return       LB_SUCCESS on success, LB_FAIL on failure
25 */
26
27 #define sch_field_update(schidx, fieldix, field)
28
29 #define sch_add_int_field(schema, name) sch_add_field((schema),
    name, DT_INT, sizeof(int64_t))
30 #define sch_add_char_field(schema, name, size)
    sch_add_field((schema), name, DT_CHAR, size)

```

```

31 #define sch_add_varchar_field(schema, name) sch_add_field((schema),
    name, DT_VARCHAR, sizeof(vch_ticket_t))
32 #define sch_add_float_field(schema, name) sch_add_field((schema),
    name, DT_FLOAT, sizeof(float))
33 #define sch_add_bool_field(schema, name) sch_add_field((schema),
    name, DT_BOOL, sizeof(bool))
34
35 /**
36  * @brief      For each field in a schema
37  * @param[in]  sch: pointer to the schema
38  * @param[in]  chunk: chunk
39  * @param[in]  field: pointer to the field
40  * @param[in]  chblix: chblix of the row
41  * @param[in]  schidx: index of the schema
42  */
43
44 #define sch_for_each(sch, chunk, field, chblix, schidx)
45 /**
46  * @brief      Initialize a schema
47  * @return     pointer to schema on success, NULL on failure
48  */
49 void* sch_init(void);
50 /**
51  * @brief      Add a field
52  * @param[in]  schema: pointer to schema
53  * @param[in]  name: name of the field
54  * @param[in]  type: type of the field
55  * @param[in]  size: size of the type
56  * @return     SCHEMA_SUCCESS on success, SCHEMA_FAIL on failure
57  */
58 int sch_add_field(schema_t* schema, const char* name, datatype_t
    type, int64_t size);
59 /**
60  * @brief      Get a field
61  * @param[in]  schema: pointer to the schema
62  * @param[in]  name: name of the field
63  * @param[out] field: pointer to destination field
64  * @return     SCHEMA_SUCCESS on success, SCHEMA_FAIL on failure
65  */
66 int sch_get_field(schema_t* schema, const char* name, field_t*
    field);
67 /**
68  * @brief      Delete a field
69  * @warning     This function delete field, but dont touch offsets
    in other fields in schema.
70  * @param[in]  schema: pointer to schema
71  * @param[in]  name: name of the field
72  * @return     SCHEMA_SUCCESS on success, SCHEMA_FAIL on failure
73  */
74 int sch_delete_field(schema_t* schema, const char* name);

```

### 3.1.3 База данных

```

1 /**
2  * @brief      Initialize database
3  * @param[in]   filename: name of the file
4  * @return      pointer to database on success, NULL on failure
5  */
6 void* db_init(const char* filename);
7 /**
8  * @brief      Close database
9  * @return      DB_SUCCESS on success, DB_FAIL on failure
10 */
11 int db_close(void);
12 /**
13  * @brief      Drop database
14  * @return      DB_SUCCESS on success, DB_FAIL on failure
15  */
16 int db_drop(void);

```

## 3.2 Модули:

Директория core - содержит модули, строящие абстракции над работой с файлом, вводом-выводом и т.п.

1. file - осуществляет базовую работу с файлом, при использовании одной страницы.
2. caching - кеширует страницы, освобождает их по мере заполнения оперативной памяти.
3. pager - хранит очередь страниц, готовых для переиспользования.
4. liked\_pages - реализация связанных страниц.
5. page\_pool - реализация пула страниц.
6. linked\_blocks - реализация связанных блоков.

Директория backend - содержит модули, реализовывающие работу с реляционной базой данных.

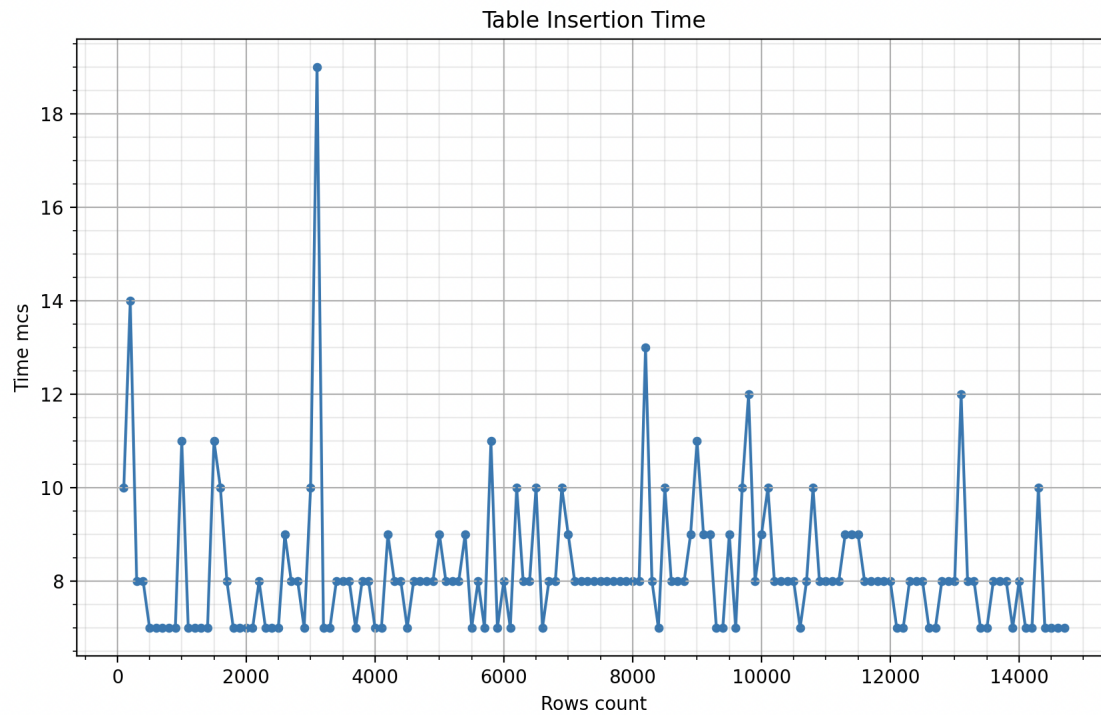
1. schema - реализация операций с схемой таблиц
2. table\_base - базовые операции с таблицами (нужны для других модулей)
3. matatab - таблица с метаданными (название и индекс страницы) других таблиц.
4. varchar\_mgr - пул хранящий строки произвольной длины и выдающий номерки для их поиска в нем.
5. db - модуль выполняющий операции на старте (инициализация файла, метатаблицы, varchar\_mgr), и в завершении работы программы (закрытие файла, удаление файла). Также хранит индекс метатаблицы и тп.
6. data-types - поддерживаемые типы данных.
7. comparator - сравнение поддерживаемых типов.

## 4 Аспекты реализации:

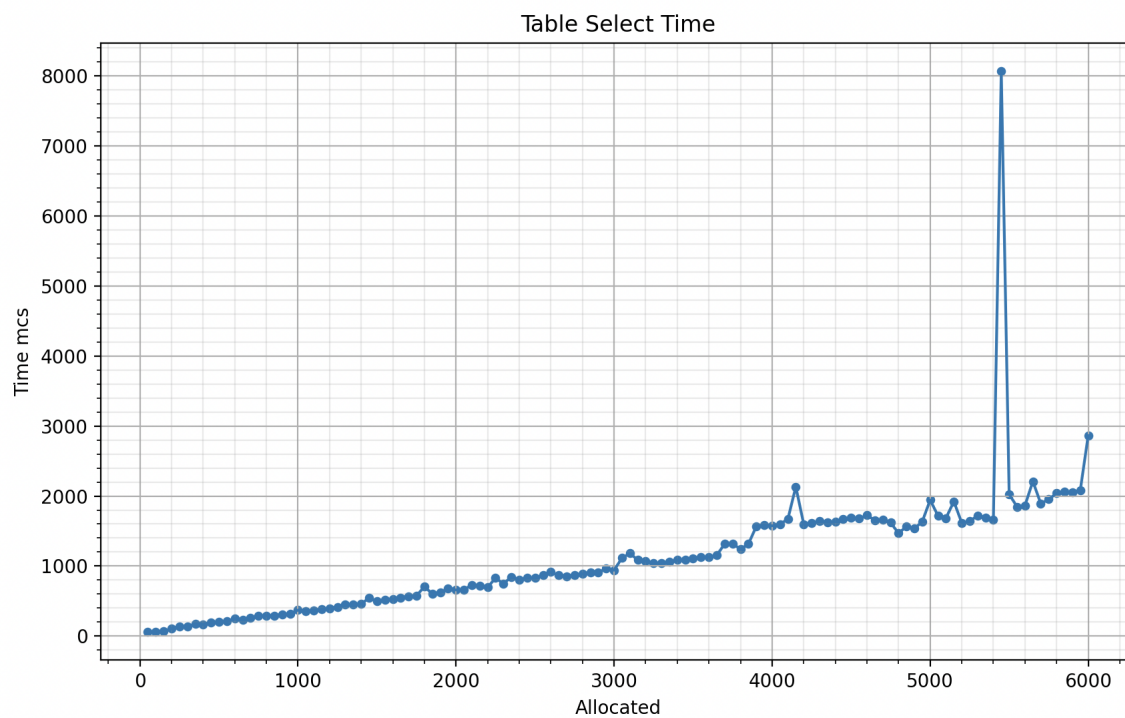
1. Данные хранятся в страницах стандартного размера для операционной системы (обычно 4 кб).
2. Страницы кешируются и освобождаются по необходимости. Для этого создана map ключем в которой является номер страницы, а значением указатель на область памяти, статус-флаги, последнее время использования, количество использования. Освобождение страницы из конца файла удаляет страницу. Освобождение страницы из середины файла ставит её в очередь на переиспользование при следующем запросе на аллокацию.
3. Чтобы поддерживать возможность добавление единиц данных больше размера страницы, страницы могут расширяться путем связывания linkedlist с новыми страницами. Для обращения и удаления этих страниц используется номер первой входящей в список страницы.
4. Чтобы выдавать элементы(строки) за  $O(1)$  на страницах построен pool. Pool состоит из chunks размером в одну связную страницу, chunks связаны между собой двусвязным linkedlist. При отдаче всех блоков из chunk мы переставляем указатель на следующий. Когда в chunk освобождается блок, то он ставится в очередь на использование, когда в текущем chunk закончится место. Если в chunk освобождены все строки, то страница полностью удаляется.
5. Чтобы поддерживать возможность записывать в блоки данные произвольной длины, блоки могут расширяться связываясь с помощью linkedlist.
6. Схема таблицы построена поверх пула страниц, а поля являются блоками этого пула.
7. Таблицы строятся поверх своих пулов страниц, строки являются блоками пула.

## 5 Результаты:

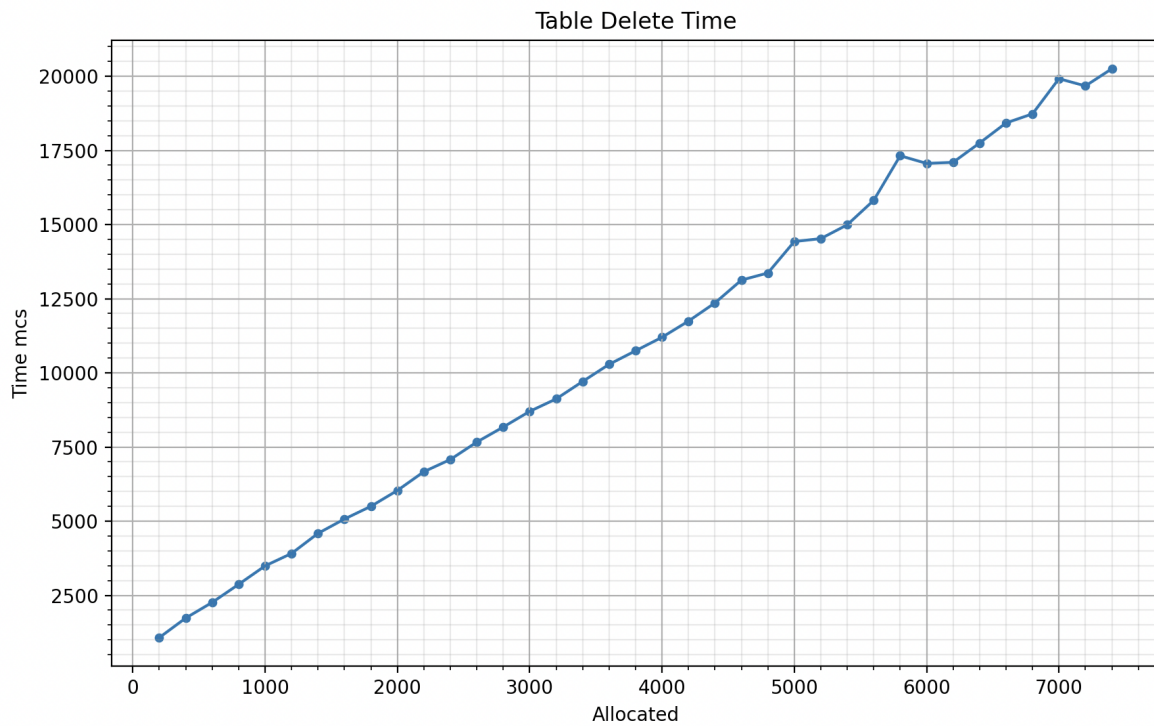
### 5.1 Операция вставки:



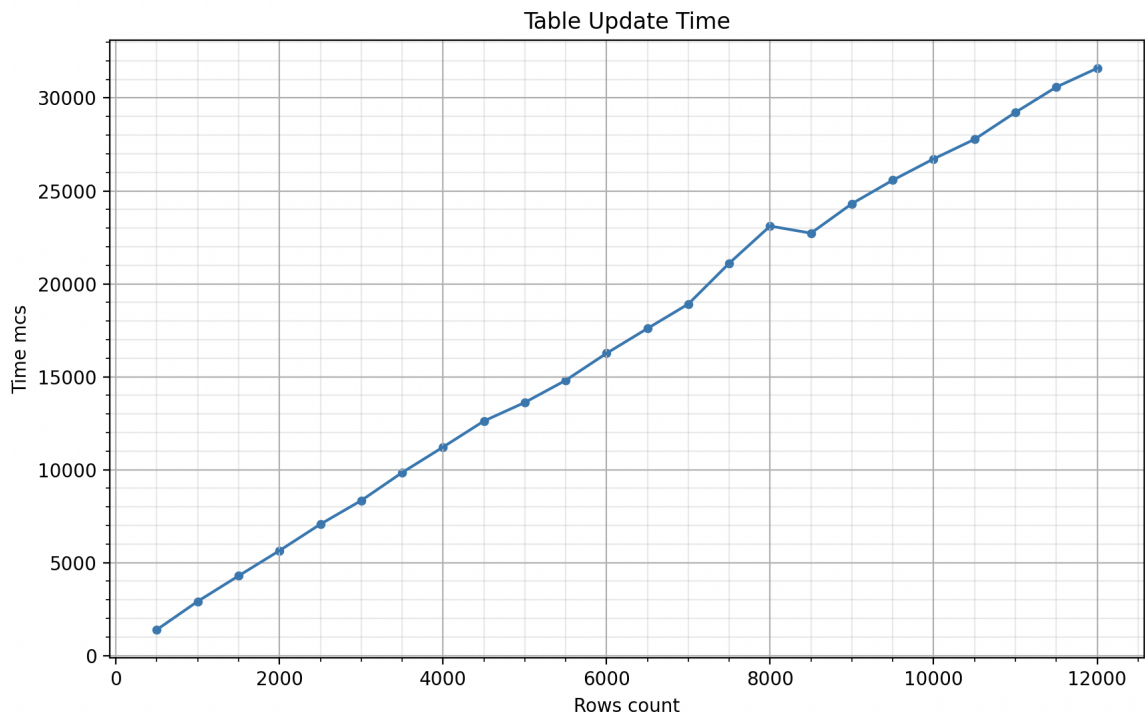
### 5.2 Операция выборки:



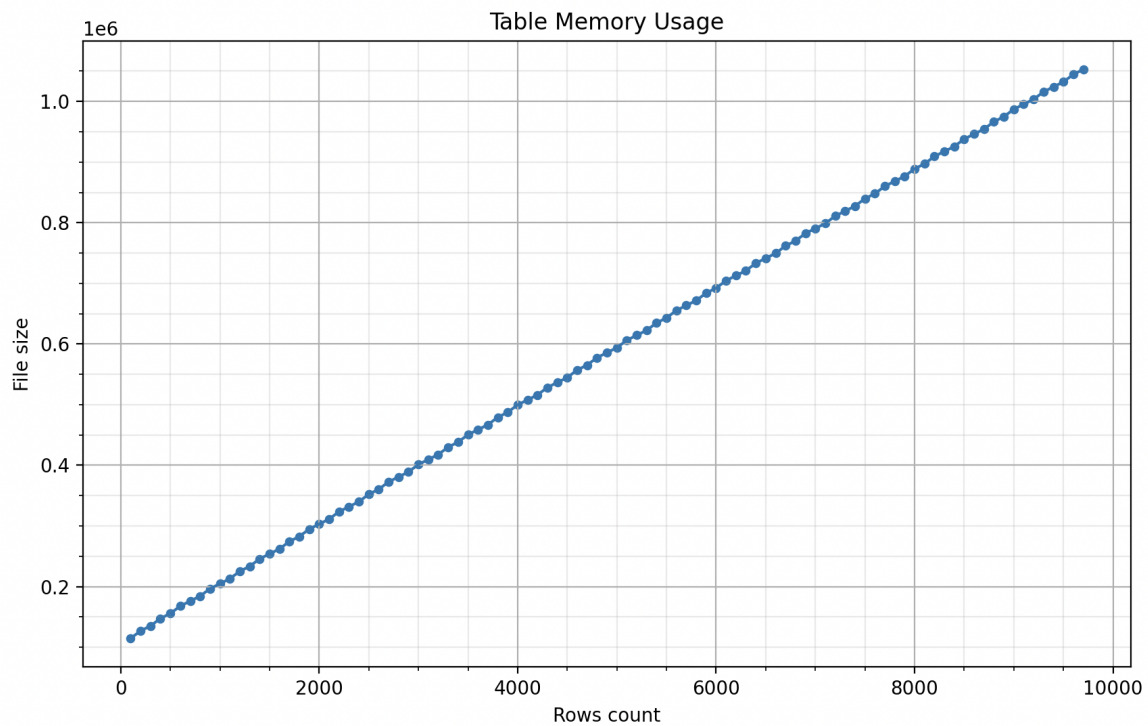
### 5.3 Операция удаления:



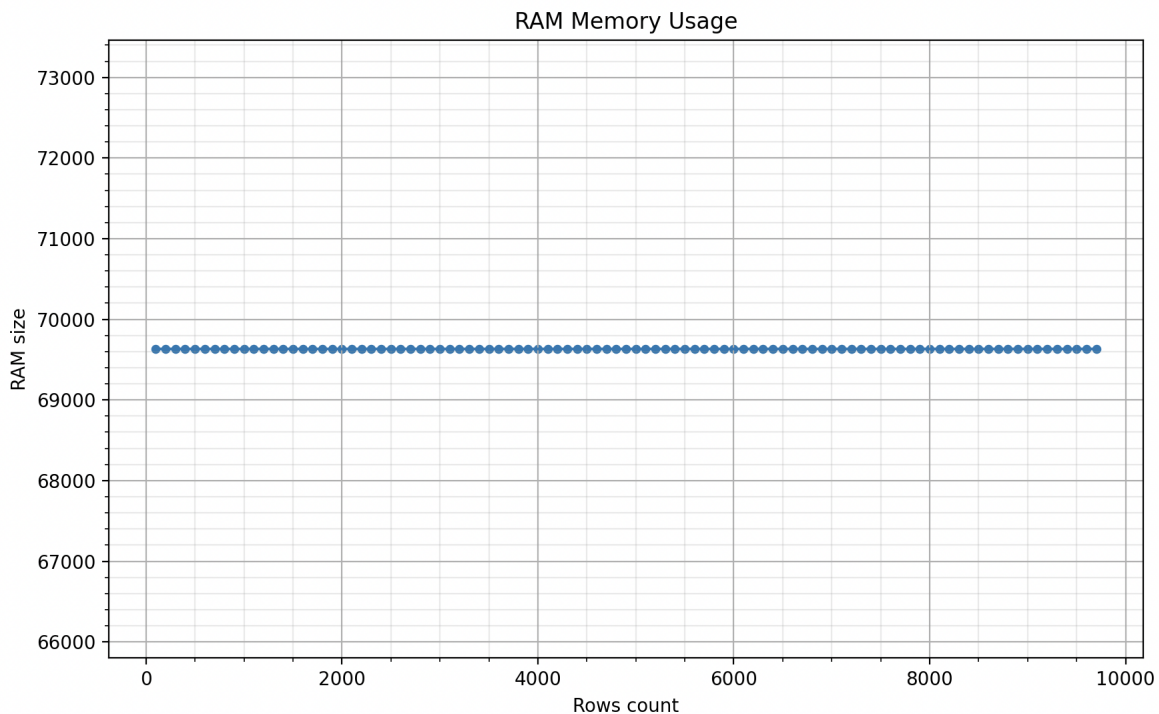
### 5.4 Операция обновления:



## 5.5 Размер файла:



## 5.6 Использование оперативной памяти:



## 6 Выводы:

Как видно из графика вставка работает за  $O(1)$ , скачки на графике связаны скорее всего с аллокацией дополнительных страниц, как для хранения строк, так и для хранения

очереди.

Выборка работает за  $O(n)$ , так как мы проходим по всем строкам. Есть одна точка, которая сильно отличается от остальных, считаю её выбросом.

Удаление работает за  $O(n)$ , так как мы проходим по всем строкам.

Обновление работает за  $O(n)$ , так как мы проходим по всем строкам.

Размер файла растёт линейно, так как мы добавляем новые страницы.

Использование оперативной памяти  $O(1)$ , так как мы загружаем страницы, когда нам они нужны и освобождаем, когда уже нет.

**Что я узнал, чему научился:**

1. Написал свою первую большую программу на C, больше прогрузился в язык, в работу с ним.
2. Погрузился в то как работают базы данных изнутри, написал свою.
3. Узнал новые алгоритмы и структуры данных, научился сам их реализовывать.
4. Научился работать с файлами, работать с памятью, работать с потоками ввода-вывода.
5. Научился писать переносимые приложения под разные операционные системы: Linux, Windows, MacOS.