Федеральное государственное автономное образовательное учреждение высшего образования

«**Университет ИТМО»**

Факультет ПИиКТ

Дисциплина: Хранилища и базы данных

Лабораторная работа 2

Выполнили: Гурин Евгений Иванович, Камышанская Ксения Васильевна

Преподаватель: Королёва Юлия Александровна

Группа: P4116

Санкт-Петербург 2023г.

Цель работы

Написать модуль на выбранном языке программирования, который примет ключи и значения и построит над ними файловый индекс, так чтобы потом можно было его взять и тем же модулем поискать значения по ключу в файле.

Реализовать один из индексов:

- хеш-индекс,

- SS-таблиц,

- LSM-деревьев

- B-деревьев

Выполнение

Для разработки были выбраны:

- Язык Python
- хеш индекс

Репозиторий: https://github.com/GulDilin/key-value-database

Перед разработкой модуля индексирования была разработана однофайловая СУБД со следующей структурой файла:

```
|-------------------------------- database.db-lab ----------------------------------------|
| prefix .... database meta .... created | first_table_offset --> | last_table_offset --> .... |
| ..................................................................................... |
| .... table meta .... name | keys | indexed_keys | first_row_offset --> | last_row_offset --> |
| next_table_offset --> | prev_table_offset --> ..............................................|
| ..................................................................................... |
| .... row meta .... data | next_tow_offset --> | prev_row_offset --> .......................|
|-------------------------------------------------------------------------------------------|
```

Хранение таблиц и строк организовано на основе записи метаданных с оффсетами внутри файла на следующие и предыдущие строки\таблицы. Для хранения использованы следующие структуры:

Метаданные базы данных хранят информацию о времени создания, оффсеты на первую и последнюю таблицы

```
class MetaDB(BaseModel):
    created: datetime
    first_table_offset: int = 0
    last_table_offset: int = 0
```

Метаданные таблицы хранят информацию о имени, ключах таблицы, наименовании ключей, для которых созданы индексы, а также оффсеты на первую и последнюю строки, следующую и предыдущую таблицы

```
class MetaTable(BaseModel):
    name: str
    keys: dict[str, DbType]
    indexes: list[str]
    first_row_offset: int = 0
    last_row_offset: int = 0
    next_table_offset: int = 0
    prev_table_offset: int = 0
```

Метаданные строки хранят данные строки по типу ключ-значение и оффсеты на следующую и прерыдущую строки

```
class MetaRow(BaseModel):
    data: dict
    next_row_offset: int = 0
    prev_row_offset: int = 0
```

Был разработан модуль курсора базы данных, который напрямую взаимодействует с файлом БД https://github.com/GulDilin/key-value-database/blob/master/app/cursor.py

Был разработан модуль базы, который взаимодействует с курсором и предоставляет апи создания базы, таблиц, строк, а также получения строк с фильтрацией https://github.com/GulDilin/key-value-database/blob/master/app/db.py

Далее был разработан модуль парсинга пользовательских команд

https://github.com/GulDilin/key-value-database/blob/master/app/parser.py

Поддерживаются следующие команды:

Получение данных и фильтрация, есть возможность указать фильтр и/или по произвольным полям, для «ИЛИ» используется список, для «И» словарь. Например

```
{ age: 1 }
{ age: 1, title: text }
[{ age: 1 }, { title: text }]
[{ age: [1, 2] }, { title: text }]
```

```
usage: select [-h] --table TABLE [--limit LIMIT] [--use-index] [--all] [--
counter]
              [--filter FILTER_]

options:
  -h, --help            show this help message and exit
  --table TABLE, -t TABLE
                        Table name
  --limit LIMIT, -l LIMIT
                        Rows limit
  --use-index, -i       Use indexes in select
  --all                 Do not pause select
  --counter             Only count items
  --filter FILTER_, -f FILTER_
                        [{ key: val }, ... ] or { key: val, ... }
```

Пример использования

```
select -t Cats --counter
select -t Cats --counter
select -t Cats -f '{name:Pretty Cat}'
select -t Cats -f '{age:0}' --all --use-index
select -t Cats -f '{age:0}' --all --use-index --counter
select -t Cats -f '{age:0}' --all
select -t Cats -f '{age:0}' --all --counter
select -t Cats -f '{name:Kitty}' --all --counter
select -t Cats -f '{name:Kitty}' --all --use-index
select -t Cats -f '{name:Kitty}' --all --counter --use-index

select -t Cats -f '{age:1}' --all
select -t Cats -f '{age:1}' --all --use-index
select -t Cats -f '[{age:1},{age: 2}]' --all --use-index
select -t Cats -f '[{age:1},{age: 2}]' --all
select -t Cats -f '[{age:[1, 2]},{owner: Lilly}, {age:3}]' --all --use-
index --counter
```

Создание таблицы

```
usage: create-table [-h] table

positional arguments:
  table       { name, keys: { key: type } }

options:
  -h, --help  show this help message and exit
```

Пример использования

```
create-table { name: Test, keys: { id: int, content: str } }
create-table { name: Cats, keys: { name: str, age: int, owner: str } }
```

Создание индекса

```
usage: create-index [-h] --table TABLE --key KEY

options:
  -h, --help            show this help message and exit
  --table TABLE, -t TABLE
                        Table name
  --key KEY, -k KEY     Table key
```

Пример использования

```
create-index -t Cats -k age
create-index -t Cats -k name
create-index -t Cats -k owner
```

Получение списка таблиц

```
usage: list-tables [-h]

options:
  -h, --help  show this help message and exit
```

Вставка в таблицу

```
usage: insert [-h] --table TABLE [--data DATA]

options:
  -h, --help            show this help message and exit
  --table TABLE, -t TABLE
                        Table name
  --data DATA, -d DATA  { key: val, ... }
```

Примеры

```
insert -t Cats -d '{ name: Kitty, age: 2, owner: Lilly}'
```

Вставка в таблицу с автогенерацией нужного количества элементов на основе типов данных, полученных из таблицы. Для строковых генерируется uuid, для числовых случайное целочисленное значение.

```
usage: insert-auto [-h] --table TABLE [--amount AMOUNT]

options:
  -h, --help            show this help message and exit
  --table TABLE, -t TABLE
                        Table name
  --amount AMOUNT, -a AMOUNT
                        Rows amount
```

Модуль индексации

Модуль индексации работает с использованием хранения значений по типу хеш – список оффсетов на строки. При этом хеш вычисляется для значения определенного ключа

Итоговая структура внутреннего хранения индексов следующая:

```
{ table_name: { table_key: { hash_value: [ offset, ... ] } } }
```

При старте модуль пытается импортировать индекс из json файла, а при окончании работы экспортирует в файл.

Код модуля индексации

```
import hashlib
import json
from dataclasses import dataclass, field
from itertools import chain
from typing import Any, Generator

from . import types
from .cursor import DatabaseCursor
```

```
@dataclass
class Indexer:
    cursor: DatabaseCursor
    # { table_name: { key: { hash: [ offset, ... ] } } }
    index_dict: dict[str, dict[str, dict[str, list[int]]]] =
field(default_factory=dict)

    @staticmethod
    def get_md_5_bytes_hash(bytes):
        result = hashlib.md5(bytes).hexdigest()
        return result

    @staticmethod
    def hash(it: Any):
        return Indexer.get_md_5_bytes_hash(str(it).encode('utf-8'))

    def _add_val(self, meta_table: types.MetaTable, key: str, meta_row:
types.MetaRow, row_offset: int):
        if meta_table.name not in self.index_dict:
            self.index_dict[meta_table.name] = {}
        if key not in self.index_dict[meta_table.name]:
            self.index_dict[meta_table.name][key] = {}
        hash_v = self.hash(meta_row.data[key])
        if hash_v not in self.index_dict[meta_table.name][key]:
            self.index_dict[meta_table.name][key][hash_v] = []
        if row_offset not in self.index_dict[meta_table.name][key][hash_v]:

self.index_dict[meta_table.name][key][hash_v].append(row_offset)

    def add_item(self, meta_table: types.MetaTable, meta_row:
types.MetaRow, row_offset: int):
        for key in meta_table.indexes:
            self._add_val(meta_table, key, meta_row, row_offset)

    def get_offsets_for(self, meta_table: types.MetaTable, key: str, value:
Any):
        if key not in self.index_dict[meta_table.name]:
            raise ValueError(f'Index for key {key} in table
{meta_table.name} does not exists')
        hash_v = self.hash(value)
        return self.index_dict[meta_table.name][key].get(hash_v, [])

    def build_for_table(self, table_name: str):
        meta_table = self.cursor.get_table_by_name(table_name)
        offset = meta_table.first_row_offset
        while offset:
            meta_row = self.cursor.read_row_meta(offset)
            self.add_item(meta_table, meta_row, offset)
            offset = meta_row.next_row_offset

    def build_for_table_key(self, table_name: str, key: str):
        meta_table = self.cursor.get_table_by_name(table_name)
        if key not in meta_table.keys:
            raise ValueError(f'Key {key} does not present in table
{table_name}')
        offset = meta_table.first_row_offset
```

```
        while offset:
            meta_row = self.cursor.read_row_meta(offset)
            self._add_val(meta_table, key, meta_row, offset)
            offset = meta_row.next_row_offset

    def save(self):
        print('Saving index to file')
        with open(f'{self.cursor.db_file}.index.json', 'w') as f:
            json.dump(self.index_dict, f, indent=2)

    def load(self):
        print('Loading index from file')
        with open(f'{self.cursor.db_file}.index.json', 'r') as f:
            self.index_dict = json.load(f)
        print('Index loaded')

    def get_filter_keys_for_indexes(
        self,
        meta_table: types.MetaTable,
        filter_: types.Filter,
    ) -> set[str]:
        filter_keys = set(chain(*[part.keys() for part in filter_])) \
            if isinstance(filter_, list) else set(filter_.keys())
        for key in filter_keys:
            if key not in meta_table.indexes:
                raise ValueError(f'Index for key {key} does not present in
table {meta_table.name}')
        return filter_keys

    def get_filter_part_val_indexes_offsets(
        self,
        meta_table: types.MetaTable,
        key: str,
        value: types.FilterValue,
    ) -> set[int]:
        result = set()
        if isinstance(value, list):
            for v in value:
                result = result.union(self.get_offsets_for(meta_table, key,
v))
        else:
            result = set(self.get_offsets_for(meta_table, key, value))
        return result

    def get_filter_part_indexes_offsets(
        self,
        meta_table: types.MetaTable,
        filter_part: types.FilterPart,
    ) -> set[int]:
        result = None
        for key, value in filter_part.items():
            sub = self.get_filter_part_val_indexes_offsets(meta_table, key,
value)
            if result is None:
                result = sub
            continue
```

```
            result = result.intersection(sub)
        return result or set()

    def get_filter_indexes_offsets(
        self,
        meta_table: types.MetaTable,
        filter_: types.Filter,
    ) -> set[int]:
        self.get_filter_keys_for_indexes(meta_table, filter_)
        result = set()
        if isinstance(filter_, list):
            for filter_part in filter_:
                result =
result.union(self.get_filter_part_indexes_offsets(meta_table, filter_part))
        else:
            result = self.get_filter_part_indexes_offsets(meta_table,
filter_)
        return result

    def get_rows_iterator_use_indexes(
        self,
        table_name: str,
        filter_: types.Filter,
    ) -> Generator[types.MetaRow, None, None]:
        meta_table = self.cursor.get_table_by_name(table_name)
        offsets = self.get_filter_indexes_offsets(meta_table, filter_)
        for offset in offsets:
            meta_row = self.cursor.read_row_meta(offset)
            yield meta_row
```

Реализованы методы добавления значения в индексацию (при добавлении новой строки в таблицу). Добавление ключа в индексацию строит индекс на основе всех значений ключа во всех строках таблицы. Метод добавления таблицы в индексацию строит индекс на основе всех ключей таблицы, которые указаны как ключи с индексацией. Метод фильтрации получает на основе фильтра пересечения и объединения оффсетов по соответствующим хеш-значениям и возвращает итератор по соответствующим строкам.

Сравнение результатов получения данных по фильтру с использованием индексов и без

```
$> select -t Cats --counter
-------- select 139661 items
Execution time: 8.552178382873535 s.

$> select -t Cats -f '{age:[1, 2]}' --counter
-------- select 303 items
Execution time: 8.051940441131592 s.
$> select -t Cats -f '{age:[1, 2]}' --counter --use-index
-------- select 303 items
Execution time: 0.03125333786010742 s.
```

Вывод

В процессе выполнения лабораторной работы была реализована однофайловая субд с командами на основе python argsparse, реализован набор автотестов, а также модуль индексации ключей таблицы на основе хеш-алгоритмов.