Pembuatan Database Engine yang Menjaga Integritas Data Antar Mesin

Farhan Dewanta Syahputra - 1313619017

Di bawah bimbingan:
1. Muhammad Eka Suryana, M. Kom.
2. Med Irzal, M. Kom.

Pendahuluan

Latar Belakang

- Penelitian ridho rizqillah pada tahun 2024 menerapkan sistem terdistribusi pada sistem crawling. Database yang digunakan adalah Sqlite.
- MongoDB merupakan database yang mempunyai performa yang lebih baik daripada sqlite.
- Jika ingin menerapkan mongodb pada *public client* ke depannya maka akan mengakibatkan perbedaan database antar *private client* dengan *public client*.
- Database yang dibuat saat ini kebanyakan memilki lisensi sehingga developer tidak punya hak kuasa secara penuh terhadap source code.
- Tercetus pembuatan database engine baru yang proses pengolahan sampai penyimpanannya dapat terlihat dan diubah sesuai dengan kegunaannya.

Rumusan Masalah

Bagaimana pembuatan database engine yang menjaga integritas data antar mesin?

Batasan Masalah

- Pembuatan database engine meliputi penerapan penyimpanan data secara persisten dan menjaga integritas data yang dibuat dengan bahasa pemrograman Rust.
- Database engine yang dibuat belum menggunakan bahasa/query secara langsung sehingga jika ingin mengambil data hanya bisa melalui (interface) yang tersedia.
- Database yang dibuat belum menerapkan foreign key atau relasi antar tabel, sehingga penerapan nanti tidak bisa memiliki constraint antar tabel. Pengambilan data antar tabel hanya sebatas joining tabel.

Tujuan Penelitian

- Membuat database engine yang dapat digunakan secara universal dan memiliki data konsisten
- Membuat database engine yang memiliki proses penyimpanan, pengambilan, pengubahan, penghapusan, joining antar table dan memiliki indexing.
- Membuat database engine yang memiliki kendali penuh mulai dari tahap penyimpanan, pengambilan, pengubahan dan penghapusan. Arti dari kendali penuh yang dimaksud adalah jika terdapat perubahan atau penambahan fitur ke depannya, maka tidak perlu melalui layer tambahan dan dapat ditambahkan ke pemrosesan database secara langsung (seamless).

Manfaat Penelitian

- **Bagi penulis** - Menambah wawasan dan ilmu tentang sistem terdistribusi serta penerapannya, memperoleh gelar sarjana pada bidang ilmu komputer dan mendapatkan pengalaman dalam menulis sebuah jurnal ilmiah.
- **Bagi Program Studi Ilmu Komputer** - Penelitian ini dapat menjadi referensi untuk penelitian yang akan dilakukan mahasiswa ilmu komputer mendatang.
- **Bagi Universitas Negeri Jakarta** - Dapat menjadi bahan evaluasi dan penilaian kualitas akademik di Universitas Negeri Jakarta khusus nya pada program studi Ilmu Komputer

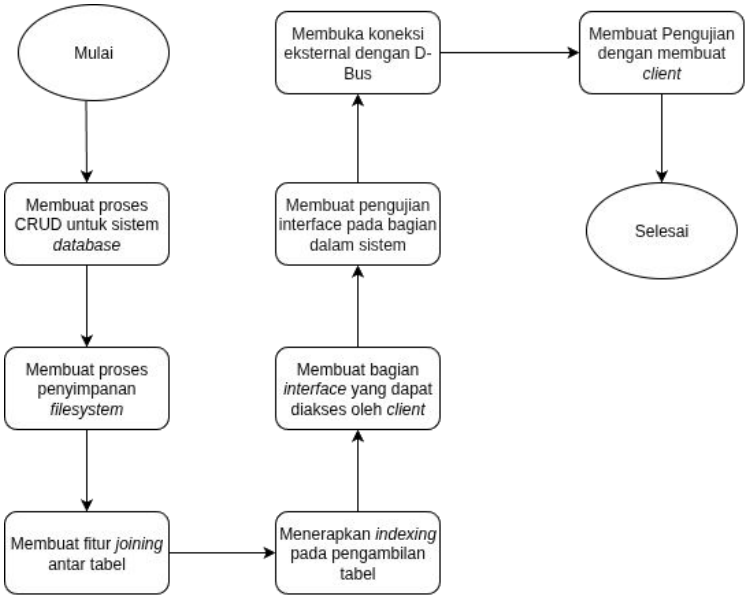
Perubahan Batasan Masalah

Karena batasan masalah pembuatan dan penerapan database engine pada sistem terdistribusi sudah sangat luas, maka batasan masalah diperkecil hingga hanya pembuatan database engine saja.

Untuk batasan masalah Penerapan database pada sistem terdistribusi tidak akan dilanjutkan pada penelitian ini dan akan coba diimplementasikan setelah pembuatan database engine yang akan dikembangkan telah selesai.

Hasil dan Pembahasan

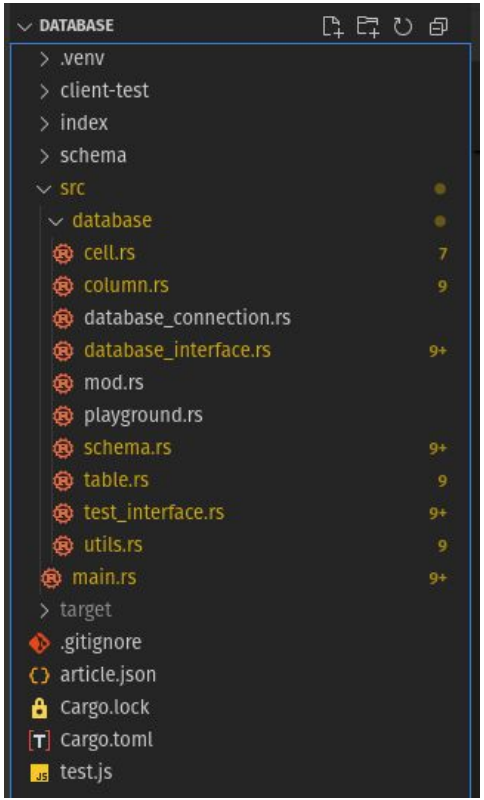
Tahapan Penelitian Penyempurnaan



Bahasa Pemrograman Rust

- Performa yang cepat
- Ketahanan yang baik (Reliability)
- Dokumentasi lengkap
- Kecepatan dapat menyaingi Bahasa C

Struktur Folder



Schema

Schema

+ name: String

+ tables: Vec<Table>

+ index: HashMap<String, Vec<HashMap<String, InputDataEnum>>>

+ to_bytes(): Vec<u8>

+ to_data(buf_u8: &mut Vec<u8>): Schema

+ save(): std::io::Result

+ add_column_to_table(table_name: String, mut column: Column): bool

+ list_column_on_table(name: String): Vec<String>

+ search_table(name: String): &mut Table

+ check_table_index(name: String): isize

+ delete_table(table_name: String):

+ add_data(table_name: String, data: HashMap<String, String>): bool

+ get_data(table_name: String): Vec<HashMap<String, InputDataEnum>>

+ search_data(table_name: String, column_name: String, value: String): Vec<HashMap<String, InputDataEnum>>

+ update_data(table_name: String, where_data: HashMap<String, String>, updated_data: HashMap<String, String>): bool

+ delete_data(table_name: String, where_data: HashMap<String, String>): bool

+ get_join_table(table_name: String, column_name: String, table_join: String, column_join: String, join_type: String): Vec<HashMap<String, InputDataEnum>>

+ join_table(table_name: String, column_name: String, table_join: String, column_join: String): Vec<HashMap<String, InputDataEnum>>

+ join_inner_table(table_name: String, column_name: String, table_join: String, column_join: String): Vec<HashMap<String, InputDataEnum>>

+ build_index():

+ save_index(key: String, result: Vec<HashMap<String, InputDataEnum>>): std::io::Result<()>

+ clear_index(): std::io::Result<()>

+ get_table_list_array_string(): Vec<String>

+ get_file(name: String): File

Alur Penyimpanan Schema

[name_len, name_val, table_len, ... <table.to_bytes> .....]

- Name_len - panjang atribut name
- Name_val - value dari atribut name
- Table_len - panjang data tersimpan pada atribut tables
- Table.to_bytes - memanggil method to.bytes pada setiap data

Table

Table

+ name: String

+ columns: Vec<Column>

+ length: usize

+ get_type(name: String): DataType

+ get_data_by_column(): HashMap<String, Vec<InputDataEnum>>

+ get_column_names(): Vec<String>

+ get_data_by_index(index: usize): HashMap<String, InputDataEnum>

+ search_by_column(column_name: String, search: String): Vec<HashMap<String, InputDataEnum>>

+ search_column(name: String): &Column

+ add_column(column: Column):

+ check_column_index(name: String): isize

+ add_data_column(name: String, input: InputDataEnum):

+ add_data(input_data: HashMap<String, String>):

+ update(index: usize, map: &HashMap<String, String>):

+ delete_column(column_name: String):

+ delete(index: usize):

+ update_data(where_data: HashMap<String, String>, updated_data: HashMap<String, String>): bool

+ delete_data(where_data: HashMap<String, String>): bool

+ to_bytes(): Vec<u8>

+ to_data(buf_u8: &mut Vec<u8>): Table

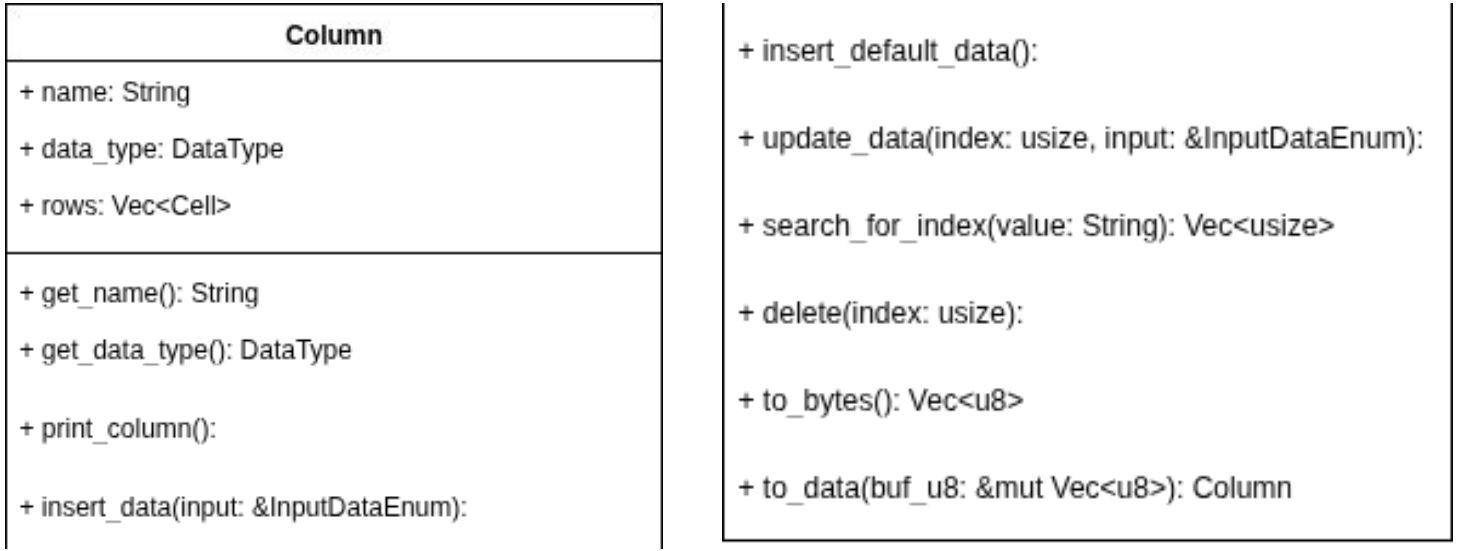
+ name_to_bytes(): Vec<u8>

Alur Penyimpanan Table

[..., name_len, name_val, column_len, data_table_len, ... <column.to_bytes> , ...]

- Name_len - panjang atribut name
- Name_val - value dari atribut name
- column_len - panjang data tersimpan pada atribut columns
- Data_table_len - value dari atribut length
- column.to_bytes - memanggil method to.bytes pada setiap data

Column

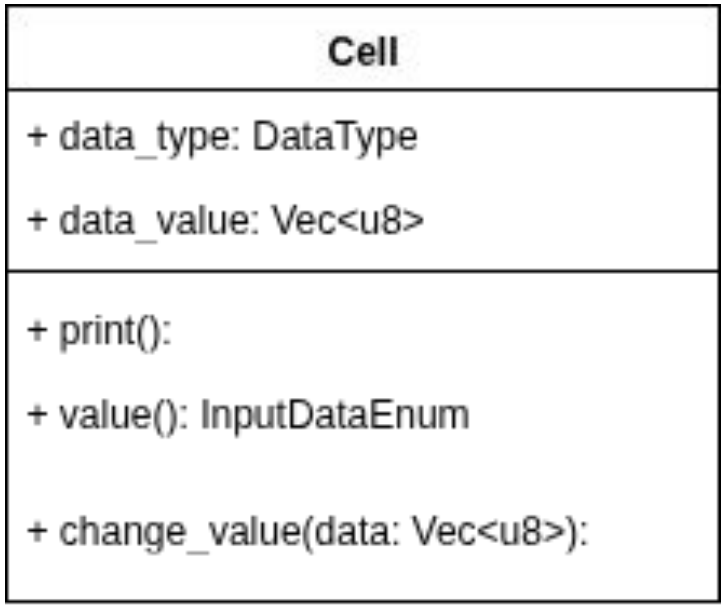

Column
+ name: String
+ data_type: DataType
+ rows: Vec<Cell>
+ get_name(): String
+ get_data_type(): DataType
+ print_column():
+ insert_data(input: &InputDataEnum):
+ insert_default_data():
+ update_data(index: usize, input: &InputDataEnum):
+ search_for_index(value: String): Vec<usize>
+ delete(index: usize):
+ to_bytes(): Vec<u8>
+ to_data(buf_u8: &mut Vec<u8>): Column

Alur Penyimpanan Column

[..., data_column_len, name_len, name_val, data_type, ..., InputDataEnum_value, ...]

- data_column_len - panjang data tersimpan pada atribut rows
- name_len - panjang data dari atribut name
- name_val - value dari atribut name
- data_type - menyimpan tipe data dari kolom
- InputDataEnum_value - atribut data_value pada setiap
- InputDataEnum

Cell


Cell
+ data_type: DataType
+ data_value: Vec<u8>
+ print():
+ value(): InputDataEnum
+ change_value(data: Vec<u8>):

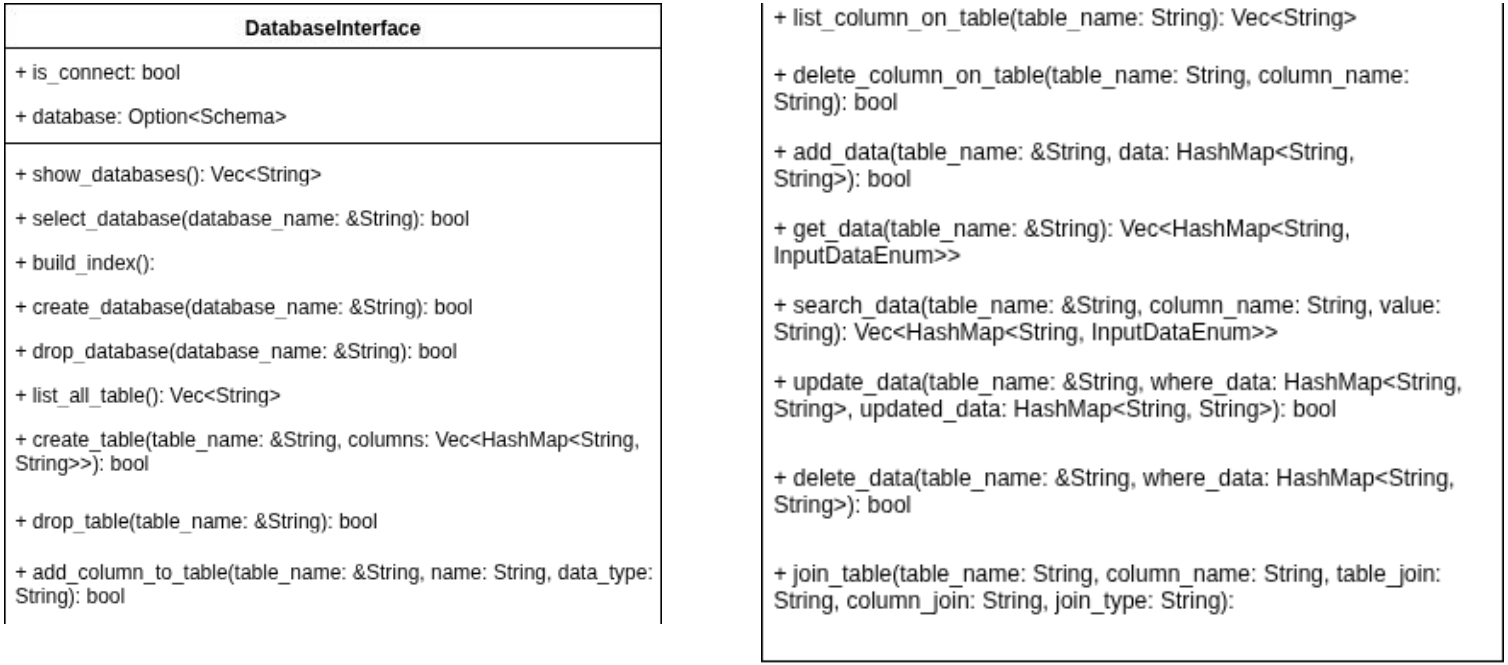
InputDataEnum

```rust
#[derive(Debug, Serialize, Deserialize)]
pub enum InputDataEnum {
    String(String),
    Integer(isize),
    Null,
}
```

DataType

```
1  pub enum DataType {
2      String,
3      Integer,
4      Null,
5  }
```

DatabaseInterface



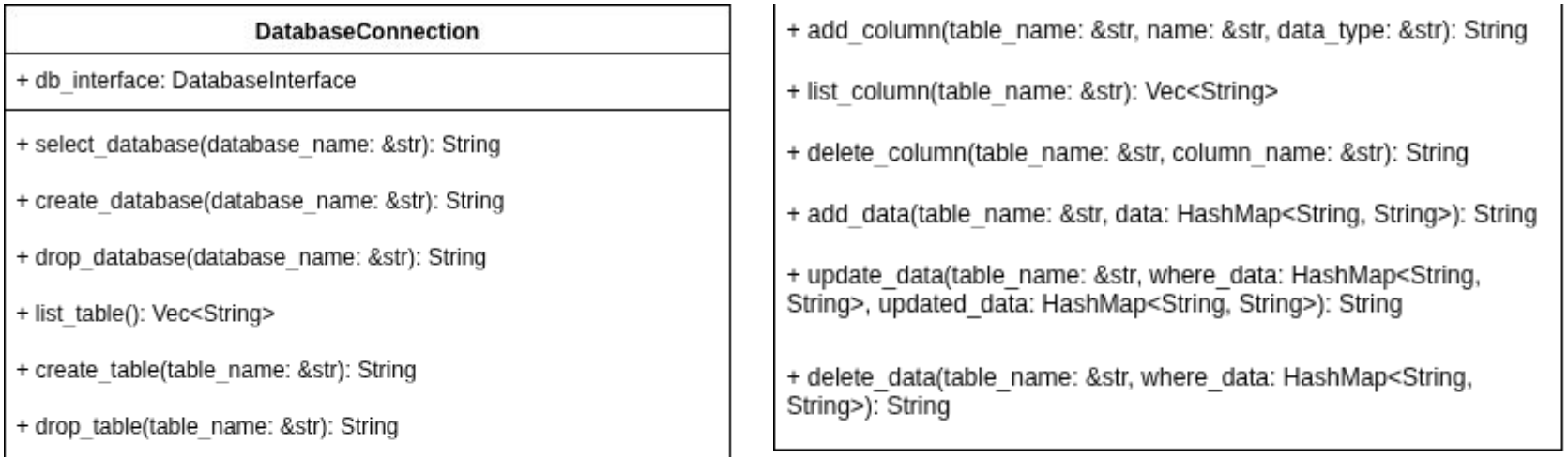
Indexing pada Fitur Join

Penerapan fitur indexing pada database engine ini dilakukan pada fitur join. Hashmap digunakan untuk mengumpulkan index dengan menggunakan key dari parameter yang dijalankan saat hendak menjalankan join.

Namun untuk penerapan indexing saat ini masih belum berjalan dengan baik, Value yang dikembalikan dari HashMap tersebut merupakan sebuah reference dari value yang disimpan. Sementara nilai yang berupa reference tidak bisa di kembalikan atau di return dalam function di bahasa pemrograman Rust.

Maka dari itu fitur join_table bisa berjalan dan mengembalikan data jika fitur indexing ini dimatikan. Jika hanya ingin melihat isi data dari hasil indexing, dapat menggunakan method println! pada rust.

DatabaseConnection



Pengujian

Fase Pengujian dibagi menjadi 2, yaitu:

- Pengujian Internal
- Pengujian Eksernal (Menggunakan Client dari koneksi D-Bus)

Pengujian Internal

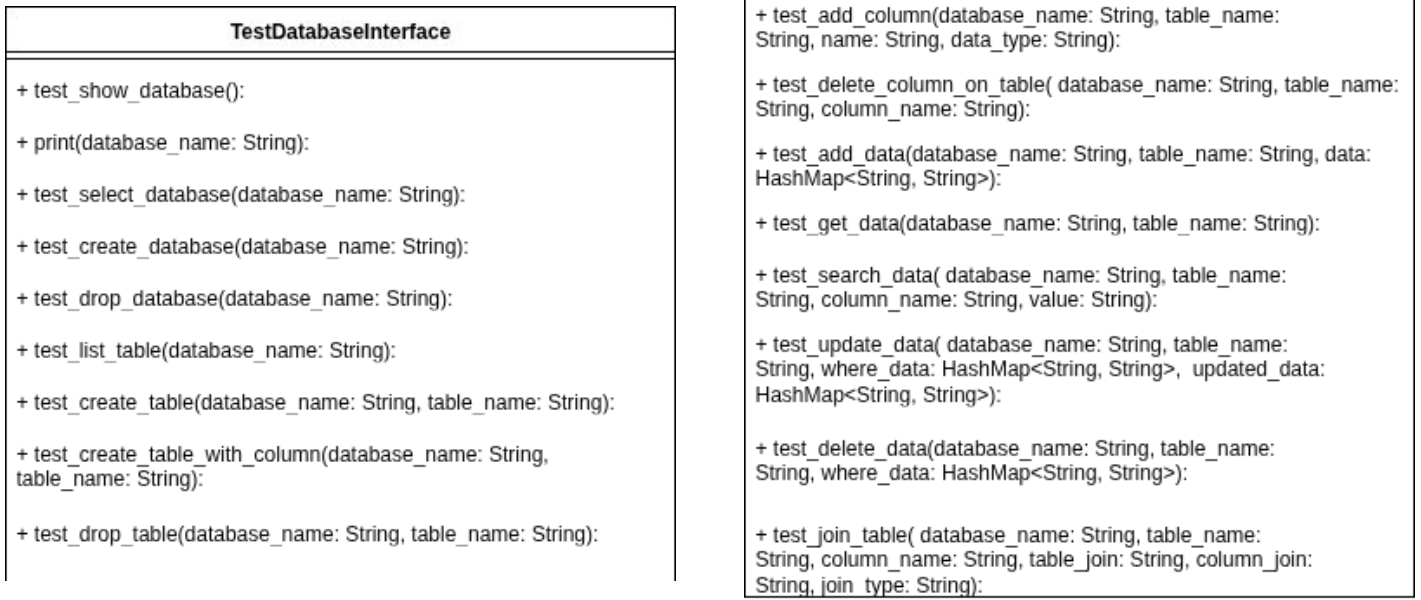
id (Integer)	first_name (String)	Last_name (String)
0	Farhan	Abdul
1	Akbar	Maulana
2	Daffa	Haryadi
3	Hanif	Ramadhan
4	Rudiansyah	Wijaya

Tabel Users untuk data pengujian

id (Integer)	user_id (Integer)	title (String)	description (String)
0	0	Judul 0	*Long Text*
1	0	Judul 1	*Long Text*
2	1	Judul 2	*Long Text*
3	2	Judul 3	*Long Text*
4	2	Judul 4	*Long Text*
5	2	Judul 5	*Long Text*
6	3	Judul 6	*Long Text*
7	2	Judul 7	*Long Text*
8	2	Judul 8	*Long Text*
9	2	Judul 9	*Long Text*
10	1	Judul 10	*Long Text*
11	1	Judul 11	*Long Text*
12	2	Judul 12	*Long Text*

Tabel Posts untuk data pengujian

TestDatabaseInterface



Hasil Pengujian Internal

Pengujian Internal dilakukan dengan menggunakan method print yang ada di dalam masing-masing class di database engine.

Hasil dari pengujian menunjukan interface yang telah dibuat berhasil berjalan sesuai dengan parameter yang diberikan. Namun perlu diperhatikan adalah bagian joining, sesuai dengan penjelasan sebelumnya.

Catatan Tambahan Pengujian Internal

Method DatabaseInterface bernama search_data dan method get_data mengalami perubahan pada bagian return pasca pengujian. Namun perubahan yang dibuat hanyalah mengganti tipe data return dan bukan merubah hal yang signifikan.

Pengujian External

Pengujian eksternal dilakukan dengan membuat *client* dengan bahasa pemrograman python serta library dbus-next. *Client* ini akan memanggil interface database yang dibuka.

Pengujian akan dilakukan pada proses pembuatan, pengubahan dan penghapusan. Untuk pengambilan data masih belum bisa diuji sebagaimana yang telah dikutip pada sub bab 4.1.11 mengenai keterbatasan tipe data.

Pengujian External

```python
from dbus_next.aio import MessageBus

import asyncio

loop = asyncio.get_event_loop()

async def main():
    bus = await MessageBus().connect()
    introspection = await bus.introspect('org.two.DatabaseConnection', '/org/two/DatabaseConnection')
    obj = bus.get_proxy_object('org.two.DatabaseConnection', '/org/two/DatabaseConnection', introspection)
    connection = obj.get_interface('org.two.DatabaseConnection')

    res = await connection.call_create_database("tests") # Membuat database test
    res = await connection.call_select_database("tests") # Select database test
    res = await connection.call_drop_database("tests") # Mencoba menghapus database test

    res = await connection.call_create_database("library") # Membuat database
    res = await connection.call_select_database("library") # Select database
    res = await connection.call_create_table("books") # Membuat table
    res = await connection.call_list_table() # Melihat list table
    res = await connection.call_add_column("books", "id", "integer") # Membuat Kolom baru
    res = await connection.call_add_column("books", "name", "string") # Membuat Kolom baru
    res = await connection.call_add_column("books", "author", "string") # Membuat Kolom baru
    res = await connection.call_delete_column("books", "author") # Membuat kolom author sebagai bahan uji penghapusan kolom
    res = await connection.call_list_column("books") # Melihat list kolom pada tabel book
    res = await connection.call_add_data("books", {'id': "0", "name": "Farhan"}) # Membuat data baru pada tabel books

    print(res)

loop.run_until_complete(main())
```

Code untuk Uji Coba client menggunakan python dengan library dbus-next

Hasil Pengujian External

```
running 1 test
test test ... ok

successes:

---- test stdout ----
Database berhasil terhubung
Database "library":
===Table "books"===
Column: ["id", "name"]
Data 1:
0
"Farhan"




()
```

Hasil Pengujian Pengiriman data dari Client

Hasil Pengujian External

Hasil dari pengujian client dijalankan dengan method print pada class TestDatabaseInterface. Method tersebut merupakan method yang berguna untuk melihat keseluruhan data yang tersimpan pada suatu database.

Hasil Implementasi

- Dari hasil penelitian didapatkan sebuah database engine baru yang dikembangkan dengan bahasa pemrograman Rust.
- Pemrosesan yang terjadi dalam database engine yang telah dibuat dapat dilihat pada setiap method-method yang telah dibuat.
- Pola dan metode penyimpanan pun dapat terlihat dari urutan byte yang disimpan pada file.
- Pengembang ke depannya dapat mengubah dan menyesuaikan pola pemrosesan data di dalam database baru ini jika menemukan pola atau algoritma yang lebih efisien.

Penyempurnaan

- Melengkapi metode-metode eksternal untuk client
- Membuat agar error ditangani dengan baik
- Memperbaiki return indexing pada Join
- Menambahkan utilitas pada fitur join
- Melihat metadata pada column di database
- Menambahkan metadata pada database dan table
- Menambahkan tipe data lain
- Pengubahan default value
- Membuat Multi Connection

Kesimpulan

- Untuk membuat database engine yang menjaga integritas data, maka harus memikirkan bagaimana cara database tersebut disimpan pada filesystem.
- Penyimpanan data pada filesystem harus konsisten agar tidak terdapat data yang hilang.
- Fitur-fitur untuk mengolah data yang disimpan pada filesystem dapat dikelola oleh atribut dan method pada bahasa pemrograman yang digunakan.
- Tipe data yang terdapat pada database harus kompatibel untuk berbagai bahasa untuk memastikan antara server danclientdapat saling tukar menukar data.

Kesimpulan

- Implementasi algoritma sinkronisasi sangat memungkinkan untuk diterapkan di dalam sistem database karena alur awal penerimaan data sampai penyimpanan data dapat dilihat secara langsung.

Saran

- Mencoba memperbaiki kekurangan pada database engine sesuai dengan yang dibahas pada sub bab 4.2.3.
- Mengimplementasikan fitur distribusi database, agar dapat segera digunakan pada sistem crawling untuk search engine.
- Implementasikan lebih dalam fitur-fitur koneksi yang ada pada D-Bus agar komunikasi keclientdapat berjalan lebih baik.
- Sangat disarankan untuk mempelajari konsep dan paradigma yang ada di bahasa pemrograman rust agar dapat mengimplementasikan fitur-fitur lain menjadi lebih baik ke depannya.

Any Questions and Feedback?