МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное автономное
образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»

ФАКУЛЬТЕТ ПРОГРАММНОЙ ИНЖЕНЕРИИ И КОМПЬЮТЕРНОЙ ТЕХНИКИ

Лабораторная работа 1
по дисциплине:
Низкоуровневое программирование
Вариант 5

Выполнил:
Студент группы P33311
Кириллов Андрей
Преподаватель:
Кореньков Юрий
Дмитриевич

УНИВЕРСИТЕТ ИТМО

Санкт-Петербург, 2023

Цель

Создать модуль, реализующий хранение в одном файле данных (выборку, размещение и гранулярное

обновление) информации общим объёмом от 10GB соответствующего варианту вида.

Задачи

1. Спроектировать структуры данных для представления информации в оперативной памяти

2. Спроектировать представление данных с учетом схемы для файла данных и реализовать базовые операции для работы с ним

3. Используя в сигнатурах только структуры данных из п.1, реализовать публичный интерфейс со следующими операциями над файлом данных:

a. Добавление, удаление и получение информации об элементах схемы данных, размещаемых в файле данных, на уровне, соответствующем виду узлов или записей

b. Добавление нового элемента данных определённого вида

c. Выборка набора элементов данных с учётом заданных условий и отношений со смежными элементами данных (по свойствам/полями/атрибутам и логическим связям соответственно)

d. Обновление элементов данных, соответствующих заданным условиям

e. Удаление элементов данных, соответствующих заданным условиям

4. Реализовать тестовую программу для демонстрации работоспособности решения

Описание структур для представления информации о запросе

```
struct RecordData {
	struct BlockHeader *rowPageBlock;
	char *unreadData;
```

```
	struct Column **columns;
	void **data;
};
```

Эта структура есть то, что возвращается из запроса - данные. А далее описаны структуры для формирования самого запроса:

```
enum DataType {
	BOOL = 0,
	INT = 1,
	STRING = 2,
	DOUBLE = 3
};
enum Operator {
	MORE, LESS, EQUALS
};

struct FreeVar {
	enum DataType dataType;
	void *operand;
};


struct Operand {
	bool isOperandAName;
	union {
	struct FreeVariable *freeVariable;
	char *columnName;
	} operandValue;
};
struct Condition {
	struct Operand left;
	struct Operand right;
	enum Operator op;
};
```

```
struct CompareResult {
	int32_t errorCode;
	int32_t compareResult;
};


struct UpdateColumnValue {
	char *name;
	struct Operand assignedValue;
};
struct JoinTablesHeaders {
	struct TableHeader *tableHeader;
	char *tableAlias;
};
struct JoinOperand {
	char *tableAlias;
	char *columnName;
};
struct JoinCondition {
	struct JoinOperand leftOperand;
	struct JoinOperand rightOperand;
	enum Operator operator;
};

struct JoinWhereOperand {
	bool isOperandAName;
	struct FreeVariable *freeVariable;
};
struct JoinWhereCondition {
	struct JoinWhereOperand leftOperand;
	struct JoinWhereOperand rightOperand;
```

```
	enum Operator operator;
};
```

Представление схемы данных в файле

Файл разбит на страницы непостоянного размера, заголовок файла это DatabaseHeader. В нем содержится корневая информация. Вот сама структура:

```
struct DatabaseHeader {
	uint64_t size;
	struct BlockCoordinate tableMapPage;
	struct BlockCoordinate freeList;
};
```

В этой структуре можно увидеть BlockCoordinate, так вот эта структура нужна для указания на блок. В ней есть отступ от начала файла и размер этого блока, а в начале каждого блока есть BlockHeader, хранящий информацию о блоке, после заголовка уже идет полезная информация. Вот эти две структуры:

```
struct BlockCoordinate{
	uint64_t offset;
	uint64_t size;
};
struct BlockHeader{
	uint64_t offset;
	uint64_t size;
	uint64_t previousBlockOffset;
	bool isFree;
};
```

В полезной части блока может лежать TableHeader.

```
struct TableHeader {
	uint64_t offsetFromBlockBegin;
	struct BlockCoordinate firstRowPage;
	int32_t columnCount;
	int32_t rowCount;
};
```

В нем хранится информация о типах и именах колонок, количетво строк и столбцов, а также ссылка на первых блок с данными. Теперь про организацию собственно страниц строк.

```
struct RowPage {
	struct BlockCoordinate nextRowPage;
	struct BlockCoordinate prevRowPage;
	int32_t rowCount;
	uint64_t freeSpaceOffsetStart;
	uint64_t freeSpaceOffsetEnd;
};
```

Можно видеть, что они объединены в связанный список, где есть ссылка на предыдущий и следующий блоки. Таким образом можно итерироваться по всем данным таблицы. Данные добавляются в конец страницы. Также храним данные о начале и конце свободного места.

```
struct RowPointer {
	bool isToDelete;
	uint64_t offset;
};
```

Эта структура находится в начале RowPage и хранит указатели на строки данных и флажок о том, нужно ли удалить строку.

```
struct Column {
	enum DataType dataType;
	char name[];
};
```

Эта структура описывает столбец.

Собственно, был сформирован следующий интерфейс для работы с бд:

```
int32_t openDatabaseFileOrCreate(const char *path);
int32_t closeDatabaseFile(void);
```

Интерфейс для работы с таблицами (DDL):

```
int32_t createTable(const char *name, int32_t columnNum, enum DataType
*types, const char **names);
int32_t deleteTable(const char *name);
int32_t deleteTableByHeader(struct TableHeader *tableHeader);
int32_t getTable(struct TableHeader **table, const char *name);
```

```
int32_t freeTable(struct TableHeader *tableHeader);
```

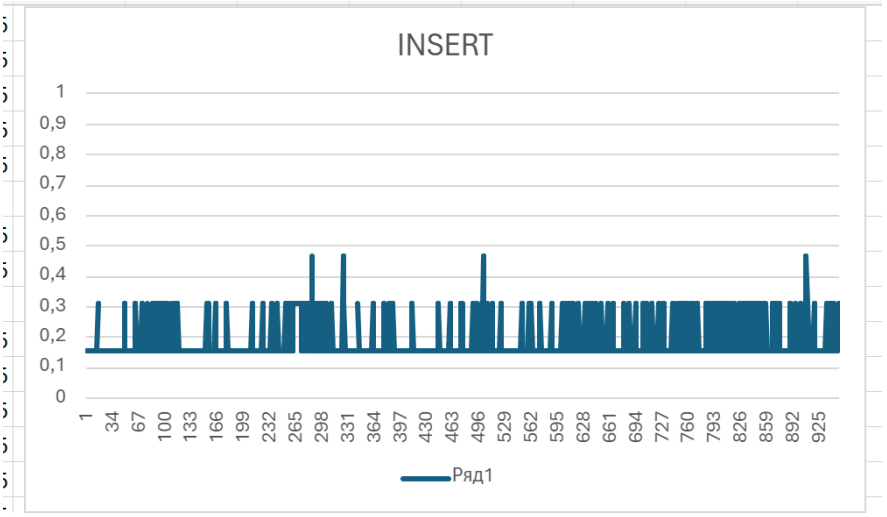
Интерфейс для работы с данными таблиц (DML):

```
int32_t addRow(struct TableHeader *tableHeader, struct RecordData

*recordData);

int32_t readRow (struct RecordData *recordData, struct TableHeader

*tableHeader);

int32_t deleteRow(struct TableHeader *tableHeader, int32_t
conditionCount,

struct Condition *conditions);

int32_t updateRow(struct TableHeader *tableHeader, int32_t
updateColumnsCount,

struct UpdateColumnValue *updateColumnValues, int32_t conditionCount,

struct Condition *conditions);

int32_t select(struct RecordData *recordData, struct TableHeader

*tableHeader, int32_t conditionCount, struct Condition *conditions);

int32_t joinTables (struct JoinResult *joinResult, int32_t
selectColumnsNum,

struct TableAliasAndColumn *selectColumns, int32_t joinTablesNum,

struct JoinTablesHeaders *joinTables, int32_t joinConditionNum, struct

JoinCondition *joinCondition, int32_t filtersNum, struct

JoinWhereCondition *joinWhereCondition);
```

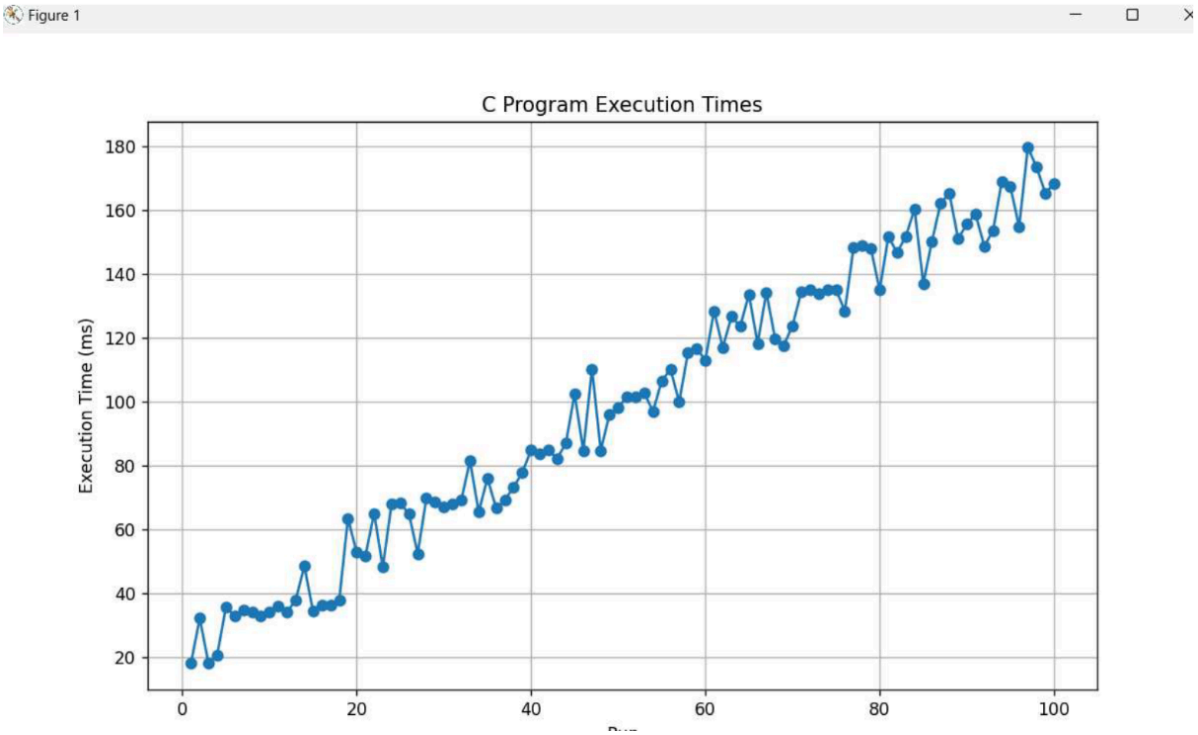
Как пользователю использовать: получаем ссылку на TableHeader и передаем ее в нужную функцию.

Показатели ресурсоемкости

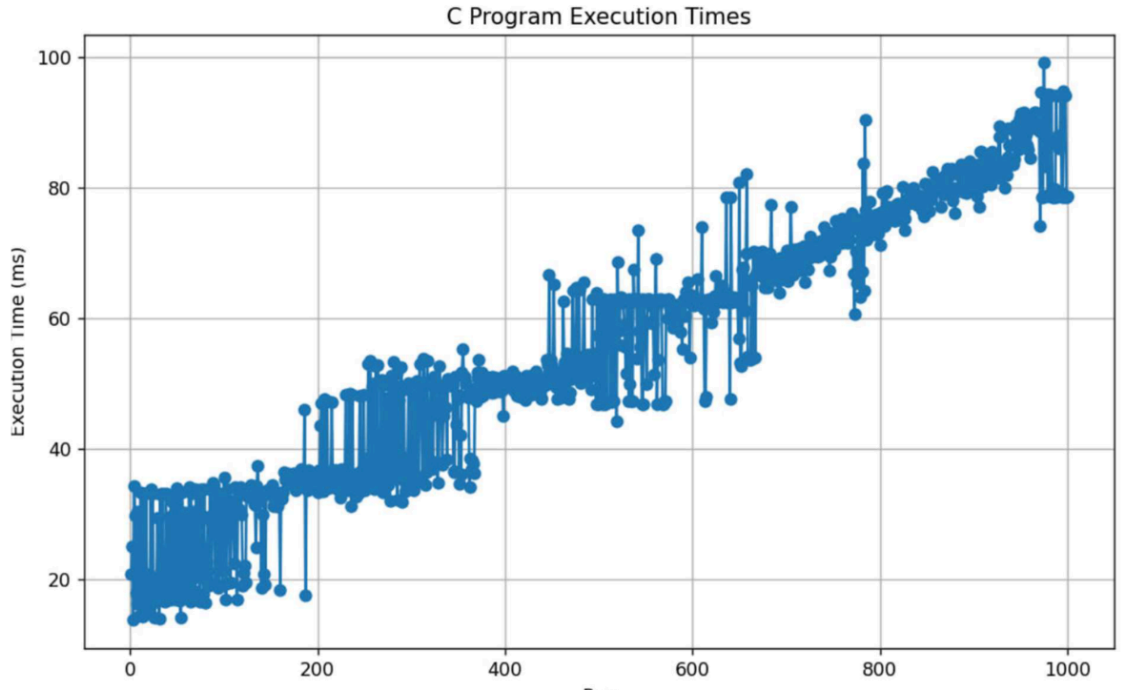
INSERT:



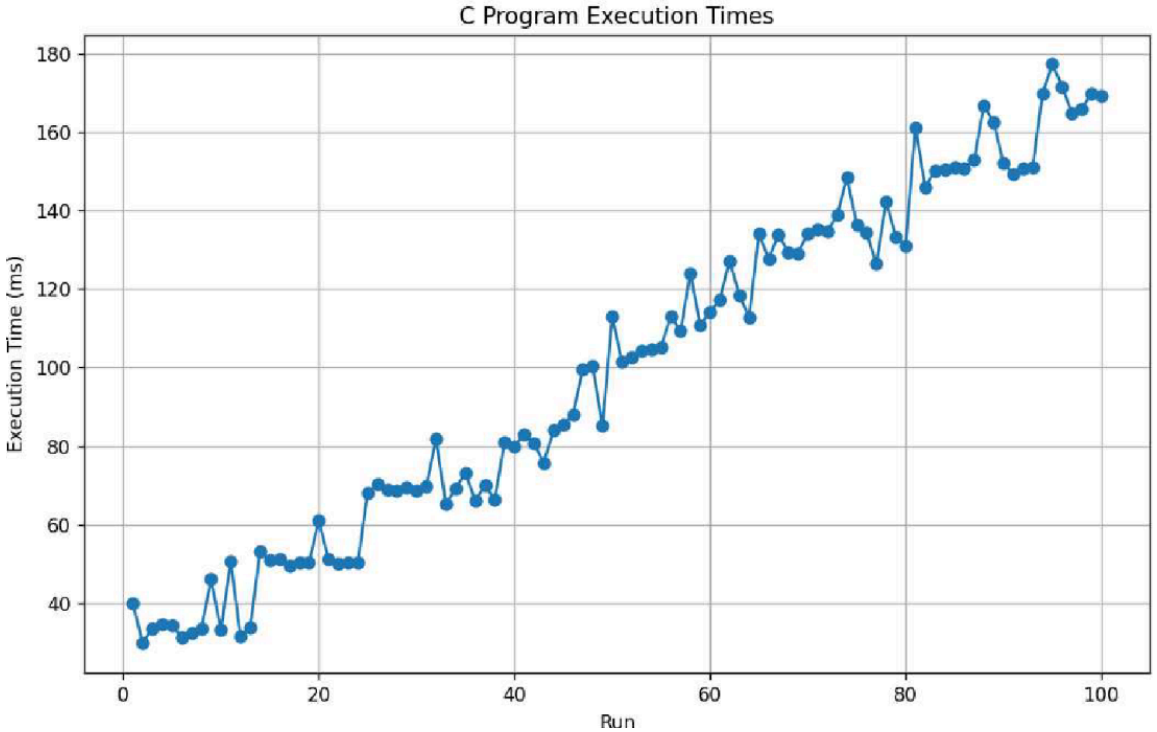
UPDATE:



READ:


C Program Execution Times
Execution Time (ms)
100
80
60
40
20
0
200
400
600
800
1000

DELETE:


C Program Execution Times
Execution Time (ms)
180
160
140
120
100
80
60
40
0
20
40
60
80
100
Run

Вывод

В ходе выполнения лабораторной работы я реализовал модуль хранения данных в одном файле. Было сложно, но полезно.