

**PERANCANGAN DAN IMPLEMENTASI *PROTOTYPE
DATABASE ENGINE BERBASIS STRUCTURE ORIENTED
PROGRAMMING* MENGGUNAKAN RUST**

Skripsi

**Disusun untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Komputer**



Intelligentia - Dignitas

Oleh:
Farhan Dewanta Syahputra
1313619017

**PROGRAM STUDI ILMU KOMPUTER
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS NEGERI JAKARTA**

2025

LEMBAR PERNYATAAN

Saya menyatakan dengan sesungguhnya bahwa skripsi dengan judul "**Perancangan dan Implementasi Prototype Database Engine Berbasis Structure Oriented Programming Menggunakan Rust**" yang disusun sebagai syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi Ilmu Komputer Universitas Negeri Jakarta adalah karya ilmiah saya dengan arahan dari dosen pembimbing.

Sumber informasi yang diperoleh dari peneliti lain yang telah dipublikasikan yang disebutkan dalam teks Skripsi ini, telah dicantumkan dalam Daftar Pustaka sesuai dengan norma, kaidah dan etika penulisan ilmiah.

Jika dikemudian hari ditemukan sebagian besar skripsi ini bukan hasil karya saya sendiri dalam bagian-bagian tertentu, saya bersedia menerima sanksi pencabutan gelar akademik yang saya sanding dan sanksi-sanksi lainnya sesuai dengan peraturan perundang-undangan yang berlaku.

Jakarta, 12 Agustus 2025

Farhan Dewanta Syahputra

KATA PENGANTAR

Segala puji bagi Allah Subhanahu wa ta'ala Rabb semesta alam. Rasa syukur penulis panjatkan kehadiran Allah Subhanahu wa ta'ala, karena berkat rahmat, hidayah, taufiq serta karunia-Nya penulis dapat menyelesaikan skripsi yang berjudul Perancangan dan Implementasi *Prototype Database Engine Berbasis Structure Oriented Programming* Menggunakan Rust.

Selama penulisan skripsi ini, banyak pihak yang memberikan bantuan dan dukungan untuk penulis agar penulis senantiasa menyelesaikan skripsi ini. Bantuan yang diberikan berupa bantuan secara langsung dan juga moral untuk penulis. Melalui kesempatan ini, penulis ingin berterima kasih kepada orang-orang tersebut, yaitu:

1. Yth. Para petinggi di lingkungan FMIPA Universitas Negeri Jakarta.
2. Yth. Ibu Dr. Ria Arafiyah, M.Si selaku Koordinator Program Studi Ilmu Komputer.
3. Yth. Bapak Muhammad Eka Suryana, M.Kom selaku Dosen Pembimbing I, dengan sabar telah membimbing, membantu, mengarahkan, dan mengoreksi selama proses penulisan skripsi ini berlangsung
4. Yth. Bapak Med Irzal, M.Kom selaku Dosen Pembimbing II dengan sabar telah membimbing, membantu, mengarahkan, dan mengoreksi selama proses penulisan skripsi ini berlangsung
5. Kedua orang tua, abang dan kakak penulis yang senantiasa memberikan dukungan moral serta doa untuk penulis.
6. Teman-teman Warung Tegang yang tidak pernah berhenti mendukung penulis selama proses penulisan skripsi
7. Teman-teman PT Amanah Karya Indonesia yang sering memberikan keringanan dalam bekerja ketika penulis menulis skripsi
8. Teman-teman Program Studi Ilmu Komputer Universita Negeri Jakarta yang telah memberikan dukungan moral dalam penulisan skripsi ini.

Penulis mengakui bahwa skripsi ini masih sangat jauh dari kata sempurna sehingga penulis sangat terbuka terhadap saran dan kritikan yang membangun. Semoga segala hal yang telah penulis buat dalam skripsi ini akan bermanfaat bagi orang lain, karena sebagaimana sabda Rasulullah Sholallahu 'alaihi wa salam bahwa sebaik-baiknya manusia adalah yang paling bermanfaat bagi manusia. Semoga Allah Subhanahu wa ta'ala membala segala kebaikan yang telah diberikan kepada orang-orang yang telah membantu penulis.

Jakarta, 12 Agustus 2025

Farhan Dewanta Syahputra

ABSTRAK

FARHAN DEWANTA SYAHPUTRA. Perancangan dan Implementasi *Prototype Database Engine Berbasis Structure Oriented Programming Menggunakan Rust.* Skripsi. Program Studi Ilmu Komputer. Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Negeri Jakarta. Juli 2025. Di bawah bimbingan Muhammad Eka Suryana, M.Kom dan Med Irzal, M.Kom.

Database menjadi sistem yang sangat penting untuk digunakan di era saat ini. Setiap *database* yang ada memiliki lisensi yang berbeda-beda, mulai dari yang ketat hingga *permissive*. Untuk *database* yang memiliki lisensi ketat, maka pengembang tidak dapat mengendalikan *source code* secara utuh. Sementara, sebagian *database* yang memiliki lisensi *permissive* juga sudah memiliki struktur *code* yang kompleks sehingga akan sulit untuk dikembangkan sesuai dengan kebutuhan. Maka dari itu penelitian pembuatan *database engine* dilakukan dengan tujuan kedepannya para pengembang dapat mengembangkan algoritma dan sistem penyimpanan sesuai dengan yang dibutuhkan, mulai dari lapisan yang sederhana. Pada kasus ini pengembangan ini, kedepannya algoritma sinkronisasi untuk sistem terdistribusi dapat diterapkan pada *database engine*. Pengembangan fitur *Database Engine* akan dilakukan berdasarkan fitur-fitur dasar *database* yang umum untuk digunakan agar dapat memenuhi kebutuhan industri saat ini. Hasil akhir dari penelitian mengenai *database engine* ini adalah semua pemrosesan dari menampilkan, membuat, mengubah, dan menghapus dapat dilihat dengan jelas. Pengembang juga bisa melakukan pengaturan sesuai dengan kebutuhannya. Meski pada hasil akhir penelitian, masih terdapat beberapa fitur yang harus diperbaiki dan disempurnakan.

Kata kunci: *database engine, index, DBMS, sistem terdistribusi*

ABSTRACT

FARHAN DEWANTA SYAHPUTRA. Perancangan dan Implementasi *Prototype Database Engine Berbasis Structure Oriented Programming Menggunakan Rust.* Mini Thesis. Computer Science. Faculty of Mathematics and Natural Sciences, State University of Jakarta. July 2025. Under guidance from Muhammad Eka Suryana, M.Kom and Med Irzal, M.Kom.

Database are becoming an important system in this era. Every database have they own license, starting from strict license to permissive one. For strictly license database, developers cannot have full control to the entire source code itself. Meanwhile, some permissive license database also have a complex code structure which will be difficult to customized based on requirements. Therefore, this research about making database engine are made for the purposes of developers so they can implement their own logic and algorithm based by their own needs, starting from simple layer codebase. For this development case, synchronizationn algorithm for distributed database can be implemented in the future. Database engine feature will be developed based by basic feature of existing database so it will fulfill the industry needs. The result of this database engine research are the entire process of reading, creating, updating and deleting can be seen. Developers also can make their own configuration based on their needs. Even in the final result of this reasearch, there are still much things to fix and develop to make these database better.

Keyword: *database engine, index, DBMS, Distributed System*

DAFTAR ISI

KATA PENGANTAR	iv
ABSTRAK	v
ABSTRACT	vi
DAFTAR ISI	vii
DAFTAR GAMBAR	x
DAFTAR TABEL	xi
I PENDAHULUAN	1
1.1 Latar Belakang Masalah	1
1.2 Rumusan Masalah	5
1.3 Batasan Masalah	6
1.4 Tujuan Penelitian	6
1.5 Manfaat Penelitian	6
II Kajian Pustaka	8
2.1 <i>Database</i>	8
2.2 Fitur-fitur pada <i>Database</i>	9
2.3 Arsitektur Internal <i>Database Management System</i>	11
2.4 <i>Database Index</i>	14
2.5 Data Struktur <i>Map</i>	14
2.6 Relasi antar Tabel (<i>JOIN</i>)	15
2.7 <i>Database</i> Terdistribusi	16
2.8 <i>D-Bus</i>	18
III METODOLOGI PENELITIAN	20
3.1 Tahapan Penelitian	20
3.2 Penerapan Kebutuhan Fitur	22
3.3 Arsitektur Struktur Data	23
3.4 Alur Penyimpanan Data kepada <i>Filesystem</i>	29
3.5 <i>Index</i> Sederhana dengan <i>Map</i>	30
3.6 <i>Join</i> Antar Tabel	31
3.7 Komunikasi Interface <i>Database</i> dengan D-Bus	31
3.8 Alat dan Bahan	32
3.9 Skema Uji	32
IV HASIL DAN PEMBAHASAN	33
4.1 Implementasi	33

4.1.1	Bahasa Pemrograman Rust	33
4.1.2	Struktur Folder	34
4.1.3	Schema	34
4.1.4	Table	39
4.1.5	Column	44
4.1.6	Cell	47
4.1.7	Enumeration	49
4.1.8	Utils	49
4.1.9	DatabaseInterface	50
4.1.10	<i>Indexing</i> pada fitur Join	52
4.1.11	DatabaseConnection	53
4.2	Pengujian	55
4.2.1	Pengujian Internal	55
4.2.2	Pengujian Eksternal	67
4.2.3	Pengujian Internal dengan Data Hasil <i>Crawling</i>	69
4.2.4	Hasil Implementasi	70
4.3	Sumber dan Orisinalitas <i>code</i>	72
V	KESIMPULAN DAN SARAN	73
5.1	Kesimpulan	73
5.2	Saran	74
DAFTAR PUSTAKA		75
LAMPIRAN		77
A	Dokumentasi Source Code	77

DAFTAR GAMBAR

Gambar 1.1	Arsitektur Sistem <i>Crawler</i> Terdistribusi Sebelumnya (Rizqillah 2024)	2
Gambar 1.2	Grafik peningkatan operasi <i>insert</i> pada <i>database</i> berbeda, biasa disebut <i>heterogenous</i> (Győrödi et al., 2023)	4
Gambar 2.1	Ilustrasi Arsitektur <i>Database</i> Umum (Petrox 2019)	12
Gambar 2.2	Ilustrasi <i>Database</i> Tersentralisasi (Watt 2014)	17
Gambar 2.3	Ilustrasi <i>Database</i> Terdistribusi (Watt 2014)	17
Gambar 2.4	Diagram alur D-Bus (freedesktop 2022)	19
Gambar 3.1	Alur tahapan penelitian pembuatan <i>database engine</i>	20
Gambar 3.2	Contoh penerapan <i>struct database</i> pada bahasa C++	24
Gambar 3.3	Proses pengambilan data dari <i>filesystem</i>	30
Gambar 3.4	Proses penyimpanan data ke <i>filesystem</i>	30
Gambar 4.1	Struktur Folder	34
Gambar 4.2	Class Diagram Schema	35
Gambar 4.3	Urutan penyimpanan Schema secara menyeluruh dalam <i>byte</i> .	36
Gambar 4.4	Urutan penyimpanan Schema dalam <i>byte</i>	36
Gambar 4.5	Class Diagram Table	40
Gambar 4.6	Urutan penyimpanan Table dalam byte	43
Gambar 4.7	Class Diagram Column	44
Gambar 4.8	Urutan penyimpanan Column dalam byte	46
Gambar 4.9	Class Diagram Cell	48
Gambar 4.10	Class Diagram <i>DatabaseInterface</i>	51
Gambar 4.11	Class Diagram Column	54
Gambar 4.12	Class Diagram <i>TestDatabaseInterface</i>	57
Gambar 4.13	File baru telah terbuat dalam <i>folder index</i> dan schema . . .	58
Gambar 4.14	Informasi <i>database</i> telah terhubung	58
Gambar 4.15	Informasi tabel telah berhasil terbuat	59
Gambar 4.16	Informasi kolom telah berhasil terbuat	59
Gambar 4.17	Informasi data telah berhasil terbuat	60
Gambar 4.18	Hasil data tabel <i>posts</i> telah berhasil terbuat	60
Gambar 4.19	Hasil data tabel <i>posts</i> yang telah diformat menggunakan <i>prettier</i> (<i>Extension Visual Studio Code</i> pada Javascript) . . .	61
Gambar 4.20	Hasil data tabel <i>users</i> telah berhasil terbuat	61
Gambar 4.21	Hasil data tabel <i>users</i> yang telah diformat menggunakan <i>prettier</i> (<i>Extension Visual Studio Code</i> pada Javascript) . . .	62
Gambar 4.22	Informasi data berhasil diubah	62
Gambar 4.23	Informasi hasil pencarian	63
Gambar 4.24	Hasil data yang telah diformat menggunakan <i>prettier</i> (<i>Extension VSCode</i> pada Javascript)	63

Gambar 4.25 Informasi hasil <i>inner join</i>	63
Gambar 4.26 Hasil data yang telah diformat menggunakan <i>prettier</i> (<i>Extension VSCode</i> pada Javascript)	64
Gambar 4.27 Informasi hasil <i>left join</i>	64
Gambar 4.28 Hasil data yang telah diformat menggunakan <i>prettier</i> (<i>Extension VSCode</i> pada Javascript)	65
Gambar 4.29 Informasi hasil <i>right join</i>	65
Gambar 4.30 Hasil data yang telah diformat menggunakan <i>prettier</i> (<i>Extension VSCode</i> pada Javascript)	66
Gambar 4.31 Informasi hasil penghapusan data	66
Gambar 4.32 Data tabel <i>users</i> setelah value "Rudiansyah" dihapus	67
Gambar 4.33 Data tabel <i>users</i> yang telah diformat menggunakan <i>prettier</i> (<i>Extension Visual Studio Code</i> pada Javascript)	67
Gambar 4.34 Isi program pengujian <i>client</i> dengan bahasa pemrograman Python	68
Gambar 4.35 Informasi hasil pengujian <i>client</i>	68
Gambar 4.36 Informasi hasil pengujian data <i>crawling</i>	70
Gambar 1.1 <i>Import</i> dalam file main.rs	77
Gambar 1.2 <i>Function</i> main dalam file main.rs	78
Gambar 1.3 <i>Function</i> test dalam file main.rs bagian 1	79
Gambar 1.4 <i>Function</i> test dalam file main.rs bagian 2	80
Gambar 1.5 <i>Function</i> test_data_crawling dalam file main.rs. Keseluruhan <i>function</i> dapat dilihat di <i>source code</i> langsung	81
Gambar 1.6 File cell.rs	82
Gambar 1.7 File column.rs bagian 1	83
Gambar 1.8 File column.rs bagian 2	84
Gambar 1.9 File table.rs bagian 1	85
Gambar 1.10 File table.rs bagian 2	86
Gambar 1.11 File table.rs bagian 3	87
Gambar 1.12 File table.rs bagian 4	88
Gambar 1.13 File table.rs bagian 5	89
Gambar 1.14 File schema.rs bagian 1	90
Gambar 1.15 File schema.rs bagian 2	91
Gambar 1.16 File schema.rs bagian 3	92
Gambar 1.17 File schema.rs bagian 4	93
Gambar 1.18 File schema.rs bagian 5	94
Gambar 1.19 File database_connection.rs bagian 1	95
Gambar 1.20 File database_connection.rs bagian 2	96
Gambar 1.21 File database_interface.rs bagian 1	97
Gambar 1.22 File database_interface.rs bagian 2	98
Gambar 1.23 File database_interface.rs bagian 3	99
Gambar 1.24 File database_interface.rs bagian 4	100

Gambar 1.25 <i>File</i> mod	101
Gambar 1.26 <i>File</i> test_interface.rs bagian 1	102
Gambar 1.27 <i>File</i> test_interface.rs bagian 2	103
Gambar 1.28 <i>File</i> test_interface.rs bagian 3	104
Gambar 1.29 <i>File</i> test_interface.rs bagian 4	105
Gambar 1.30 <i>File</i> test_interface.rs bagian 5	106
Gambar 1.31 <i>File</i> test_interface.rs bagian 6	107
Gambar 1.32 <i>File</i> test_interface.rs bagian 7	107
Gambar 1.33 <i>File</i> utils.rs	108

DAFTAR TABEL

Tabel 3.1	Tabel linimasa perkiraan awal dan estimasi penggeraan	22
Tabel 4.1	Tabel Users	55
Tabel 4.2	Tabel Posts	56

BAB I

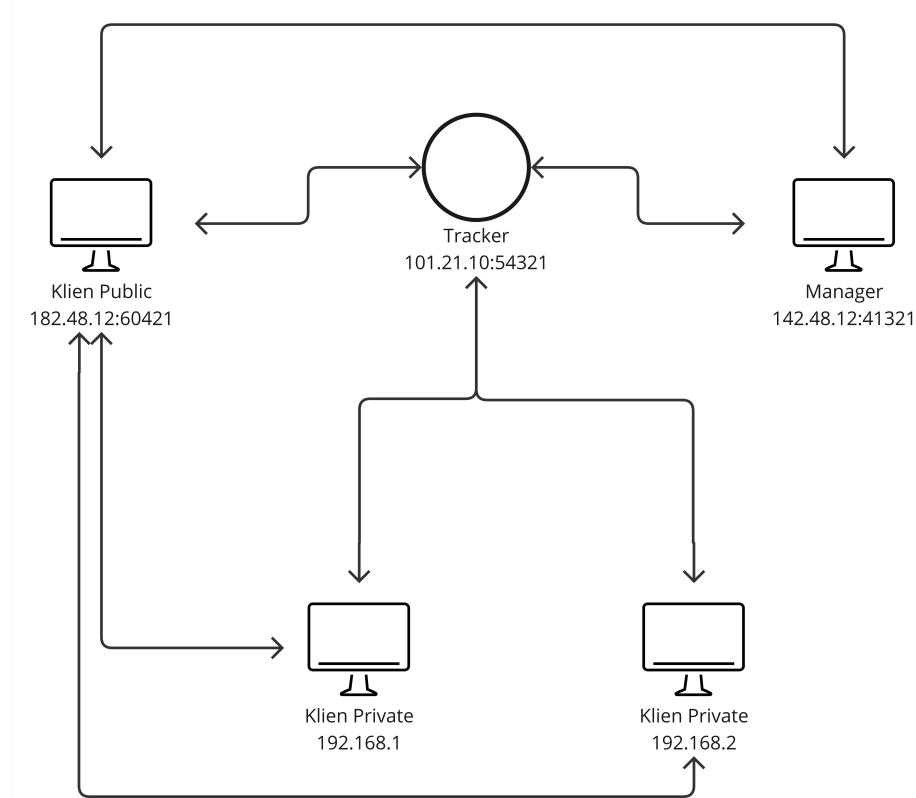
PENDAHULUAN

1.1 Latar Belakang Masalah

Perkembangan teknologi pada bidang informasi pada era ini sangatlah pesat. Hampir semua kalangan masyarakat dapat menggali informasi di mana pun dan kapanpun selagi terhubung dengan koneksi internet. Salah satu perkembangan teknologi di bidang informasi yang paling berperan adalah *search engine*. Dengan *search engine*, pengguna dapat mencari informasi yang diinginkan dengan cara mengetikkan *keyword* dari informasi yang dicari. *Search engine* akan menampilkan semua informasi yang relevan berdasarkan *website* yang mengandung *keyword* dari yang pengguna cari. Setelah menampilkan *website* yang relevan, pengguna dapat melihat lebih detail terkait informasi yang dicari dengan mengunjungi *website* tersebut. Selain memiliki manfaat bagi pengguna, *search engine* juga bermanfaat bagi pemilik *website* karena dapat menambahkan jumlah pengunjung *website* setiap saat tanpa perlu melakukan promosi ataupun iklan.

Terdapat penelitian yang berkaitan dengan *search engine* yang belum lama telah dibuat, di antaranya adalah penelitian mengenai *crawler* berjalan secara *multi-threaded* (Qoriiba 2021). Penelitian ini dilanjutkan dengan melakukan *refactoring* pada *crawler* dan menambahkan pencarian *similarity score* dari halaman *web* hasil *crawling* (Khatulistiwa 2023). Lalu penelitian ini dilanjutkan dengan mengembangkan *crawling engine* yang berjalan secara terdistribusi yang akan mengimplementasikan penggunaan *socket* dalam mendistribusikan tugas ke setiap *slave machines* (Rizqillah 2024). *Crawling* terdistribusi diterapkan dengan melakukan pemecahan *crawling* terhadap mesin yang berbeda. Hal ini berguna untuk meringankan beban kerja yang dijalankan pada masing-masing mesin. Selain meringankan beban kerja, mesin yang berjalan juga akan menghasilkan *crawling* dalam jumlah masif yang lebih efisien.

Meninjau hasil akhir dari percobaan penelitian sebelumnya (Rizqillah 2024), terdapat kelemahan yang harus diperbaiki pada penyimpanan data. Berikut adalah gambar arsitektur sistem terdistribusi yang dibuat oleh penelitian sebelumnya (Rizqillah 2024).



Gambar 1.1: Arsitektur Sistem *Crawler* Terdistribusi Sebelumnya (Rizqillah 2024)

Pada arsitektur tersebut *database* disimpan secara terpusat pada 1 *public client* walaupun masing-masing *private client* juga menyimpan data. Akan lebih baik jika penyimpanan dilakukan secara terdistribusi daripada harus terpusat. Berdasarkan *Paper An Overview of Distributed Databases* (Tomar et al., 2014) dinyatakan bahwa salah satu manfaat dari penerapan *database* terdistribusi ini adalah *High Performance-Queries and updates are largely local so that there is no network bottleneck*. Berdasarkan pernyataan tersebut didapatkan bahwa *database* terdistribusi membuat performa *query database* terutama *query* yang jumlahnya cukup besar akan menjadi lebih efektif serta akan terhindar dari *network bottleneck* karena operasi penyimpanan atau pengubahan dilakukan secara lokal. Terdapat manfaat lain yang dinyatakan di *paper* tersebut di antaranya:

1. *Robust* (Kokoh), jika satu bagian mengalami masalah maka tidak akan menghentikan atau mengganggu bagian lainnya.
2. *Security*, akses pekerja dapat dibatasi sesuai dengan porsi *database* mereka

masing-masing.

3. Arus jaringan berkurang sehingga membuat *bandwidth cost* berkurang.
4. *Database* lokal masih dapat bekerja meskipun jaringan perusahaan sedang rusak atau kendala.
5. Dalam sistem terdistribusi akan lebih mudah untuk membuat sebuah *error* tidak menyebar ke seluruh sistem.

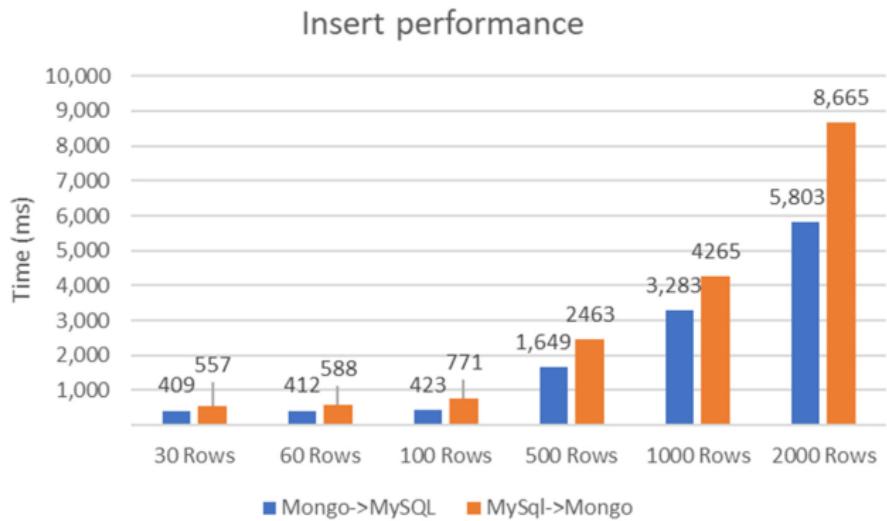
Dalam sistem tersebut, pengguna juga hanya bisa memiliki 1 *public client* sementara jika mencoba untuk menambah 1 *public client* lagi, maka *public client* tersebut akan berperan layaknya *private client* yang menjalankan *crawling*. Cara lain untuk menambah *public client* agar *database* tidak menjadi terpusat adalah dengan menjalankan sistem yang sama pada mesin yang berbeda, sehingga dalam sistem tersebut dapat berisi 2 *public client*, 2 manajer dan 2 *tracker*.

Namun jika ingin menerapkan cara tersebut terdapat sebuah permasalahan pada penyimpanan data antar mesin dan sinkronisasinya. Sinkronisasi yang berjalan pada sistem saat ini hanya diterapkan pada *public* dan *private client*. Sementara antara sebuah *public client* dengan *public client* yang lain masih belum bisa melakukan sinkronisasi. Karena belum bisa melakukan sinkronisasi antara *public client* maka akan sangat rentan terjadinya duplikasi data di penyimpanan *database*.

Pada *public client*, penggunaan MongoDB sebagai *database* utama akan digunakan. Berdasarkan *Paper A Comparative Study: MongoDB vs. MySQL* (Gyorodi et al., 2015) dikatakan bahwa "*MongoDB provided lower execution times than MySQL in all four basic operations, which is essential when an application should provide support to thousands of users simultaneously. Thus, the above comparison tests, proves that for large amounts of data MongoDB has a good performance and it is preferred than MySQL*". Dari pernyataan tersebut disimpulkan bahwa MongoDB mempunyai performa yang lebih baik dibanding MySQL yang mempunyai basis *Relational Database*. Namun terdapat permasalahan baru, karena *Private Client* pada arsitektur sebelumnya (Rizqillah 2024) menggunakan sqlite yang membuat dalam sistem terdapat 2 jenis *database* berbeda antara *public client* dan *private client*

Penggunaan 2 jenis *database* yang berbeda juga akan membutuhkan lapisan sinkronisasi untuk membuat *database* yang terhindar dari duplikasi data. Berdasarkan *paper Implementing a Synchronization Method between a Relational and a Non-Relational Database* (Győrődi et al., 2023) didapatkan bahwa

sinkronisasi antar *database* pasti akan menimbulkan *delay*. Waktu delay yang dialami juga mengalami peningkatan secara eksponensial. Peningkatan signifikan ini akan terus meningkat dengan seiring dengan bertambahnya jumlah data. Berikut adalah gambar *chart* yang didapat dari paper tersebut.



Gambar 1.2: Grafik peningkatan operasi *insert* pada *database* berbeda, biasa disebut *heterogenous* (Győrödi et al., 2023)

Waktu *delay* antara sinkronisasi *database* dapat diatasi dengan melakukan efisiensi terhadap algoritma sinkronisasi. Salah satu algoritma sinkronisasi yang dapat dicoba diterapkan adalah arsitektur dan algoritma sinkronisasi yang berada pada *paper Design and Implementation of a Heterogeneous Relational Database Synchronization Mechanism Based on Tree Distribution Architecture* (Xu et al., 2019). Untuk menerapkan algoritma sinkronisasi tersebut juga akan membutuhkan lapisan sinkronisasi tambahan dikarenakan sqlite dan mongoDB merupakan entitas *database* yang berbeda. Lapisan tambahan ini bisa dihilangkan dengan memasukkan lapisan sinkronisasi tersebut sebagai fitur *native* dari MongoDB dan SQLite. Namun *source code* MongoDB memiliki lisensi yang memiliki aturan-aturan tertentu ketika pengembang melakukan memodifikasi dan mengubah *source code*. Aturan-aturan pada lisensi membuat pengembang tidak bisa memiliki kendali penuh terhadap *source code* sistem tersebut. Permasalahan ini yang memicu timbulnya ide untuk membuat *database* baru yang dapat lebih adaptif terhadap sistem *crawling* yang berjalan saat ini. Hal ini dikarenakan pengembang dapat memegang kendali penuh mulai dari proses pengambilan, penyimpanan, pengubahan dan penghapusan data.

Algoritma sinkronisasi antar *database* pun juga dapat diterapkan langsung pada *database* tanpa melalui perantara lapisan sinkronisasi.

Jika meninjau *database* selain MongoDB yang memiliki lisensi lebih aman seperti Postgresql terdapat permasalahan baru yaitu kompleksitas dari *source code*. Kompleksitas ini berbanding lurus dengan banyaknya fitur postgresql, di dalam *source code* tersebut terdapat banyak sekali *file* serta *function* yang dibuat. Oleh karena itu, untuk mencoba mengadaptasi *source code* dari postgresql dan mengubah sesuai dengan kebutuhan akan sulit dilakukan dengan kemampuan penulis saat ini. Selain itu, *database engine* yang telah dibuat biasanya telah memiliki alur lapisan pemrogamannya tersendiri. Sementara dengan membuat *database engine* tersendiri, pengembang dapat menyesuaikan alur serta algoritma yang sesuai dengan sistem *crawling* pada penelitian sebelumnya (Rizqillah 2024)

Lalu manfaat lain dari pembuatan *database* baru ini adalah pengembang dapat menyesuaikan skema distribusi *database* yang paling cocok dengan sistem *crawling* yang ada saat ini (Rizqillah 2024). Salah satu konsep distribusi *database* yang dapat digunakan adalah konsep distribusi yang ada pada *paper* berjudul *Research on The Improvement of MongoDB Auto-Sharding in Cloud Environment* (Liu et al., 2012). *Paper* ini membahas tentang arsitektur yang berjalan pada MongoDB. Konsep distribusi dilakukan dengan memecah kumpulan-kumpulan data menjadi potongan-potongan kecil. Potongan tersebut dapat didistribusikan antar mesin agar setiap mesin bertanggung jawab pada subset dari total data set yang ada. Penggunaan *database* pada sistem terdistribusi akan diterapkan pada sistem *crawling* ini berdasarkan saran dari penelitian sebelumnya (Rizqillah 2024). Akan tetapi penerapan *database* pada sistem terdistribusi baru akan dilaksanakan setelah mesin dan fitur utama dari *database* baru ini selesai.

1.2 Rumusan Masalah

Berdasarkan dari rincian dan penjelasan latar belakang di atas, maka didapatkan rumusan masalah "**Bagaimana Perancangan dan Implementasi Prototype Database Engine Berbasis Structure Oriented Programming Menggunakan Rust?**".

1.3 Batasan Masalah

Rumusan masalah yang didapat memiliki permasalahan yang cukup banyak dan luas sehingga harus diperjelas batasannya. Batasan Masalah pada penelitian yang dilakukan antara lain:

1. Pembuatan *database engine* meliputi penerapan penyimpanan data secara persisten dan menjaga integritas data yang dibuat dengan bahasa pemrograman Rust.
2. *Database engine* yang dibuat belum menggunakan bahasa/*query* secara langsung sehingga jika ingin mengambil data hanya bisa melalui (interface) yang tersedia.
3. *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.

1.4 Tujuan Penelitian

1. Membuat *database engine* yang dapat digunakan secara *universal* (Dapat digunakan pada sistem lain) dan memiliki data konsisten
2. Membuat *database engine* yang memiliki proses penyimpanan, pengambilan, pengubahan, penghapusan, *joining* antar table dan memiliki *indexing*.
3. 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*).

1.5 Manfaat Penelitian

1. 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.

2. Bagi Program Studi Ilmu Komputer

Penelitian ini dapat menjadi referensi untuk penelitian yang akan dilakukan mahasiswa ilmu komputer mendatang.

3. Bagi Universitas Negeri Jakarta

Dapat menjadi bahan evaluasi dan penilaian kualitas akademik di Universitas Negeri Jakarta khusus nya pada program studi Ilmu Komputer.

BAB II

Kajian Pustaka

2.1 Database

Pada Era teknologi saat ini, terdapat banyak sekali informasi-informasi yang tersebar. Mulai dari informasi mengenai pendidikan, penelitian, ekonomi dan lainnya. Informasi yang didapat bisa dikatakan sebagai data. Data dapat berbentuk angka, kumpulan kata, symbol dan lainnya. Data-data ini harus disimpan supaya bisa diolah atau disebarluaskan. Untuk tercapai hal nya tersebut dibutuhkan sebuah teknologi untuk menyimpan data-data tersebut. Teknologi tersebut adalah *database*. *Database* adalah sebuah kumpulan data yang digunakan untuk mendukung aktivitas dari individu ataupun kelompok (Watt 2014). *Database* dapat berisi sebuah tabel, sebagai contoh sebuah *database* perpustakaan dapat berisi tabel buku dan tabel pengunjung di mana kedua tabel tersebut dapat berkaitan satu sama lain.

Database juga dapat didefinisikan sebagai kumpulan data yang berkaitan di mana pengguna dapat mengambil data tersebut secara efisien (Limited 2008). Data-data yang telah dikumpulkan tersebut sangat memungkinkan untuk diolah, diambil, dan diubah. Untuk melakukan hal tersebut dibutuhkan sebuah sistem yaitu *database management system* yang disingkat sebagai DBMS. DBMS adalah sebuah program yang membuat pengguna dapat mengelola *database* dan mengontrolnya (Watt 2014). Tujuan utama dari DBMS adalah menyediakan aksesibilitas bagi pengguna agar dapat mengolah dan mengontrol data dengan baik. Di sisi lain DBMS juga harus untuk memastikan keamanan dari data yang telah disimpan dari pihak luar atau pihak tidak dikenal.

Berdasarkan buku *database system* (Limited 2008), DBMS merupakan manajemen data tersentralisasi. Terdapat beberapa kelebihan jika data disimpan secara tersentralisasi, kelebihan inilah yang juga menutupi kekurangan metode penyimpanan data sebelumnya yaitu *file-based*. Berikut ini adalah beberapa kelebihan yang telah disebutkan pada buku tersebut (Limited 2008):

1. Kontrol Redudansi Data

Database didesain untuk mengurangi redudansi data dikarenakan sebagian besar data yang disimpan didalamnya terintegrasi dengan data yang ada di tempat lain. Kondisi ini meminimalisir terjadinya replikasi data pada tempat

lain dan juga memastikan konsistensi pada penyimpanannya. Walaupun dalam kondisi mungkin saja terjadi duplikasi data untuk mengoptimalkan performa saat menggunakan *database*.

2. Integritas Data

Pada penggunaan *database*, penerapan integritas data menjadi lebih mudah. Saat proses melakukan desain, pengguna dapat menentukan hubungan dan relasi antar data pada *database* yang akan dibuat. Beberapa penerapan integritas data dapat dibuat langsung di *database* secara otomatis atau dapat melalui aplikasi yang menggunakan data tersebut.

3. Aksesibilitas

Data yang disimpan di *database* dapat diakses oleh beberapa pengguna ataupun program. Dengan kasus seperti ini, bisa saja terdapat dua aplikasi yang berbeda namun menggunakan sumber data yang sama.

4. Kemudahan dalam Pengembangan

Dengan menggunakan DBMS, permasalahan seperti keamanan, konkurensi, integritas data dan lainnya sudah ditangani oleh DBMS itu sendiri. Pengembang aplikasi menjadi lebih mudah karena tidak perlu memikirkan hal tersebut.

5. Keamanan

Karena penyimpanan data secara terpusat, membuat penerapan keamanan menjadi lebih mudah. DBMS dapat memastikan bahwa pengolahan data hanya bisa terjadi pada akses yang diizinkan. Pada umumnya DBMS menyediakan sebuah code atau password bagi pengguna yang ingin mengakses data.

2.2 Fitur-fitur pada *Database*

Berdasarkan buku *database design* (Watt 2014), *database* (dalam pembahasan ini adalah *relational database*) dapat memiliki banyak tabel. Setiap tabel tersebut juga dapat memiliki banyak kolom yang menyimpan data sesuai dengan kebutuhan dari pengguna. Setiap kolom yang ada di dalam tabel memiliki bisa memiliki tipe data yang berbeda-beda. Selain kolom, dalam *database* memiliki baris data, dan nilai dari

baris data tersebut akan berkaitan dengan kolom yang telah dibuat pada tabel. Tipe data dari baris juga harus sama dengan tipe data dari kolom.

Pengguna sering sekali membutuhkan akses kepada data yang telah tersimpan tersebut seperti melakukan pembuatan, pengambilan data untuk melakukan analisa dan lainnya. Seperti yang dikutip pada subbab sebelumnya bahwa untuk melakukan pengolahan database, pengguna dapat menggunakan DBMS. Salah satu model DBMS yang sering digunakan adalah *relational DBMS* dan *SQL (Structured Query Language)* sebagai bahasa standar yang digunakan untuk mengakses DBMS. Berdasarkan buku *database design* (Watt 2014), SQL digunakan dalam DBMS untuk melakukan berbagai hal seperti:

1. Membuat *database* dan tabel
2. Melakukan pengolahan data yang umum digunakan seperti (membuat, menghapus, dan mengubah)
3. Melakukan pengambilan data untuk mengubah data mentah menjadi data yang dapat terbaca dengan baik

Untuk melakukan pembuatan tabel, pembuatan *database* harus terlebih dahulu dilakukan. Tabel akan memiliki sebuah nama, dan dalam tabel tersebut pengguna juga bisa membuat kolom yang banyak. Kolom tersebut berisi nama dari kolom, tipe data dari kolom dan beberapa konfigurasi tambahan lainnya. Nama yang diberikan pada kolom harus bersifat unik dan tidak boleh sama antara satu dengan yang lainnya. Untuk tipe data, SQL memiliki beberapa jenis tipe data yang didukung, beberapa di antaranya yaitu:

1. Bit - Sebuah data integer bernilai 1 atau 0.
2. Int - Data integer dari -2.147.483.648 hingga 2.147.483.647
3. Smallint - Data integer dari -32.768 hingga 32.767
4. Tinyint - Data integer dari 0 hingga 255
5. Uniqueidentifier - *Global Unique Identifier (GUID)*
6. Float - Tipe data yang mengambil data secara presisi
7. Varchar - non-unicode karakter data yang berjumlah maksimal 8.000 karakter

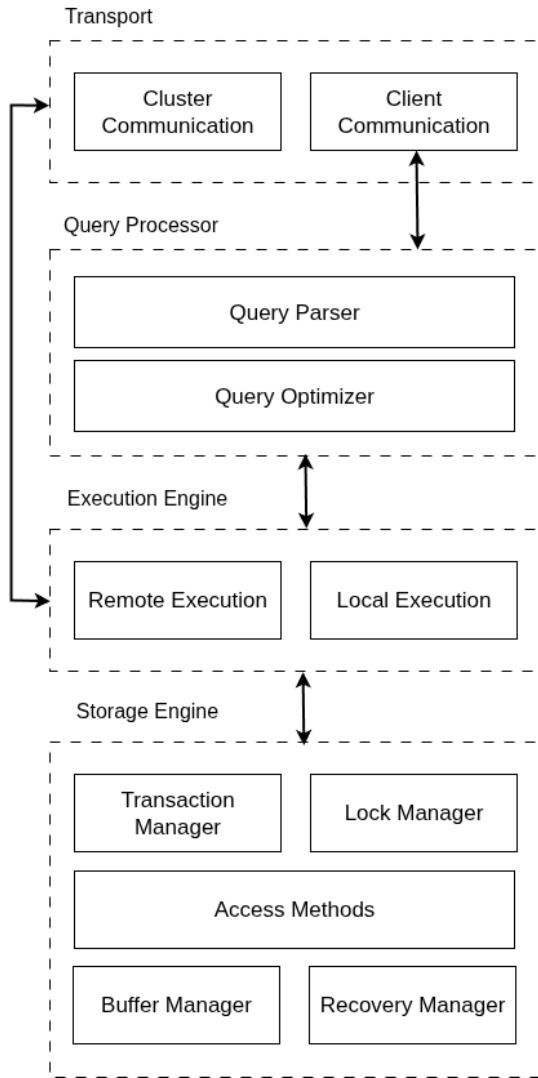
8. Text - non-unicode karakter data yang berjumlah maksimal 2.147.483.647 karakter

Pada SQL, pengguna juga dapat menghapus kolom, menambahkan kolom, menghapus tabel dan beberapa konfigurasi lainnya. SQL juga mendukung operator *join* yang berguna untuk menghubungkan kedua tabel atau lebih. Pembahasan mengenai *join* tabel akan dibahas pada subbab 2.6.

Di dalam DBMS umumnya juga memiliki fitur *index* yang berguna untuk melakukan pengambilan data secara efisien. Berdasarkan buku *Database System* (Garcia-Molina et al., 2008), *index* adalah sebuah struktur data yang disimpan dengan tujuan untuk mempercepat akses terhadap sebuah data yang telah tersimpan. Pembahasan lebih rinci mengenai *index* akan dibahas pada subbab 2.4.

2.3 Arsitektur Internal *Database Management System*

Berdasarkan buku *Database Internal* (Petrox 2019), arsitektur yang diterapkan dalam pengembangan *database management system*, tidak mempunyai standar penerapan yang pasti. Setiap *database* dapat dibuat dengan arsitektur yang berbeda dan arsitektur yang diterapkan pun juga sulit dilihat dan ditentukan. Meski arsitektur *database* ini dijelaskan pada sebuah tulisan seperti dokumentasi, dalam penerapan dan penulisan *code* aslinya, arsitektur yang berbeda mungkin saja diterapkan untuk optimalisasi performa, penanganan beberapa kasus, atau keputusan arsitektur. Berikut ini adalah gambaran besar arsitektur DBMS yang sering digunakan.



Gambar 2.1: Ilustrasi Arsitektur Database Umum (Petrox 2019)

Database management system menggunakan *client/server model*, di mana sebuah sistem *database* berperan sebagai server, dan aplikasi mengambil peran sebagai *client*. Permintaan *client* akan diterima melalui *transport subsystem*. Permintaan datang dalam bentuk sebuah *query*, dan sering sekali dikatakan sebagai *query language*. *Transport subsystem* juga bertugas untuk melakukan komunikasi dengan node yang lain dalam cluster *database*. Setelah menerima permintaan, *transport subsystem* menyerahkan *query* kepada *query processor* yang akan menguraikan (*parse*), menafsirkan (*interprets*) dan melakukan validasi terhadap *query* tersebut. Setelah semua proses ini selesai, pemeriksaan akses kontrol akan dilaksanakan.

Query yang telah diuraikan akan diberikan kepada *query optimizer*, yang akan menghilangkan bagian yang tidak memungkinkan untuk dijalankan dan bagian yang redundant dari *query*. Lalu, sistem akan mencoba untuk mencari cara yang paling efisien untuk mengeksekusi *query*-nya berdasarkan statistik *internal* (kardinalitas *index*, perkiraan ukuran *intersection* dan lainnya) dan penempatan data (*node* yang menyimpan data dalam *cluster* dan biaya yang berkaitan dengan perpindahannya). *Optimizer* menangani 2 hal yaitu, operasi yang dibutuhkan untuk resolusi pada *query* (biasa dipresentasikan sebagai *dependency tree*) dan optimisasi seperti pengurutan *index*, estimasi kardinalitas dan pemilihan metode akses.

Query biasanya dibuat dalam bentuk *execution plan* (atau bisa disebut *query plan*). *Execution plan* adalah barisan / kumpulan operasi yang harus dibawa untuk memastikan hasil dianggap selesai. Karena *query* yang sama dapat membuat hasil yang sama dengan *execution plan* yang berbeda berdasarkan tingkatan efisiensinya, *optimizer* akan mengambil *execution plan* yang paling terbaik. *Execution plan* akan diproses oleh *execution engine* yang akan mengagregat hasil operasi dari lokal dan *remote*. *Remote execution* dapat meliputi proses writing (membuat, modifikasi, menghapus, dan lainnya) dan reading (mengakses) data dari node lainnya dalam cluster. *Remote execution* juga dapat meliputi replikasi. Sementara *local queries* (yang datang langsung dari *client* atau *node* lainnya) akan diproses oleh *storage engine*. Sebuah *Storage Engine* memiliki beberapa komponen yang masing-masing komponen tersebut memiliki tugasnya masing-masing, di antaranya adalah sebagai berikut:

1. *Transaction Manager*

Komponen pengelolaan ini menjadwalkan *transactions* dan memastikan bahwa *transactions* tidak dapat selesai dengan kondisi *state* didalamnya tidak konsisten.

2. *Lock manager*

Komponen ini akan mengunci obyek *database* untuk menjalankan *transactions*, untuk memastikan operasi konkurensi tidak merusak integritas data.

3. *Access Methods*

Komponen ini membantu pengelolaan akses dan mengorganisir data dalam *disk*. *Access methods* meliputi *heap files* dan struktur penyimpanan seperti *B-Trees* atau *LSM Trees*.

4. *Buffer Manager*

Komponen pengelolaan ini akan melakukan *caching* data halaman-halaman dalam *memory*.

5. *Recovery Manager*

Komponen ini mengelola *log* operasi dan merestorasikan sistem jika sistem mengalami kegagalan.

Secara bersama, *transaction* dan *lock managers* bertanggung jawab untuk pengendalian konkurensi. Keduanya menjamin integritas data logis dan fisik sambil memastikan bahwa operasi konkurensi dilaksanakan se efisien mungkin. Teori-teori yang dijabarkan diatas ditulis berdasarkan informasi yang didapatkan dari buku (Petrox 2019).

2.4 *Database Index*

Berdasarkan buku *Introduction to Database System* (Limited 2008), setelah data berhasil disimpan di *disk* menggunakan metode-metode pengolahan *file*, terdapat sebuah permasalahan utama yaitu bagaimana cara untuk melakukan pengambilan data dengan cepat pada permintaan atau *query* yang berbeda. Permasalahan ini dapat diselesaikan dengan menggunakan sebuah struktur tambahan yang bernama *index*. Sebuah *file* dapat dibuat untuk menerapkan penambahan struktur *index* ini. Penerapan *index* tidak akan memengaruhi peletakan data yang telah disimpan sebelumnya, namun dapat memengaruhi kecepatan dalam proses pengambilan datanya. Umumnya dalam suatu *database* dapat lebih dari 1.

Berdasarkan *Practical SQL, 2nd Edition* (DeBarros 2022), *index* pada *database* berfungsi layaknya *index* yang sering ditemukan pada buku. *Query* dapat dipercepat dengan menambahkan sebuah *index*, *index* disini adalah sebuah struktur data terpisah yang dikelola oleh sistem *database*. Penerapan *index* dalam *database* biasanya di terapkan pada kolom yang tersedia pada *database*. *Database* akan menggunakan *index* sebagai jalur cepat dalam melakukan pengambilan data daripada melakukan pengambilan data dengan cara normal (melakukan pemindaian baris per baris).

2.5 *Data Struktur Map*

Berdasarkan buku *Data Structures and Algorithms in Java, 6th Edition* (Michael T. Goodrich 2014), Map adalah sebuah abstraksi data yang didesain untuk

melakukan pengambilan dan penyimpanan data secara efektif. Map melakukan penyimpanan dan pengambilan data dengan mendefinisikan sebuah *key* yang bersifat unik untuk setiap nilai yang disimpan. *Key* yang bersifat unik ini digunakan untuk melakukan pengambilan data sehingga jika ingin mengambil sebuah data yang tersimpan maka harus mengetahui *key* yang menjadi pasangan dari nilai tersebut.

Berdasarkan informasi dari buku *A Common-Sense Guide to Data Structures and Algorithms, Second Edition, 2nd Edition* (Wengrow 2020), hampir semua bahasa pemrograman memiliki fitur map ini. Hanya saja istilah yang digunakan pada masing-masing bahasa pemrograman berbeda-beda seperti *Hash Table*, *Hash Maps*, *associative array* dan *dictionary*. Walaupun istilah yang digunakan berbeda, namun yang tetap menjadi kelebihan adalah satu yaitu kecepatan untuk pengambilan data. Untuk mencari sebuah value dalam struktur data map mempunyai tingkat efisiensi sekitar O(1) berdasarkan rata-rata. Namun yang harus dijadikan catatan adalah untuk mencari value dalam map, pengguna harus mengetahui dulu *key* dari value tersebut. Jika tidak, maka pengguna harus melakukan pengecekan pada setiap *key* dan *value* dari *map* tersebut. Hal ini menyebabkan tingkat efisiensi meningkat yang awalnya O(1) menjadi O(N)

2.6 Relasi antar Tabel (*JOIN*)

Pada jenis-jenis *database*, terdapat salah satu tipe *database* yang sering digunakan yaitu *Relational Database*. Berdasarkan buku *Practical SQL, 2nd edition* (DeBarros 2022), *Relational Database* menyimpan data pada berbagai macam tabel yang saling berkaitan. Tabel yang saling berhubungan akan menyimpan sebuah data tambahan yang bertujuan untuk menghubungkan tabel tersebut ke tabel lain. Untuk menggabungkan data antar tabel tersebut membutuhkan sebuah proses yang bernama *join*. Pada *relational database* seperti MySQL, terdapat beberapa metode dalam melakukan *join* (DeBarros 2022), yaitu:

1. *JOIN / INNER JOIN*

Dengan menggunakan metode *join* ini, data yang ditampilkan hanyalah data yang memiliki hubungan satu sama lain. Sehingga data yang tidak punya hubungan ke tabel lain, maka tidak akan ditampilkan.

2. *LEFT JOIN*

Menampilkan seluruh baris data dari tabel sebelah kiri, dan ikut menampilkan

data dari tabel sebelah kanan jika terdapat data tambahan pada tabel sebelah kiri yang memiliki hubungan ke tabel sebelah kanan.

3. *RIGHT JOIN*

Prosesnya mirip dengan *LEFT JOIN* akan tetapi berkebalikan. Menampilkan seluruh baris data dari tabel sebelah kanan, dan ikut menampilkan data dari tabel sebelah kiri jika terdapat data tambahan pada tabel sebelah kanan yang memiliki hubungan ke tabel sebelah kiri.

4. *FULL OUTER JOIN*

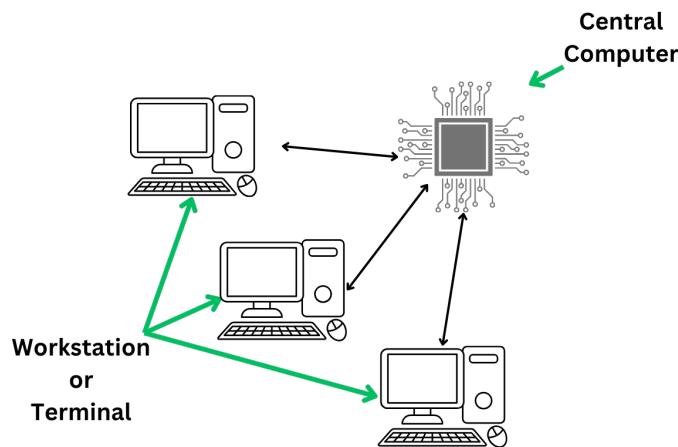
Menampilkan gabungan semua data dari tabel kiri dan tabel kanan. Akan tetapi jika dalam suatu baris terdapat data tambahan pada masing-masing tabel yang menghubungkan kedua tabel, maka baris dari kedua tabel tersebut akan bergabung menjadi satu. Untuk proses *Full Outer Join*, jika dari masing-masing tabel tidak ada data tambahan yang menghubungkan antar tabel satu sama lain, maka proses *join* ini tidak akan menampilkan apa-apa.

5. *CROSS JOIN*

Menampilkan segala jenis kemungkinan penggabungan antar tabel tanpa memperhatikan data tambahan pada masing-masing tabel.

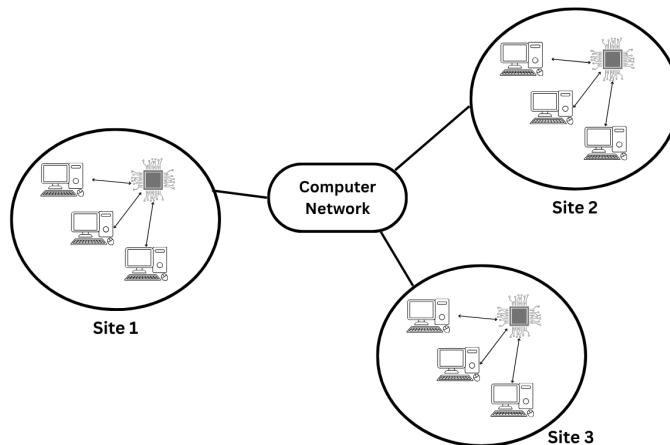
2.7 *Database Terdistribusi*

Berdasarkan buku *database design*(Watt 2014), dilihat dari jumlah penyebaran dan penggunaan sebuah *database*, terdapat 2 cara penggunaan *database* yaitu *database* tersentralisasi dan *database* terdistribusi. *Database* tersentralisasi berjalan pada satu buah mesin. *Database* dan DBMS berjalan pada sistem yang sama. Pengguna dapat berinteraksi dengan *database* tersebut melalui koneksi yang telah terhubung pada mesin. Berikut adalah ilustrasi penerapan *database* tersentralisasi:



Gambar 2.2: Ilustrasi *Database* Tersentralisasi (Watt 2014)

Sementara dalam *database* terdistribusi, *database* dan DBMS tersebar ke beberapa mesin yang berbeda. Antar mesin berkomunikasi satu sama lainnya melalui sebuah jaringan yang memiliki kecepatan tinggi. Berikut adalah ilustrasi dari penerapan *database* terdistribusi:



Gambar 2.3: Ilustrasi *Database* Terdistribusi (Watt 2014)

Database terdistribusi dapat diklasifikasikan menjadi 2 yaitu *homogeneous* dan *heterogenous*. *Homogeneous* menggunakan DBMS yang sama antar mesin yang terhubung. Dengan begitu perpindahan dan pengolahan data antar mesin menjadi lebih mudah. Misalnya sebuah mesin A dan mesin B yang masing-masing didalamnya terdapat satu DBMS yang sama. Cara pengambilan data pada kedua mesin tersebut akan sama, sehingga ketika melakukan komunikasi, setiap mesin

tidak perlu melakukan interpretasi kembali. Berikutnya terdapat *heterogenous* yang pada penerapan *database* terdistribusi, memungkinkan untuk memiliki DBMS yang berbeda antar mesin. Dengan penerapan ini, pengguna harus memastikan bahwa kedua DBMS yang berbeda tersebut harus bisa saling berkomunikasi. Salah satu penerapannya, beberapa *database* menggunakan format *machine-readable cataloguing* (MARC) yang sama untuk mendukung pertukaran data.

2.8 D-Bus

Berdasarkan dokumentasi dari freedesktop.org (freedesktop 2022), D-bus adalah sebuah *message bus system*, sebuah cara sederhana untuk aplikasi berkomunikasi dengan aplikasi lainnya. D-Bus memiliki beberapa lapisan:

1. *libdbus*

Merupakan sebuah *library* yang berguna untuk menghubungkan kedua aplikasi satu sama lain dan bertukar *messages*

2. *Message Bus Daemon Executable*

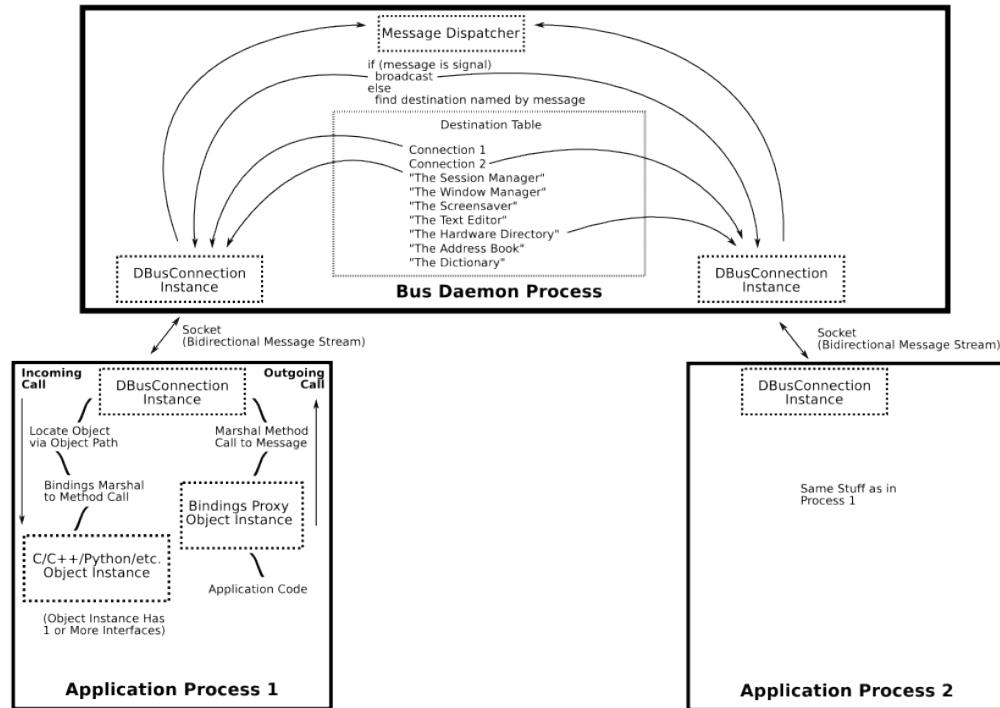
Dibuat dengan *libdbus*, dan beberapa aplikasi dapat terhubung ke dalamnya. Daemon dapat menyalurkan *messages* dari sebuah aplikasi ke beberapa aplikasi lainnya.

3. *Wrapper Libraries* atau *Binding Based*

Wrapper yang digunakan berdasarkan *framework* atau bahasa pemrograman yang digunakan. Contoh dari *wrapper libraries* ini adalah *libdbus-glib* dan *libdbus-qt*. Masih terdapat hal lain seperti *binding* dengan bahasa pemrograman. *Wrapper libraries* ini digunakan untuk mempermudah penggunaan D-bus. *Library libdbus* sendiri memang ditujukan untuk melakukan *binding* dengan *higher level bindings*. Kebanyakan penggunaan API D-Bus dipergunakan untuk penerapan *binding*.

libdbus hanya mendukung koneksi *one to one*. Paket pengiriman yang dikirim bukanlah *bytes*, namun dikenal dengan istilah *messages*. *Messages* mempunyai *header* yang berguna untuk mengidentifikasi jenis dari *message* nya, dan *body messages* berisi sebuah data atau dikenal dengan *payload*. *Library libdbus* juga akan menangani autentikasi.

Untuk bagian *message bus daemon*, berperan layaknya sebuah pusat pengiriman *messages*. *Bus Daemon* akan menerima *messages* dari sebuah aplikasi dan meneruskan *messages* tersebut ke aplikasi yang dituju. *Message bus daemon* dapat juga disebut sebagai *router*. Umumnya pada setiap komputer memiliki beberapa *message bus daemon*. Berikut adalah diagram proses bekerja D-Bus.

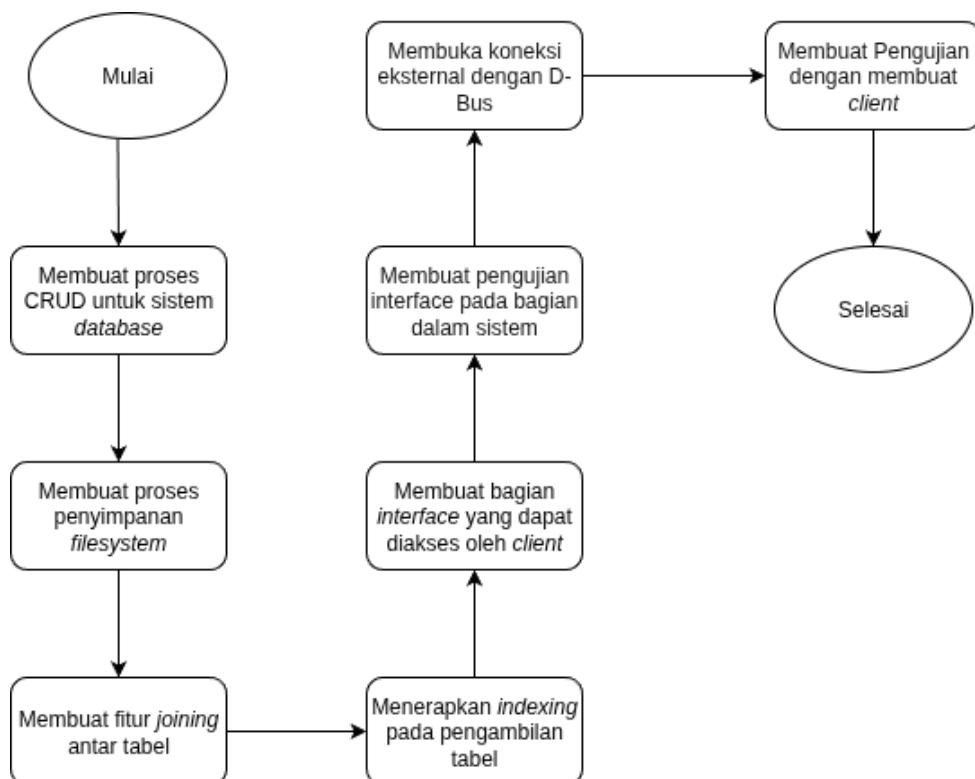


Gambar 2.4: Diagram alur D-Bus (freedesktop 2022)

BAB III

METODOLOGI PENELITIAN

3.1 Tahapan Penelitian



Gambar 3.1: Alur tahapan penelitian pembuatan *database engine*

Penelitian dimulai dengan mencoba menerapkan fungsional dasar (CRUD) pada *database* yaitu proses pengambilan, penambahan, pengubahan dan penghapusan data. Pengembangan fungsi-fungsi dasar akan melibatkan berbagai macam struktur data. Langkah pertama dari belum bisa memiliki data yang persisten karena data yang diolah pada langkah ini belum terhubung ke *filesystem*.

Langkah berikutnya yaitu penerapan penyimpanan secara persisten di *filesystem*. Meneruskan proses-proses data yang telah masuk di langkah pertama dan membuat data tersimpan pada *filesystem*. Setelah penyimpanan berhasil dibuat, dilanjutkan dengan proses pembuatan fitur *joining* antar tabel. Proses *joining* ini akan melakukan penggabungan data dari 2 tabel yang terpisah. Setelah berhasil

melakukan *join* tabel, pengembangan fitur *indexing* akan dilakukan. Fitur *indexing* akan berjalan ketika hendak melakukan pengambilan data pada saat melakukan *join* tabel. Dengan memanfaatkan fitur *indexing*, penerapan fitur *joining* tabel akan dilakukan agar proses penggabungan tabel menjadi lebih cepat.

Setelah fungsi-fungsi berhasil dibuat, maka diperlukan sebuah *interface* untuk memisahkan fungsi mana saja yang dapat diakses oleh pengguna dan fungsi mana yang hanya bisa diakses oleh sistem. Fungsi pada *Interface* ini yang nanti akan dibuka dan dapat diakses oleh pengguna. Lalu untuk memastikan fungsi-fungsi yang telah dibuat berjalan dengan baik, dibuatlah *interface* untuk melakukan pengujian secara internal. Interface pengujian internal dibuat berdasarkan fungsi pada interface yang telah dibuat sebelumnya (*Interface* untuk fungsi pengguna). Maksud dari internal disini adalah proses pengujian dilakukan dengan menggunakan bahasa pemrograman yang sama yaitu rust, dan fungsi di panggil secara langsung tanpa melalui koneksi external seperti D-bus.

Setelah pembuatan *interface* pengujian internal, pembuatan interface koneksi D-Bus dibuat, disusul dengan pembuatan pengujian eksternal menggunakan bahasa pemrograman lain untuk memastikan koneksi D-Bus berjalan dengan baik. Berikut adalah linimasa dari tahapan penelitian. Linimasa ini hanyalah perkiraan awal dan estimasi dari waktu pengembangan *database engine*.

No.	Tahapan	Estimasi Waktu
1.	Membuat proses <i>CRUD</i> untuk <i>interface database</i>	3 pekan
2.	Membuat Proses Penyimpanan <i>Filesystem</i>	1 bulan
3.	Membuat fitur <i>joining</i> tabel	2 pekan
4.	Menerapkan <i>indexing</i>	2 pekan
5.	Membuat <i>interface</i> yang dapat diakses pengguna	1 Pekan
6.	Membuat pengujian untuk <i>interface</i> pada bagian dalam sistem	1 Pekan
7.	Membuka koneksi external dengan D-Bus	1 Pekan
8.	Membuat pengujian eksternal dengan membuat <i>client</i>	1 Pekan

Tabel 3.1: Tabel linimasa perkiraan awal dan estimasi penggeraan

3.2 Penerapan Kebutuhan Fitur

Berdasarkan subbab 2.2, *database* memiliki beberapa fitur umum yang dapat berfungsi untuk mengolah data. Untuk proses pengembangan *database engine*, fitur-fitur umum yang ada pada *database* akan dikembangkan. Beberapa fitur tersebut di antaranya adalah:

1. Kemampuan untuk Membaca data (*Read*)
Fitur ini berguna untuk melakukan pengambilan data yang berasal dari sebuah *file* yang telah disimpan pada *filesystem*.
2. Kemampuan untuk melakukan proses penyimpanan dan mengolah data secara persisten (*Create, Update dan Delete*)
Proses penyimpanan dan hasil pengolahan data akan disimpan secara persisten pada *filesystem* dengan menggunakan format tersendiri.
3. Penggunaan *Index*
Penerapan *indexing* sederhana akan diterapkan terhadap data yang ada dalam *database*

4. Penggabungan 2 tabel (*Join*)

Kapabilitas untuk melakukan *joining* 2 tabel terhadap data yang telah disimpan

5. Koneksi servis *Client* via *D-Bus*

Fitur untuk menerima proses permintaan dari *client* untuk menjalankan fungsi-fungsi yang ada dalam *database*.

3.3 Arsitektur Struktur Data

Sistem *database* terdistribusi dilakukan dengan melakukan pemecahan penyimpanan *database* pada mesin yang berbeda. Setiap mesin *database* yang terpisah dapat berkomunikasi satu sama lain namun tidak secara langsung. Dibutuhkan 2 mesin tambahan yang berguna untuk menghubungkan *database* terpisah tersebut. Berikut adalah ilustrasi dari arsitektur yang akan diterapkan:

```

1 #include <string>
2 #include <vector>
3 #include <ctime>
4 #include <map>
5
6 using namespace std;
7
8 enum dataType{
9     INT,
10    STRING,
11    DOUBLE,
12    BOOL
13 };
14
15 template<typename T>
16 struct value{
17     T val;
18 };
19
20 struct rows{
21     // value akan memiliki tipe data dinamis
22     vector<value> values;
23     int unique_id;
24     time_t created_at;
25     time_t updated_at;
26 };
27
28 struct metadata{
29     bool nullable;
30     int max_length;
31     dataType type;
32     bool unique;
33     map<string, int> unique_index;
34 };
35
36 struct table{
37     string name;
38     int primary_key;
39     int unique_key_counter = 0;
40     vector<string> columns;
41     vector<metadata> metadatas;
42     map<int, rows> data;
43 };
44
45 struct result
46 {
47     vector<string> columns;
48     vector<rows> data;
49 };
50
51 struct column{
52     string name;
53     metadata meta;
54 };
55
56 struct database {
57     string name;
58     vector<table> tables;
59     bool is_distributed = false;
60     bool is_connected = false;
61     string manager_ip;
62 };

```

Gambar 3.2: Contoh penerapan *struct database* pada bahasa C++

Terdapat beberapa atribut dalam *class* tabel di antaranya adalah *name* yang akan menampung nama dari tabel, *primary_key* yang akan menampung kolom mana

yang akan menjadi primary key, column yang merupakan array *string* berisi urutan nama kolom pada tabel, lalu metadata merupakan array dari *class* metadata yang berisi metadata berkaitan dengan kolom yang ada sesuai dengan urutan dan terakhir terdapat atribut data. Atribut data ini akan berisi *hashmap* dengan key unique id dengan value *class rows*. Value unique id berasal dari atribut *unique_key_counter* yang memiliki default value 0 dan akan terus bertambah ketika terdapat *row* baru yang masuk ke dalam *hashmap*.

Pada penerapan *class rows*, terdapat atribut *unique_id*, *created_at* dan *updated_at*. Untuk atribut *unique_id* akan berisi sebuah integer yang berasal dari key penyimpanan *rows* di *class* tabel. Atribut *created_at* dan *updated_at* akan memiliki *value* epoch timestamp yang berasal dari waktu data dibuat dan waktu data diubah. Atribut *created_at* juga memiliki peran pada saat melakukan penerapan sinkronisasi. Lalu pada bagian *row* memiliki array dari *class values*. *Class* ini memiliki *interface Get Value* yang berguna untuk mengembalikan *value* generik yang ada dalam *class values*. Penerapan seperti ini dilakukan agar *value* yang disimpan dapat dinamis.

Class column yang dibuat pada gambar, berguna ketika menampilkan semua column beserta metadata nya. Terdapat *class* metadata yang akan menyimpan informasi terkait column yang ada seperti tipe kolom, *nullable* dan lainnya. Di dalam *class* metadata terdapat sebuah *hashmap* yang akan digunakan sebagai *index*. *Index* ini berfungsi untuk mempercepat pencarian terhadap sebuah *value* sehingga akan digunakan ketika column memiliki tipe unik dan ketika hendak melakukan sinkronisasi data ketika menerapkan sistem terdistribusi. *Hashmap* akan memiliki *key value pair* dengan *key* yang berisi *value* dari kolom yang di *index* sementara untuk *value* dari *hashmapnya* adalah unique id yang berasal dari *key* penyimpanan *rows* pada *class table*. Enum *dataType* yang ada pada metadata berguna untuk membatasi tipe data yang akan dipakai pada column *database*.

Terakhir terdapat *database class* atau *struct* yang merupakan *parent* dari semua *class*. *Database class* mempunyai atribut *table* yang merupakan array dari *table Class*. *Database class* juga mempunyai atribut *name* yang dibutuhkan ketika saat mendeklarasikan *class* tersebut. Dalam *class* yang dibuat akan mengimplementasikan *interface* nya masing-masing. *Interface* ini akan berisi metode atau *function* yang berguna untuk menjalankan operasi dalam *database*.

Berikut ini adalah isi metode atau *function* yang ada dalam *interface* class *database*:

1. *Get Table*

Berguna untuk mengambil *instance class table* yang berada dalam *class database*. *Method* ini menerima 1 *parameter* berupa *string* yang isinya adalah nama table dari list *instance* yang ada dalam *database*. Hasil dari *return method* ini adalah *instance table* yang mempunyai nama yang sama dengan *value* yang ada di *parameter*.

2. *Create Table*

Method ini berguna untuk menambahkan *class table* baru ke dalam *database*. Menerima 1 *parameter* berupa *class table* yang nantinya *instance* tersebut akan dimasukkan ke dalam list *class table* pada *database*. Hasil dari *return method* ini adalah *class table* yang sudah berhasil dibuat.

3. *Delete Table*

Method ini berguna untuk menghapus tabel yang ada pada *class database*. Menerima 1 *parameter* berupa *string* yang isinya adalah nama *table* yang ingin dihapus dalam *database*.

4. *Get Database Name*

Method ini berfungsi untuk mengambil nama dari *database*, tidak akan menerima *parameter* apapun dan akan mengembalikan return sebuah *string* nama dari *database*.

5. *Update Database Name*

Method ini berfungsi untuk Mengganti nama dari *database*, akan menerima 1 *parameter* berupa *string* yang isinya adalah nama *database* terbaru yang ingin diubah.

6. *Read File*

Method ini berfungsi untuk membaca data yang sudah disimpan secara persisten pada *virtual file system*. Attribute list *table* nantinya akan diisi melalui *method Read File* ini.

7. *Write File*

Berfungsi untuk menyimpan data secara persisten dan menuliskannya ke *file* yang telah ditentukan.

8. *Create File*

Berfungsi untuk membuat *space* baru ketika *database* ingin dibuat. *Method* ini dijalankan hanya jika belum ada *space* untuk *database*.

9. *Set Distributed*

Berfungsi untuk memberikan informasi kepada *database* untuk berjalan secara terdistribusi. *Method* ini menerima 2 *parameter*, *parameter* pertama adalah boolean untuk menentukan apakah berjalan terdistribusi atau tidak. Lalu untuk *parameter* kedua berisi IP dari manajer untuk menerapkan sistem terdistribusi.

Dalam *database* nantinya akan terdapat list *class table*. *Class table* juga akan memiliki *method* yang berguna untuk melakukan pengolahan data. Berikut ini adalah *interface* yang ada pada *class table*:

1. *Get Data*

Berguna untuk mengambil data yang tersedia dalam *class table*. *Method* ini akan memiliki dua *parameter*, *parameter* *hashmap* yang di dalamnya terdapat key dan value yang sama dengan kolom pada table. Hal ini berfungsi untuk mencari value yang tersedia pada data yang ada dalam *class table*. *Parameter* kedua akan berperan sebagai konfigurasi pada query seperti mengatur urutan atau melakukan limitasi.

Dalam *method* ini terdapat proses pengambilan data yang diambil dari storage persisten. Hasil akhir dari *method* ini berupa *class result* yang berisi atribut column dan row tanpa metadata dengan ketentuan yang diberikan pada *parameter*.

2. *Show Data*

Method Show Data berguna untuk mengambil satu buah data yang sesuai dengan id yang diberikan. *Method* ini akan mengembalikan *class result*.

3. *Create Data*

Method yang berguna untuk membuat data / row baru pada *class table*. *Method* ini menerima 1 *parameter*, yaitu *hashmap* yang berisi *key* dan *value* yang sesuai dengan kolom yang ada pada table yang dituju. Untuk *hashmap* yang disediakan harus menyesuaikan dengan metadata yang dibuat pada *class table*. Jika terdapat kolom yang tidak *nullable*, maka *value* dari kolom tersebut

harus ada pada *hashmap* yang disediakan. Hasil Return dari *method* ini adalah *class result* yang sudah berisi data-data yang diminta untuk dibuat.

Method ini akan mengirimkan *event* ke manajer sebelum melakukan penyimpanan jika *database* berjalan secara terdistribusi. Lalu juga akan melakukan perubahan pada *unique_index* pada *class* metadata jika kolom mempunyai *value* yang unik.

4. *Update Data*

Method update ini berguna untuk mengubah sebuah data yang ada di dalam *class table* dengan ketentuan yang telah diberikan. *Method* ini akan menerima 2 *parameter*, *parameter* pertamanya adalah *hashmap* yang nantinya digunakan untuk mencari data yang sesuai. Lalu untuk *parameter* kedua juga akan berisi *hashmap* yang terdiri dari *key* dan *value* yang ingin diubah. Hasil Return dari *method* ini adalah *class result* yang sudah berisi data terbaru yang telah diubah. Jika terdapat column yang memiliki *index*, maka *value index* tersebut juga akan di *update*. Termasuk jika menerapkan *database* terdistribusi, maka akan mengirimkan *event* ke manajer untuk memastikan data yang di-*update* tidak duplikasi dengan data di mesin lain.

5. *Delete Data*

Berguna untuk menghapus data yang ada dalam *class table*. *Method* ini menerima 1 *parameter*, *parameter* pertamanya adalah *hashmap* yang akan menentukan data mana yang akan di hapus. Isi dari *hashmap* harus sesuai dengan kolom yang tersedia pada *class table*. *Method* ini akan mengembalikan status apakah dia berhasil menghapus data atau tidak. Jika terdapat kolom yang mempunyai *index*, maka *index* tersebut juga akan dihapus.

6. *Get Column*

Method ini berguna untuk mengambil *list column* yang ada pada *class table* beserta dengan metadata nya. *Method* ini tidak menerima *parameter* apapun dan akan mengembalikan sebuah array dari *class column*.

7. *Create Column*

Method ini berguna untuk menambahkan *column* baru pada *class table*. Terdapat 1 *parameter* dalam *method* ini yaitu *instance column* yang nantinya akan dimasukkan ke dalam *list columns* yang tersedia pada table. Lalu untuk penambahan *column* ini, nantinya pada bagian *row* data juga akan ikut

bertambah data kosong atau default *value* dari *column* yang baru saja ditambahkan.

8. *Update Column*

Berguna untuk mengganti nama *column*. *Method* ini akan menerima 1 *parameter* berupa *string* yang isinya merupakan nama baru dari *column* yang ingin diubah. Hasil *return* dari *method* ini adalah instance *column* yang datanya berhasil diubah.

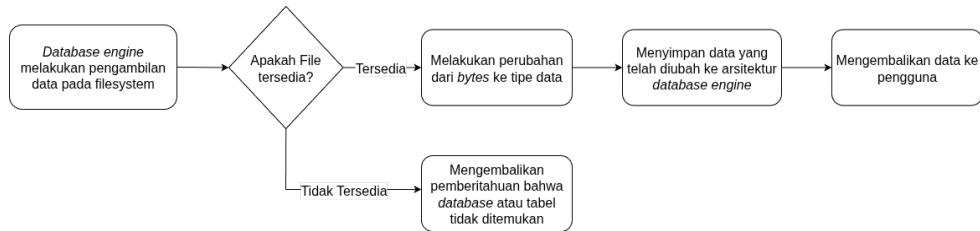
9. *Delete Column*

Berfungsi untuk menghapus *column* beserta data dari *column* tersebut. *Method* ini hanya menerima 1 *parameter* saja yaitu nama dari *column* yang ingin kita hapus sehingga *parameter* ini akan mempunyai tipe data *string*. Hasil *return* dari *method* ini adalah status keberhasilan penghapusan *column*.

3.4 Alur Penyimpanan Data kepada *Filesystem*

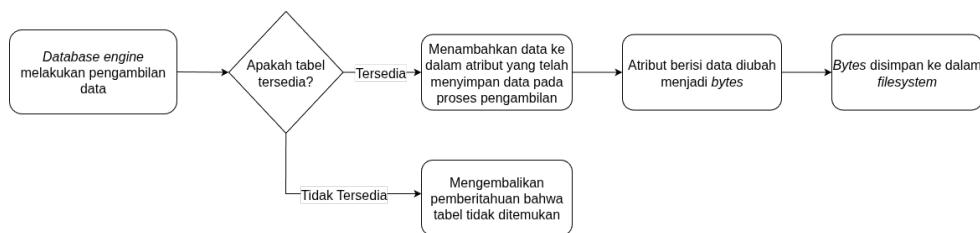
Data akan disimpan secara persisten ke dalam *filesystem* dalam bentuk *bytes*, sehingga terdapat beberapa tahapan untuk melakukan penyimpanan ke *filesystem*. *Client* akan melakukan pengolahan data melalui *interface database* yang disediakan. Terdapat proses pengambilan data dan proses penyimpanan data. Kedua proses ini mempunyai alur yang mirip namun ada sedikit perbedaan.

Untuk proses pengambilan data, *database engine* akan terlebih dahulu melakukan pengecekan, apakah *database* atau tabel yang dituju tersedia atau tidak. Jika tidak, maka *database* atau tabel belum dibuat atau tidak ada sehingga diperlukan pembuatan terlebih dahulu. Namun, jika *database* atau tabel tersedia maka langkah berikutnya adalah melakukan pengubahan data dari *bytes* ke tipe datanya. Tipe data ini akan tersedia di dalam *bytes* yang disimpan pada *filesystem*. Setelah berhasil melakukan pengubahan, maka data tersebut akan disimpan secara sementara pada atribut yang ada di dalam struktur data *database*. Berikut adalah alur pengambilan data dalam bentuk diagram:



Gambar 3.3: Proses pengambilan data dari *filesystem*

Pada bagian proses pembuatan dan penyimpanan data, memiliki alur yang sama dengan pengambilan data, namun terdapat proses tambahan, yaitu data harus dikembalikan dalam bentuk *bytes* dan disimpan di dalam *filesystem*. Proses ini dimulai dari *client* yang akan menjalankan proses *insert* pada *interface database*. *Database engine* akan melakukan hal yang sama yaitu pengambilan data, namun jika tabel yang dituju tidak tersedia maka akan mengembalikan sebuah pemberitahuan bahwa tabel tak bersedia. Setelah data berhasil diambil dari *filesystem* dan disimpan ke dalam atribut, maka proses perubahan dari data ke *bytes* akan dilakukan. Data yang telah berubah menjadi *bytes* akan disimpan ke dalam *filesystem*. Berikut adalah alur dari proses penyimpanan jika data dalam atribut *database* tersedia:



Gambar 3.4: Proses penyimpanan data ke *filesystem*

3.5 Index Sederhana dengan *Map*

Untuk proses *indexing* pada *database engine*, akan memanfaatkan struktur data bernama *Map*. Struktur data *map* yang digunakan mengikuti *library* standar pada bahasa pemrograman yang akan digunakan. Dengan menggunakan *map* yang ada dalam bahasa pemrograman, maka tidak perlu membuat struktur data khusus untuk menerapkan *index* pada *database*. Karena menggunakan *library* standar, struktur data ini juga akan memiliki *function* yang dapat digunakan untuk mempermudah proses pengembangan. Selain memiliki *function-function* yang dapat digunakan, *Map* yang tersedia pada *library* standar bahasa pemrograman umumnya

dapat menyimpan berbagai jenis tipe data. Penerapan *Map* di arsitektur *database engine* ini kemungkinan akan diterapkan dengan struktur data yang lain seperti *array*.

3.6 *Join* Antar Tabel

Sesuai dengan pernyataan pada batasan masalah, fitur *join* yang ada pada *database engine* awal ini adalah melakukan *join* antar 2 tabel saja. Prinsip *join* yang digunakan sama seperti *join* yang dilakukan pada *database relational* lainnya, yaitu dengan menggunakan *foreign key*. Untuk menghubungkan kedua tabel dalam satu *database*, maka terdapat satu kolom (biasanya berupa ID) dalam sebuah tabel yang nilai kolom dari tabel tersebut merujuk ke tabel lain. Jika terdapat baris dengan nilai sama pada kolom yang telah ditentukan antar tabel, maka baris tersebut akan dianggap menjadi 1 baris. Perlu diketahui bahwa fitur *join* ini masih bersifat mirip seperti *LEFT JOIN* yang ada di *relational database* dan baru bisa dilakukan antar 2 tabel.

3.7 Komunikasi Interface *Database* dengan D-Bus

Komunikasi dengan D-Bus akan menggunakan *library* D-Bus yang tersedia pada bahasa pemrograman yang digunakan. Pembuatan koneksi ini menggunakan bahasa pemrograman yang sama dengan bahasa pemrograman yang digunakan dalam pengembangan *database engine*. Namun, meski dibuat dengan satu bahasa pemrograman, koneksi D-Bus tetap bersifat universal (Dapat digunakan pada sistem lain). Sehingga jika *client* menggunakan bahasa pemrograman lain, koneksi D-Bus tetap bisa terhubung satu sama lain.

Koneksi pada D-bus ini akan dihubungkan melalui path yang nanti akan ditentukan sebagai path dari jalur koneksi *database engine*. *Client* harus mendefinisikan *path* yang sama dengan *database engine* agar bisa terhubung. Dengan koneksi ini, pihak *client* dapat menjalankan fungsi-fungsi *database engine* berdasarkan *interface* yang telah dibuka. *Interface* yang dibuka hanyalah interface yang dapat berguna bagi *client* dan bukan *interface* yang bersifat *low-level*.

3.8 Alat dan Bahan

Untuk melaksanakan penelitian terdapat alat dan bahan yang dibutuhkan, di antaranya adalah:

- Laptop yang mempunyai spesifikasi untuk menjalankan bahasa pemrograman *rust* dan *python*. Versi bahasa pemrograman yang digunakan saat pengembangan adalah rust versi 1.84.1 dan python versi 3.10.12.
- Gemini AI dan ChatGPT sebagai alat untuk mencari ide dan *brainstorm*.
- *Visual Studio Code* sebagai *Code Editor* beserta *extension rust analyzer*.
- Koneksi Internet.

3.9 Skema Uji

Pengujian akan dilakukan dengan menjalankan servis *Client* dan *Database Engine*. Untuk melakukan pengujian, servis *Client* akan menyiapkan data *Dummy* yang berasal *Faker API*. *Faker API* merupakan sebuah alat yang dapat membuat berbagai macam jenis data sehingga akan cocok untuk digunakan dalam melakukan pengujian. Data dari *Faker API* tersebut akan disimpan menggunakan JSON yang nanti akan di *import* ke dalam servis *Client*. Lalu servis *Client* yang akan mengirimkan *data dummy* tersebut ke *database engine* melalui koneksi D-Bus yang telah di *expose*. *Database engine* akan mengolah data yang telah dikirim, berdasarkan *interface* yang telah di *expose* pada D-Bus.

BAB IV

HASIL DAN PEMBAHASAN

4.1 Implementasi

Terdapat perubahan pada *interface* yang telah dibuat pada gambar 3.2. Perubahan serta fungsionalitas dari *method* pada interface akan dijelaskan pada sub bab implementasi dibawah. Selain itu, terdapat perbedaan pada bahasa pemrograman yang di jelaskan pada gambar 3.2, bahasa pemrograman yang digunakan dalam penelitian ini adalah bahasa *low-level* bernama rust. Versi bahasa rust yang digunakan pada saat pengembangan adalah rust dengan versi 1.84.1, namun untuk menjalankan hasil akhir dari *database engine* ini, tidak perlu menginstall bahasa rust karena hasil akhir atau hasil build dari bahasa rust merupakan *file bin* yang dapat langsung dijalankan.

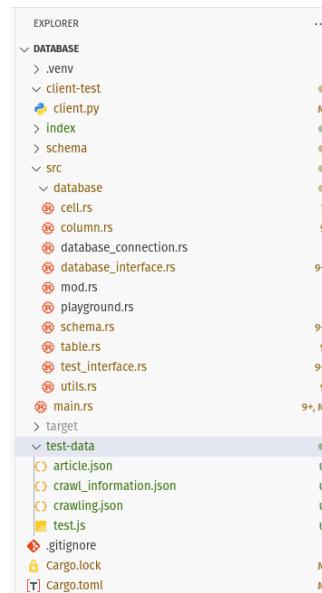
4.1.1 Bahasa Pemrograman Rust

Rust merupakan salah satu bahasa pemrograman yang mempunyai tingkat kecepatan yang tinggi. Berdasarkan web resmi dari dokumentasi rust, rust memiliki performa yang baik, ketahanan yang baik dan produktivitas yang baik (dokumentasi yang lengkap). Rust dapat digunakan pada teknologi *low-level* seperti embedding system, web assembly, pembuatan command line dan lainnya. Berdasarkan paper *Towards Understanding the Runtime Performance of Rust* Zhang et al., 2022, kecepatan performa rust tidak jauh berbeda dengan bahasa pemrograman C. Dengan begitu rust menjadi salah satu bahasa tercepat untuk saat ini. Rust bukanlah bahasa pemrograman yang mengadopsi Object Oriented Programming sehingga tidak memiliki class. Namun terdapat alternatif dalam penggunaan *class* yaitu struct. Agar penulisan lebih jelas dan mudah dipahami, istilah class akan digunakan daripada istilah struct.

Dengan membuat menggunakan bahasa rust, hasil akhir dari pengembangan ini akan dilakukan sebuah proses yang bernama *build*. Dari proses *build* ini, database akan berjalan. Jika sudah dalam bentuk hasil *build*, maka sudah tidak perlu memasang bahasa rust pada komputer yang dituju karena hasil dari *build* merupakan *file* yang dapat dijalankan secara langsung.

4.1.2 Struktur Folder

Struktur folder yang digunakan merupakan dasar struktur *folder* dari rust. Dalam *folder* src terdapat *folder database* yang berisi *class* dan *method* yang akan digunakan dalam menjalankan *database engine*. Terdapat 1 *file* bernama playground.js hanya sebagai penyimpanan code saat pengembangan. dan tidak digunakan pada *database engine*. Terdapat 3 *folder* tambahan di bagian *root folder*, yaitu client-test (untuk menyimpan pengujian client), schema (menyimpan *file database*) dan index (untuk menyimpan *index* yang terdapat pada database). Lalu di bagian *root folder* juga terdapat *folder* test-data, dalam *folder* ini berguna untuk menyimpan data-data yang berguna untuk testing dan dokumentasi. Terdapat dapat beberapa *file* seperti article.json, crawling.json, dan crawl_information.json untuk menampung data pengujian sementara dan test.js yang hanya digunakan untuk membantu penulisan (Untuk memanfaatkan *extension prettier* untuk mempercantik dokumentasi hasil data, seperti pada gambar 4.19). Berikut adalah struktur *folder* dari *database engine*:



Gambar 4.1: Struktur Folder

4.1.3 Schema

Class Schema merupakan *class* induk pada *database engine* yang dibuat. *Class* ini menampung bagian penyimpanan data dan sebagai portal bagi pengolahan

data yang terjadi pada *database engine*. *Class Schema* ini akan merepresentasikan satu entitas *database*. Berikut adalah *class diagram* dari *class Schema*.

Schema
<pre>+ 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): usize + 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, table_inner_join: String, column_inner_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</pre>

Gambar 4.2: Class Diagram Schema

Atribut dan *method* yang tertulis pada *class diagram* berguna untuk fitur yang ada di *database* dan juga berguna saat pengembangan *database engine*. Terdapat 3 attribut pada *class Schema*, yaitu:

1. name

Berguna untuk menyimpan nama dari *database* dalam bentuk tipe data string.

2. tables

Attribut ini berguna untuk menyimpan list tabel yang ada pada *database* atau schema yang dibuat. Tipe data atribut ini adalah Vector berisi *class table*, yang akan dijelaskan pada pembahasan *class* berikutnya.

3. index

Atribut ini berguna untuk menyimpan *index* yang ada pada instance schema yang dibuat. Tipe data dari atribut ini adalah *HashMap*, dan berisi struktur

data yang cukup kompleks atau dalam. Hasmap pada atribut index ini memiliki key dengan tipe String, dan value sebuah vector. Vector didalamnya berisi beberapa HashMap yang nanti digunakan sebagai baris data yang telah disimpan. Lalu HashMap terdalam ini mempunyai key String dengan Value *class InputDataEnum*.

Untuk menggunakan dan mengolah data-data yang ada di dalam atribut, dibutuhkan method-method yang sudah dibuat dalam *class Schema*. Terdapat 21 *method* yang dapat digunakan, yaitu:

1. *to_bytes*

Method ini berguna untuk mengubah data yang telah disimpan pada atribut tabel. *Method* ini mengembalikan sebuah vector yang berisi u8. Tipe u8 ini yang akan mempresentasikan *bytes* dan disimpan dalam *filesystem*. Penyimpanan *bytes* dalam vector memiliki urutan yang harus diperhatikan dan tidak boleh salah. Karena jika salah dapat menyebabkan data gagal diambil. Berikut adalah ilustrasi urutan dari penyimpanan *byte* untuk *class Schema*.

[schema_config, table_1_config, column_1_config, column_2_config,..., table_2_config, column_1_config,]

Gambar 4.3: Urutan penyimpanan Schema secara menyeluruh dalam *byte*

Untuk ilustrasi diatas, merupakan urutan penyimpanan data ke dalam bentuk *byte* secara menyeluruh. *schema_config*, *table_1_config*, *table_2_config*, *column_1_config* dan lainnya merupakan urutan penyimpanan dari masing-masing class. Isi dari urutan penyimpanan tersebut akan dijelaskan di dalam *method to_bytes* pada masing-masing *class*. Berikut ini merupakan isi urutan dari *schema_config*.

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

Gambar 4.4: Urutan penyimpanan Schema dalam *byte*

(a) *name_len*

merupakan dari panjang atribut name yang disimpan dalam bentuk u8. Bytes yang disimpan dibuat dengan panjang 8 mengikuti aturan primitif pada isize di bahasa pemrograman rust.

(b) name_val

merupakan value dari atribut name yang disimpan dalam bentuk u8.

Panjang *bytes* ini dapat berbeda-beda bergantung dengan isi dari nilainya.

(c) table_len

merupakan dari panjang data tersimpan pada atribut tables yang disimpan dalam bentuk u8. Panjang penyimpanan juga mengikuti seperti apa yang telah di kutip di name_len

(d) table.to_bytes

Urutan dilanjutkan memanggil *method* to.bytes pada setiap data tersimpan pada atribut tables

2. to_data

Method ini berguna untuk mengembalikan data yang telah disimpan pada *method* to_bytes dari bentuk vector u8 menjadi *class* yang sebelumnya disimpan pada *filesystem*. Sehingga proses pengembaliamnya merupakan kebalikan dari proses yang ada pada *method* to_bytes. *Method* ini mengembalikan tipe data schema yang telah berisikan data. *Method* ini bersifat static.

3. save

Method ini digunakan untuk melakukan proses penyimpanan dari vector *bytes* ke dalam *filesystem*. Di dalam *method* save, *method* to_bytes akan dipanggil dan return dari *function* tersebut yang akan disimpan pada *filesystem*. File disimpan ke dalam *filesystem* dalam *folder* schema, dan *file* tersebut dinamakan dengan nama schema yang telah dibuat. *Method* save ini nanti juga akan selalu dipanggil setiap ada perubahan data seperti create, update dan juga delete untuk menunjang integritas data.

4. add_table

Method yang berguna untuk menambah jumlah table yang disimpan pada *class* Schema. Di dalam *method* ini juga terdapat pengecekan keunikan nama tabel. Jika terdapat tabel baru yang memiliki nama yang sama, maka *method* ini akan mengembalikan error.

5. add_column_to_table

Method yang berguna untuk menambahkan sebuah kolom baru pada tabel yang dituju.

6. `list_column_on_table`

Method yang berguna untuk menampilkan kolom-kolom pada tabel yang dituju.

7. `search_table`

Method yang berguna untuk mencari tabel yang terdapat dalam schema.

8. `check_table_index`

Method yang berguna untuk menampilkan *index* atau urutan tabel di dalam vector table berdasarkan nama tabel pada *parameter*.

9. `delete_table`

Method yang berguna untuk menghapus tabel berdasarkan nama pada *parameter*.

10. `add_data`

Method yang berguna untuk menambah data baru berdasarkan tabel yang telah disimpan pada *database*.

11. `get_data`

Method yang berguna untuk menampilkan semua data yang telah tersimpan pada tabel yang dituju.

12. `search_data`

Method yang berguna untuk menampilkan pencarian data berdasarkan kolom-kolom pada tabel yang dituju.

13. `update_data`

Method yang berguna untuk mengubah data berdasarkan kondisi yang ada pada *parameter*. Data yang diubah adalah data yang juga ditentukan pada *parameter*.

14. `delete_data`

Method yang berguna untuk menghapus data berdasarkan kondisi yang ada pada *parameter*.

15. `join_table`

Berguna untuk melihat data antar tabel yang dihubungkan melalui kolom yang telah dibuat pada saat pembuatan tabel. *Method* ini berguna sebagai pengondisian pemanggilan *method* berdasarkan tipe *join* yang tersedia pada *database engine* ini yaitu *inner join*, *left join* dan *right join*.

16. `left_right_join_table`

Method yang berguna untuk melakukan *left* atau *right join*. Perbedaan proses dari *left* dan *right join* terletak hanya pada *parameter* dari *method* ini.

17. `join_inner_table`

Method yang berguna untuk melakukan *inner join*.

18. `build_index`

Method yang berguna untuk membangun *index* berdasarkan data *index* yang telah disimpan pada *file index*. *Method* ini dipanggil di dalam *method to_data*.

19. `save_index`

Method yang berguna untuk menyimpan *index* ke dalam *filesystem*. *Index* ini disimpan ke *file* didalam *folder index*. Nama dari *file* tersebut juga akan disamakan dengan nama schema yang telah dibuat.

20. `clear_index`

Method yang berguna untuk membersihkan atau menghapus *index* yang telah disimpan. *Method* ini selalu dipanggil di dalam *method save*.

21. `get_file`

Sebuah *method* yang bersifat private, berguna untuk mengembalikan Object File berdasarkan nama yang ada pada *parameter*. *Method* ini di panggil di dalam *method build_index*.

22. `list_all_table`

Method yang berguna untuk menampilkan nama tabel yang tersimpan pada schema dalam bentuk vector string.

23. `print`

Method yang digunakan hanya pada saat pengembangan berlangsung, berguna untuk menampilkan data pada terminal tempat *database engine* di jalankan.

4.1.4 Table

Class `Table` merupakan *class* yang merepresentasikan sebuah tabel dalam *database*. Berikut adalah gambar *class diagram* dari *class Table*:

Table
<pre>+ 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></pre>

Gambar 4.5: Class Diagram Table

Atribut dan *method* yang tertulis pada *class* diagram berguna untuk melakukan pengolahan data dari satu entitas Table. Sebagian besar dari atribut dan *method* ini dipanggil pada *class* Schema. Terdapat 3 atribut pada *class* Schema, yaitu:

1. name

Berguna untuk menyimpan nama dari tabel dalam bentuk tipe data string.

2. columns

Attribut ini berguna untuk menyimpan list kolom pada tabel yang dibuat. Tipe data atribut ini adalah Vector berisi *class Column*, yang akan dijelaskan pada pembahasan *class* berikutnya.

3. length

Berguna untuk menyimpan panjang dari vector columns. Atribut ini dapat dikatanak menyimpan panjang dari baris di tabel.

Setelah atribut, terdapat method-method dalam *class Table* yang berguna untuk mengolah data yang disimpan di dalam atribut. Terdapat 18 *method* dalam *class Table*, yaitu:

1. get_type

Berguna untuk mengembalikan tipe data dari kolom yang dituju.

2. get_data_by_column

Berguna untuk mengambilkan data-data yang tersimpan dalam kolom tabel dalam bentuk hashmap.

3. get_column_names

Berguna untuk melihat nama kolom yang tersimpan pada tabel dalam bentuk vector string.

4. get_data

Berguna untuk mengambil seluruh data yang tersimpan pada satu *class Table*.

5. get_data_by_index

Method ini bersifat private, berguna untuk mengambil *index* atau urutan data yang tersimpan pada tabel.

6. search_by_column

Method ini berguna untuk melakukan pencarian terhadap kolom yang dituju.

7. search_column

Berguna untuk melakukan pencarian kolom berdasarkan nama kolom yang telah dibuat.

8. print

Berfungsi sama seperti *method* print yang ada pada *class Schema*, berguna untuk menampilkan data yang tersimpan pada *class Table* di terminal tempat *database engine* berjalan. *Method* ini sering digunakan pada saat pengembangan saja.

9. add_column

Berfungsi untuk menambah atau membuat kolom baru.

10. check_column_index

Berfungsi untuk mengembalikan *index* atau urutan kolom yang tersimpan pada atribut *columns*.

11. add_data_column

Berfungsi untuk menambahkan 1 data pada suatu kolom, *method* ini tidak digunakan dalam fitur *database engine* ini karena dapat menghancurkan integritas antar kolom di sebuah tabel. Mungkin *method* ini akan berguna di fitur yang akan datang.

12. add_data

Berfungsi untuk menambahkan 1 baris data baru pada tabel.

13. update

Berfungsi untuk mengubah nilai atau isian pada 1 baris data yang dituju di sebuah tabel menggunakan *index* atau urutan kolom pada tabel.

14. delete_column

Berfungsi untuk menghapus kolom yang tersimpan pada tabel. Dengan menghapus kolom, maka semua data yang tersimpan pada kolom tersebut juga akan dihapus. Menggunakan nama kolom untuk mencari kolom yang dituju.

15. delete

Berfungsi untuk menghapus data pada tabel menggunakan *index* atau urutan dari data yang telah tersimpan.

16. update_data

Memiliki fungsi yang sama dengan *method* update, namun menggunakan pencarian berdasarkan kolom-kolom yang ada pada tabel untuk mencari data yang ingin diubah.

17. `delete_data`

Memiliki fungsi yang sama dengan *method* `delete`, perbedaannya terletak pada pencarinya. *Method* ini menggunakan kolom-kolom yang tersedia untuk menentukan baris data yang ingin dihapus.

18. `to_bytes`

Memiliki fungsi yang sama dengan *method* `to_bytes` pada *class* `Schema`, namun dikhususkan untuk mengubah *class* `Table` saja. Proses pengubahan disimpan ke dalam vector `u8` dengan urutan yang harus konsisten. Berikut adalah ilustrasi urutan penyimpanan dalam vector tersebut.

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

Gambar 4.6: Urutan penyimpanan Table dalam byte

(a) `name_len`

merupakan dari panjang atribut `name` yang disimpan dalam bentuk `u8`. *Bytes* yang disimpan dibuat dengan panjang 8 mengikuti aturan primitif pada `isize` di bahasa pemrograman rust.

(b) `name_val`

merupakan value dari atribut `name` yang disimpan dalam bentuk `u8`. Panjang *bytes* ini dapat berbeda-beda bergantung dengan isi dari nilainya.

(c) `column_len`

merupakan dari panjang data tersimpan pada atribut `columns` yang disimpan dalam bentuk `u8`. Panjang penyimpanan juga mengikuti seperti apa yang telah dikutip di `name_len`

(d) `data_table_len`

merupakan value dari atribut `length` yang disimpan dalam bentuk `u8`. Panjang *bytes* ini juga mengikuti aturan pada `name_len`.

(e) `column.to_bytes`

Urutan dilanjutkan memanggil *method* `to.bytes` pada setiap data tersimpan pada atribut `columns`

(f) `next_table_config`

Dilanjutkan dengan memanggil pengaturan tabel yang berikutnya.

19. `to_data`

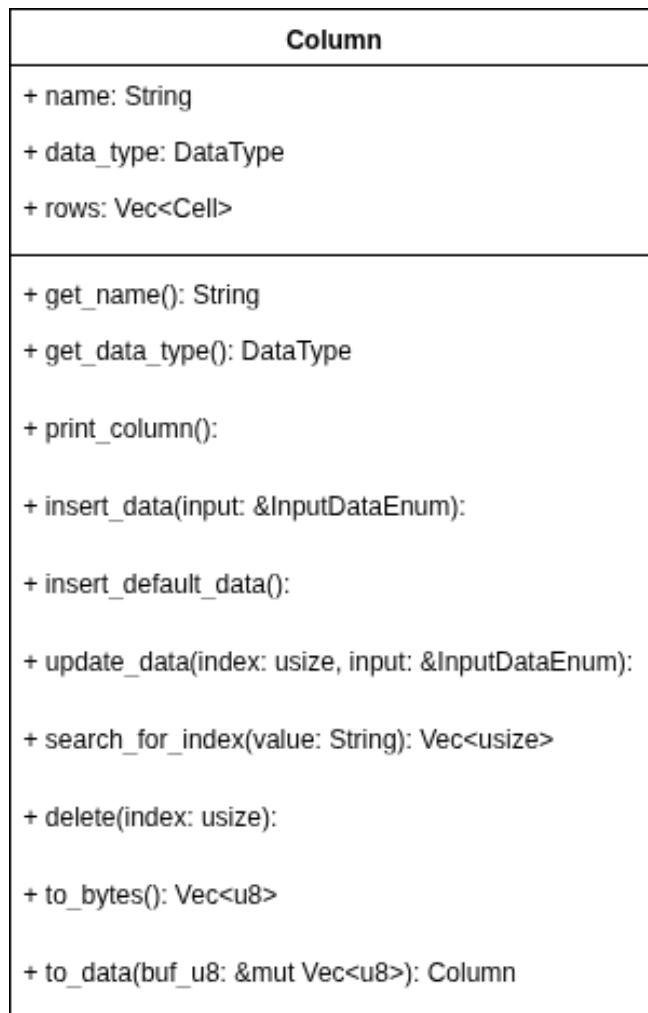
Proses yang berkebalikan dari *method* `to_bytes`, yaitu mengembalikan data dalam bentuk *bytes* ke bentuk aslinya (class). Data yang dikembalikan dari *method* ini berupa *class* Table.

20. `name_to_bytes`

Method yang berfungsi untuk mengubah nama tabel ke dalam bentuk vector u8.

4.1.5 Column

Class `Column` merupakan *class* yang merepresentasikan sebuah kolom didalam sebuah tabel. Berikut adalah gambar *class* diagram dari *class* `Column`:



Gambar 4.7: Class Diagram Column

Atribut dan *method* yang tertulis pada *class* diagram berguna untuk melakukan pengolahan data dari satu entitas Column. *Class* Column dapat tersimpan dalam jumlah banyak di *class* Table. Sebagian besar dari atribut dan *method* ini juga akan dipanggil di *class* Table. Terdapat 3 atribut pada *class* Column, yaitu:

1. name

Berguna untuk menyimpan nama dari kolom dalam bentuk tipe data string.

2. data_type

Attribut ini berguna untuk menyimpan tipe data dari sebuah kolom. Tipe data atribut ini adalah Enum *DataType*.

3. rows

Berguna untuk menyimpan data pada columns. Tipe data atribut rows memiliki tipe vector yang berisi *class* Cell.

Selain atribut, *class* Column ini juga memiliki *method* yang berguna untuk mengolah data yang disimpan di atribut. Terdapat 10 *method* di dalam *class* Column, yaitu:

1. get_name

Berguna untuk mengembalikan nama dari kolom.

2. get_data_type

Method yang berguna untuk mengembalikan tipe data dari data yang disimpan dalam *class* Column.

3. print_column

Method ini hanya dipakai pada saat pengembangan, dan memiliki fungsional yang sama seperti *print* *method* di *class* Schema dan *class* Table. Berfungsi untuk menampilkan data yang disimpan pada terminal tempat *database engine* dijalankan.

4. insert_data

Method ini berguna untuk membuat data baru pada kolom. Data baru yang masuk ke dalam kolom harus disesuaikan dengan tipe data yang ada pada atribut data_type.

5. insert_default_data

Method ini berguna untuk membuat data baru pada kolom dengan nilai default. String kosong untuk tipe data string dan nilai 0 untuk tipe data integer. Hal ini dibuat jika pengguna tidak mendefinisikan nilai pada kolom yang ada pada tabel saat menambahkan data baru.

6. update_data

Berguna untuk mengubah data yang tersimpan pada atribut rows. *Method* ini dipanggil di dalam *method* update di *class* Table.

7. search_for_index

Berguna untuk mengembalikan *index* dari sebuah data yang tersimpan pada atribut rows. Rows yang dikembalikan adalah rows yang memiliki value yang ada di *parameter*. *Method* ini dapat mengembalikan indeks lebih dari 1 dalam bentuk vector.

8. delete

Berguna untuk menghapus data berdasarkan urutan atau indeks yang tersimpan pada atribut rows.

9. to_bytes

Sama seperti *method* di *class* sebelumnya, berguna untuk mengubah data tersimpan ke dalam bentuk *bytes* atau vector u8. Proses penyimpanan ke dalam vector memiliki urutan yang harus sesuai. Berikut adalah ilustrasi urutan yang disimpan dalam vector.

```
[..., data_column_len, name_len, name_val, data_type, cell_1.data_value, cell_2.data_value, ..., next_column_config]
```

Gambar 4.8: Urutan penyimpanan Column dalam byte

(a) data_column_len

merupakan dari panjang data tersimpan pada atribut rows yang disimpan dalam bentuk u8. Bytes yang disimpan dibuat dengan panjang 8 mengikuti aturan primitif pada isize di bahasa pemrograman rust.

(b) name_len

merupakan dari panjang data dari atribut name yang disimpan dalam bentuk u8. Panjang penyimpanan juga mengikuti seperti apa yang telah di kutip di data_column_len

(c) name_val

merupakan value dari atribut name yang disimpan dalam bentuk u8. Panjang *bytes* ini dapat berbeda-beda bergantung dengan isi dari nilainya.

(d) data_type

Berguna untuk menyimpan tipe data dari kolom. Pada urutan ini jika bernilai 0 maka tipe data dari kolom nya adalah 0, dan jika bernilai 1 maka bernilai integer.

(e) cell.data_value

Urutan dilanjutkan dengan melakukan for loop dan memanggil atribut data_value pada setiap cell yang disimpan di dalam atribut rows. Karena atribut data_value sudah berupa vector u8 maka nilainya bisa langsung dimasukkan ke dalam vector. Namun terdapat sedikit perbedaan proses pada string, karena harus menambah panjang dari value string yang tersimpan terlebih dahulu sebelum menyimpan nilai aslinya.

(f) next_column_config

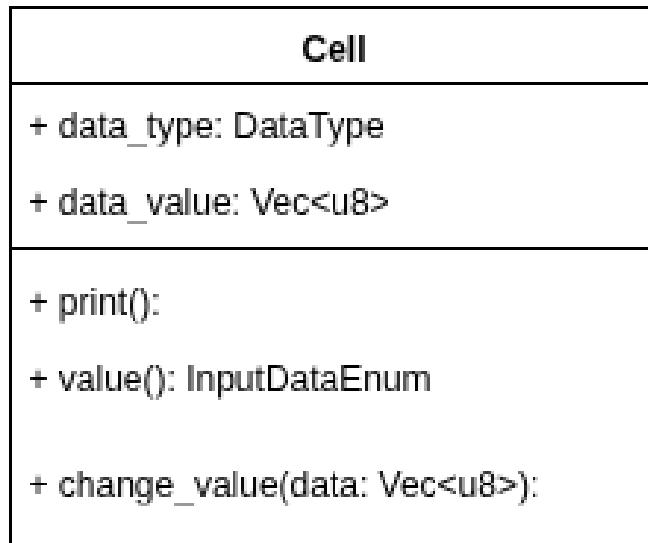
Dilanjutkan dengan memanggil pengaturan kolom yang berikutnya.

10. to_data

Berfungsi untuk mengembalikan data kolom yang telah diubah ke *bytes* menjadi data *class Column* kembali.

4.1.6 Cell

Class Cell merupakan *class* yang merepresentasikan sebuah nilai yang tersimpan pada satu baris tabel. *Class Cell* dibutuhkan karena, setiap nilai yang tersimpan dalam database dapat memiliki tipe data yang berbeda-beda. Sehingga diperlukan sebuah *class* kembali untuk menentukan hal tersebut. Berikut adalah gambar *class diagram* dari *class Cell*:



Gambar 4.9: Class Diagram Cell

Class Cell memiliki atribut dan beberapa *method* yang berguna untuk menunjang fitur pada *database engine*. Terdapat 2 atribut dalam *class* Cell, yaitu:

1. *data_type*

Berguna untuk menyimpan tipe data dari sebuah cell. Atribut ini memiliki tipe data Enum *DataType* dan berfungsi untuk memvalidasi tipe data yang ada pada cell dan *class* Column tempat cell disimpan.

2. *data_value*

Berguna untuk menyimpan nilai atau value dari sebuah cell. Nilai atribut ini disimpan dalam bentuk vector u8. Tujuan dibuat seperti ini dikarenakan nilai yang disimpan dalam sebuah cell dapat memiliki tipe yang berbeda-beda. Selain itu, dengan menggunakan vector u8 sebagai nilai dari atribut ini, proses penyimpanan ke *filesystem* menjadi lebih karena dapat dilakukan tanpa perlu mengubahnya terlebih dahulu.

Atribut yang disimpan dalam sebuah Cell harus diolah agar dapat dikelola oleh *database engine*. Dengan menggunakan method, kita dapat mengolah atribut agar dapat digunakan pada fitur *database engine*. Terdapat 3 *method* yang terdapat pada *class* Cell, yaitu:

1. *print*

Method yang berguna selama pengembangan *database engine*. Berguna untuk menampilkan value dalam cell di terminal tempat *database engine* dijalankan.

2. value

Berguna untuk mengembalikan value dari bentuk *byte* ke *Enum InputDataEnum*. Hal ini diperlukan karena nilai yang tersimpan dalam cell adalah *bytes*, sehingga membutuhkan pemrosesan kembali ke tipe data aslinya.

3. change_value

Berguna untuk mengubah value yang tersimpan pada atribut *data_value* sesuai dengan yang ada pada *parameter*. Data dalam *parameter* juga sudah dalam bentuk vector u8.

4.1.7 Enumeration

Terdapat beberapa nilai enum yang digunakan sebagai penyimpanan tipe data dalam *database*. Terdapat 2 *Enum class*, yaitu:

1. InputDataEnum

Enum ini berguna untuk menyimpan data dalam bentuk String, Integer dan juga Null. Namun pada fitur *database* kali ini, hanya 2 tipe data saja yang baru didukung untuk penyimpanannya yaitu String dan Integer. Khusus untuk *enum* ini, akan menggunakan library bernama *serde* dan *serde_derive* untuk melakukan *Serialization* dan *Deserialization*. Hal ini diperlukan saat sedang melakukan penyimpanan *index*, karena *index* disimpan dalam bentuk *HashMap*. Dengan menggunakan library ini *Hashmap* yang berisi *InputDataEnum* dapat berubah ke dalam bentuk *bytes* secara langsung tanpa perlu mengubahnya secara manual. Selain library *serde* dan *serde_derive*, terdapat 1 macro lagi yang diterapkan pada *enum* ini yaitu *Debug*. Macro ini berguna agar value yang memiliki tipe *InputDataEnum* dapat dilihat hasilnya di terminal.

2. DataType

Enum yang berguna untuk menentukan tipe kolom yang tersimpan pada tabel dan tipe *Cell* yang tersimpan pada kolom.

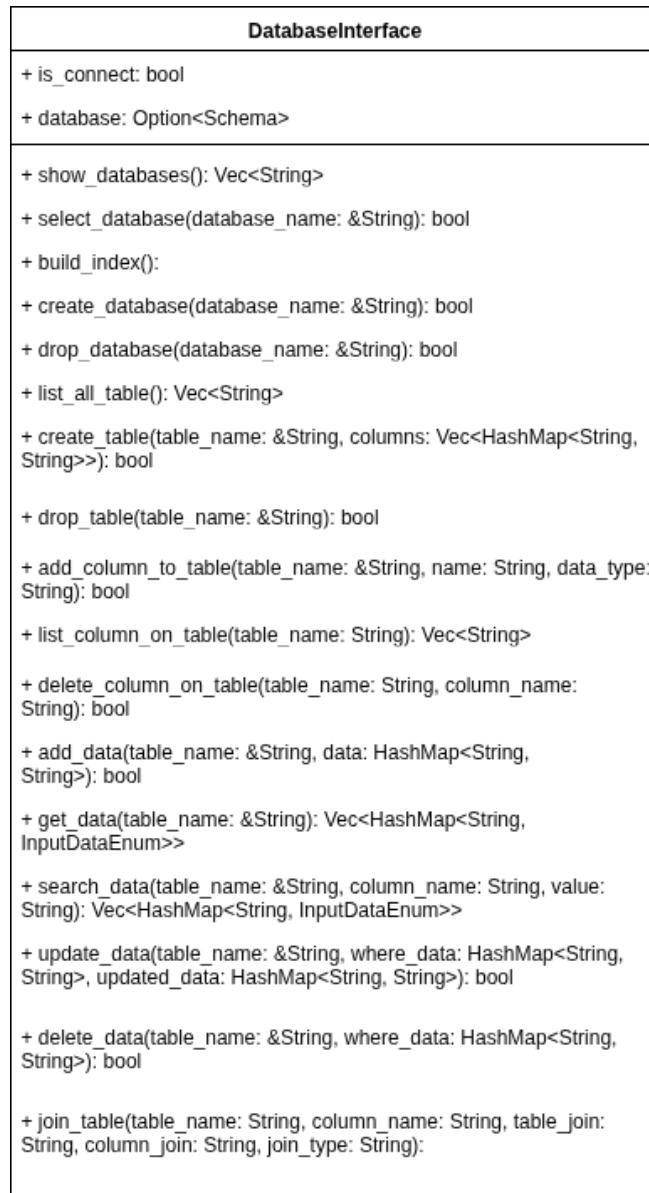
4.1.8 Utils

Selain beberapa *class* inti, terdapat beberapa *function* tambahan yang berguna untuk menunjang fitur-fitur yang tersedia pada *database engine*. Semua *function* ini

tersimpan secara statis pada *file* utils.rs. Namun tidak semua *function* dalam utils digunakan dalam penerapan fitur *database engine*, sebagian hanya digunakan saat hendak melakukan debugging atau saat hendak melakukan percobaan.

4.1.9 DatabaseInterface

Class *DatabaseInterface* ini berfungsi sebagai penentu *method* atau *function* mana saja didalam *database* yang dapat dijalankan oleh pengguna. Oleh karena itu, *method-method* di dalam class ini hanya meneruskan paramater yang diterima ke *method* yang ada dalam *class Schema* sesuai dengan nama dari methodnya. Berikut adalah *class Diagram* *DatabaseInterface*.



Gambar 4.10: Class Diagram *DatabaseInterface*

Perlu diperhatikan bahwa sebagian method-method di dalam *class* ini, terutama yang berhubungan dengan pengolahan data dalam database, memiliki validasi atau pengecekan apakah *class DatabaseInterface* sudah terhubung dengan *class Schema*. *Class DatabaseInterface* ini memiliki 2 atribut yaitu:

1. `is_connect`

Atribut dengan tipe data boolean untuk menentukan apakah instance schema sudah terhubung atau belum.

2. database

Atribut dengan tipe Option yang akan berisi instance dari *class Schema* dan dapat berisi None jika instance Schema belum terhubung.

Selain sebagai penentu method, terdapat method lain yang terdapat pada *DatabaseInterface*. Berikut adalah method-method tambahan yang ada pada *DatabaseInterface*:

1. show_databases

Method ini berguna untuk melihat list *database* yang tersimpan dalam bentuk *file* di *folder schema*.

2. select_database

Method ini berguna untuk memilih *database* dan memasukkannya ke dalam attribute *database*.

3. create_database

Berguna untuk membuat *database* dan memanggil *function save* di dalam *class Schema* sehingga sebuah *file* baru dibuat di dalam *folder schema* sesuai dengan nama *database* yang dimasukkan dalam *parameter*.

4. drop_database

Berguna untuk menghapus *database* dengan cara menghapus *file database* pada *folder schema* dan *folder index*.

5. join_data

Terdapat sedikit perbedaan pada proses yang terjadi di dalam *method join_data* ini. Isi *method* ini akan dijelaskan lebih secara khusus pada sub bab 4.1.10 tentang *indexing*

4.1.10 *Indexing* pada fitur Join

Penerapan fitur *indexing* pada *database engine* ini dilakukan pada fitur *join*. Prinsip penerapannya sama seperti *cache* pada umumnya. Hashmap digunakan untuk mengumpulkan *index* dengan menggunakan key dari *parameter* yang dijalankan saat hendak menjalankan *join*. Jika key dari *parameter* tersedia dalam Hashmap, maka value dari key tersebut akan dikembalikan. Sementara jika key tersebut tidak tersedia, maka akan memanggil *method join_data* yang ada pada *class*

Schema. Alur lebih detail terkait penerapan *indexing* dapat dilihat pada *method* *join_data* pada *class DatabaseInterface*. Namun untuk penerapan *indexing* saat ini masih belum berjalan dengan baik, dikarenakan value disimpan dalam *HashMap*, dimana *HashMap* tersebut merupakan library bawaan dari rust. 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. Untuk melakukan hal tersebut, membutuhkan sebuah fitur yang bernama lifetime dan fitur lifetime ini masih belum bisa dikuasai penulis saat ini. 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. Untuk penyimpanan *HashMap* dalam *filesystem*, akan menggunakan library bernama *bincode*. Karena proses pengubahan dari data ke byte yang saat ini dibuat, hanya ditujukan bagi *class schema*, *table* dan *column*.

4.1.11 DatabaseConnection

Class *DatabaseConnection* merupakan *class* yang berguna untuk *database engine* melakukan koneksi ke sistem lain. *Class* ini menggunakan *DatabaseInterface* sebagai penghubung ke *class Schema* untuk menjalankan fungsi-fungsi *database*. Seperti yang dikutip pada sub bab 3.7 bahwa untuk melakukan koneksi eksternal dengan sistem lain, *database engine* akan menggunakan D-Bus. Pada penelitian ini, penerapan D-Bus di *database* akan menggunakan library atau biasa disebut crates dalam bahasa rust. Crates tersebut bernama zbus. Untuk menggunakan zbus pada *class DatabaseConnection*, penerapan macro interface diperlukan. Hal ini sesuai dengan petunjuk yang ada pada dokumentasi zbus. Lalu *library* bernama tokio akan digunakan untuk membuat *connection* menggunakan zbus pada function main di *file* main.rs. Berikut adalah *class diagram* dari *class DatabaseConnection*.

DatabaseConnection
+ db_interface: DatabaseInterface
+ select_database(database_name: &str): String
+ create_database(database_name: &str): String
+ drop_database(database_name: &str): String
+ list_table(): Vec<String>
+ create_table(table_name: &str): String
+ drop_table(table_name: &str): String
+ add_column(table_name: &str, name: &str, data_type: &str): String
+ list_column(table_name: &str): Vec<String>
+ delete_column(table_name: &str, column_name: &str): String
+ add_data(table_name: &str, data: HashMap<String, String>): String
+ update_data(table_name: &str, where_data: HashMap<String, String>, updated_data: HashMap<String, String>): String
+ delete_data(table_name: &str, where_data: HashMap<String, String>): String

Gambar 4.11: Class Diagram Column

Class *DatabaseConnection* memiliki sebuah atribut bernama *db_interface*. Atribut ini memiliki tipe data dari class *DatabaseInterface*, yang berguna untuk memastikan bahwa *class DatabaseConnection* sudah terhubung dengan *class Schema*. Karena pada saat pertama kali *database engine* ini dinyalakan, *database* atau *schema* harus dipilih terlebih dahulu untuk menentukan di *schema* mana data akan diolah. Untuk memilih *schema*, dapat menggunakan *method* *select_database* yang terdapat pada *class DatabaseInterface*.

Perlu diketahui bahwa koneksi dengan menggunakan D-Bus untuk *database engine* saat belum mendukung semua fungsi-fungsi pada *database*. Karena secara default, koneksi dengan D-Bus hanya mendukung beberapa tipe data yang ada pada Rust saja. Untuk tipe data *class* yang dibuat sendiri, diperlukan pengaturan tambahan agar bisa melakukan koneksi ke sistem lain. Diantara *method* yang mengembalikan tipe *class* pada *class DatabaseInterface* adalah *get_data*, *search_data*, dan *join_table*. Oleh karena itu implementasi ketiga *method* tersebut masih belum didukung pada koneksi melalui D-Bus saat ini.

4.2 Pengujian

Untuk pengujian akan dilakukan dengan 2 skema, yaitu pengujian secara menggunakan bahasa pemrograman yang sama (dalam penelitian ini menggunakan bahasa rust) dan pengujian secara eksternal dengan menggunakan D-Bus yang akan digunakan pengguna untuk menjalankan fungsi *database*. Pengujian terpisah ini dilakukan berdasarkan alasan yang terdapat pada sub bab 4.1.11, yaitu tidak semua *method* di DatabaseInterface dapat diakses oleh pengguna melalui D-Bus.

Untuk memudahkan dalam pelaksanaan ujian, pada file main.rs sebuah *function* baru akan dibuat untuk masing pengujian internal dan pengujian internal dengan data hasil *crawling*. *Function* untuk pengujian internal bernama test dan *function* untuk pengujian internal dengan data dummy bernama test_data_crawling. Masing-masing *function* tersebut akan menerapkan *macro* bernama test. *Macro* ini merupakan *macro* bawaan dari rust. Lalu untuk *function* main dalam main.rs, akan berisi konfigurasi untuk menjalankan *library* zbus untuk menerima koneksi dari luar.

4.2.1 Pengujian Internal

Pengujian internal dilakukan dengan membuat sebuah *class* baru bernama TestDatabaseInterface. Karena pengujian dilakukan maka *method* print pada setiap *class* yang telah dijelaskan di sub bab sebelumnya, akan digunakan. Pengujian akan dilakukan dengan menggunakan data *Dummy* yang terdiri dari 2 tabel yang saling berkaitan. Berikut adalah isi tabel dari data *Dummy* yang digunakan.

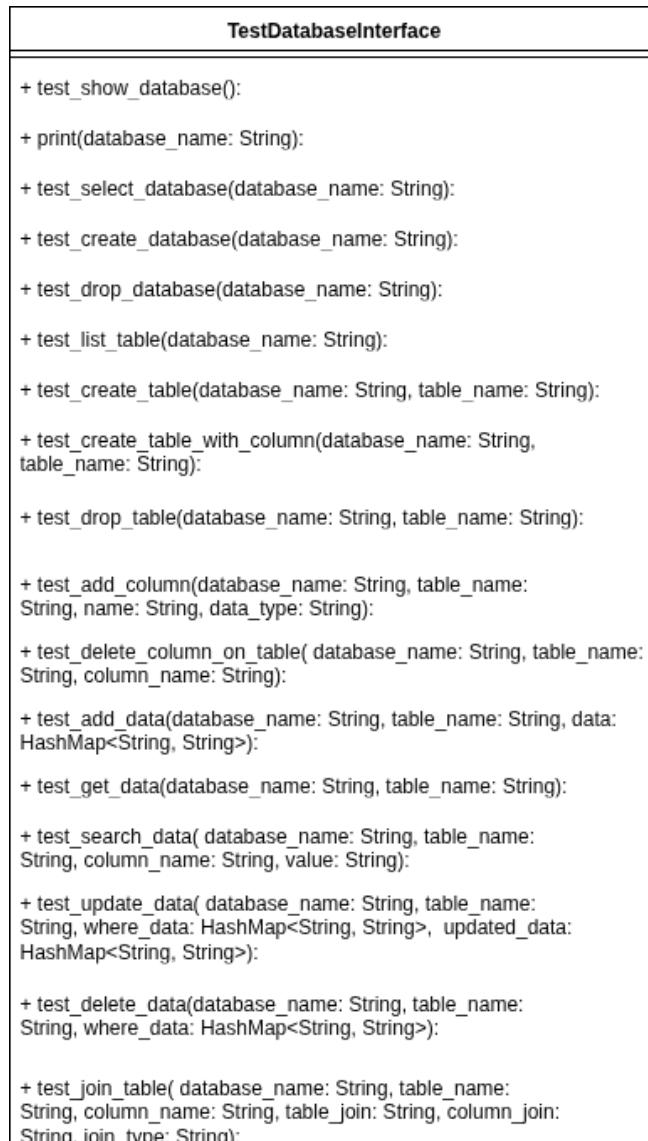
id (Integer)	first_name (String)	last_name (String)
0	Farhan	Abdul
1	Akbar	Maulana
2	Daffa	Haryadi
3	Hanif	Ramadhan
4	Rudiansyah	Wijaya

Tabel 4.1: Tabel Users

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

Tabel 4.2: Tabel Posts

Pada tabel posts, data yang digunakan pada kolom description akan diambil Faker API (*Faker API* 2025) supaya dapat berisi text yang panjang. Lalu untuk kolom user_id pada tabel posts, berguna untuk menghubungkan antara tabel posts dengan tabel users. Pengujian dilakukan pada masing-masing *method* yang ada pada *class DatabaseInterface*. Berikut adalah *class diagram* dari *class TestDatabaseInterface*.



Gambar 4.12: Class Diagram *TestDatabaseInterface*

Pada penulisan ini pengujian akan berfokus pada fungsional dasar *database* yaitu membuat, melihat, mengubah dan menghapus data (CRUD). Perlu diketahui bahwa pengujian yang dilakukan pada masing-masing *function* dibuat dengan isian data yang sama satu sama lain. Lalu terdapat sebuah informasi "Database berhasil terhubung" saat proses pengujian dibawah, dikarenakan proses *database* harus terhubung terlebih dahulu jika ingin menjalankan fungsinya. Jika terdapat tambahan seperti boolean true, juga penanda bahwa proses berhasil dijalankan. Berikut ini adalah hasil pengujian database:

1. test_create_database

Pengujian dimulai dari mencoba melakukan pembuatan *database*. *Database* akan dibuat dengan nama articles. Dengan menjalankan *function* ini maka akan terlihat sebuah *file* baru dalam *folder schema* dan *folder index*. File tersebut memiliki nama yang sama dengan *database* yang dibuat. Pada terminal juga terdapat sebuah informasi bahwa *database* berhasil terbuat.



Gambar 4.13: File baru telah terbuat dalam *folder index* dan *schema*

2. test_select_database

Method yang berguna untuk memilih *database* terlebih dahulu. Karena tanpa melakukan pemilihan database, maka *database engine* tidak akan berjalan. *Database* yang dipilih adalah *database* yang telah dibuat sebelumnya yaitu article. Jika pengujian ini berhasil maka akan muncul informasi bahwa *database* berhasil terhubung.

```

successes:
---- test stdout ----
Database berhasil terhubung

successes:
test
  
```

Gambar 4.14: Informasi *database* telah terhubung

3. test_create_table

Method yang berguna untuk membuat tabel pada *database* yang dituju. Untuk

kasus pengujian ini, vector kosong akan digunakan sebagai input dari kolomnya. Sehingga tabel yang terbuat adalah sebuah tabel tanpa kolom. *Method* ini akan dijalankan 2x untuk menyesuaikan dengan data *Dummy* yang terdapat pada tabel 4.1 dan tabel 4.2. Nilai true pada gambar mengindikasikan bahwa pembuatan tabel telah berhasil.

```
---- test stdout ----
Database berhasil terhubung
Berhasil membuat tabel
Database berhasil terhubung
Berhasil membuat tabel

successes:
    test
```

Gambar 4.15: Informasi tabel telah berhasil terbuat

4. test_add_column

Pengujian yang dilakukan untuk menambah kolom baru pada tabel yang telah dibuat. Kolom-kolom akan ditambahkan sesuai dengan nama dan tipe data yang ada pada tabel 4.1 dan tabel 4.2. Oleh karena itu pengujian akan dilakukan sebanyak 7 kali menyesuaikan dengan jumlah kolom pada tabel yang disebutkan sebelumnya.

```
successes:
---- test stdout ----
Database berhasil terhubung
Berhasil membuat column
```

Gambar 4.16: Informasi kolom telah berhasil terbuat

5. test_add_data

Setelah berhasil membuat kolom, maka pengujian pembuatan dapat dilakukan. *Method* pengujian ini akan membuat sebuah baris data baru di tabel dan kolom

yang sudah dibuat. Jumlah dan isian yang dibuat akan sama seperti data pada tabel 4.1 dan tabel 4.2.

Gambar 4.17: Informasi data telah berhasil terbuat

6. test_get_data

Setelah berhasil membuat data, data yang telah disimpan akan diuji untuk proses pengambilannya. *Method* ini akan menampilkan semua data pada tabel yang dituju dengan menggunakan *method* print di dalam *class* Table.

Gambar 4.18: Hasil data tabel *posts* telah berhasil terbuat

```

2   {
3     id: Integer(0),
4     title: String("Abdu5 9"),
5     description: String(
6       "Quisque fugit voluptates dolor et qui voluptate. Atque veritatis velit modi recidens rerum. Magni voluptate laudantium ipsum est vero expedita aspernatur cum. Quas neque dolores sequi et velit quia sapiente."
7     ),
8     user_id: Integer(0),
9   },
10  {
11    id: Integer(1),
12    title: String("Abdu5 1"),
13    user_id: Integer(0),
14    ad: Integer(2),
15    description: String(
16      "Sed autem sunt deserunt libero voluptas beatissime recusandas excepturi libero. Quae dolores impedit et deserunt. Consequuntur expedita ipsam detectos qui consequatur sunt dolorem."
17    ),
18  },
19  {
20    id: Integer(2),
21    user_id: Integer(1),
22    ad: Integer(2),
23    title: String("Abdu5 2"),
24    description: String(
25      "Sed architecto consequuntur rerum beatae. Inventore et porro ullam omnis eos nam sit id. Provident ducimus explicabo sed nostrum quia laudantium quia."
26    ),
27  },
28  {
29    id: Integer(3),
30    user_id: Integer(2),
31    ad: Integer(3),
32    title: String("Abdu5 3"),
33    description: String(
34      "Et esse eius totam. Et laborum aperiam et maiores est doloremque ut hic. Quis et nesciunt reprehenderit velit."
35    ),
36  },
37  {
38    id: Integer(4),
39    user_id: Integer(2),
40    ad: Integer(3),
41    title: String("Abdu5 4"),
42    description: String(
43      "Et esse eius totam. Et laborum aperiam et maiores est doloremque ut hic. Quis et nesciunt reprehenderit velit."
44    ),
45  },
46  {
47    id: Integer(5),
48    user_id: Integer(2),
49    ad: Integer(3),
50    title: String("Abdu5 5"),
51    description: String(
52      "Et esse eius totam. Et laborum aperiam et maiores est doloremque ut hic. Quis et nesciunt reprehenderit velit."
53    ),
54  },
55  {
56    id: Integer(6),
57    user_id: Integer(3),
58    ad: Integer(3),
59    title: String("Abdu5 6"),
60    description: String(
61      "Et esse eius totam. Et laborum aperiam et maiores est doloremque ut hic. Quis et nesciunt reprehenderit velit."
62    ),
63  },
64  {
65    id: Integer(7),
66    user_id: Integer(3),
67    ad: Integer(3),
68    title: String("Abdu5 7"),
69    description: String(
70      "Et esse eius totam. Et laborum aperiam et maiores est doloremque ut hic. Quis et nesciunt reprehenderit velit."
71    ),
72  },
73  {
74    id: Integer(8),
75    user_id: Integer(2),
76    ad: Integer(3),
77    title: String("Abdu5 8"),
78    description: String(
79      "Et esse eius totam. Et laborum aperiam et maiores est doloremque ut hic. Quis et nesciunt reprehenderit velit."
80    ),
81  },
82  {
83    id: Integer(9),
84    user_id: Integer(2),
85    ad: Integer(3),
86    title: String("Abdu5 9"),
87    description: String(
88      "Et esse eius totam. Et laborum aperiam et maiores est doloremque ut hic. Quis et nesciunt reprehenderit velit."
89    ),
90  },
91  {
92    id: Integer(10),
93    user_id: Integer(1),
94    ad: Integer(2),
95    title: String("Abdu5 10"),
96    description: String(
97      "Et esse eius totam. Et laborum aperiam et maiores est doloremque ut hic. Quis et nesciunt reprehenderit velit."
98    ),
99  },
100  {
101    id: Integer(11),
102    user_id: Integer(1),
103    ad: Integer(2),
104    title: String("Abdu5 11"),
105    description: String(
106      "Et esse eius totam. Et laborum aperiam et maiores est doloremque ut hic. Quis et nesciunt reprehenderit velit."
107    ),
108  },
109  {
110    id: Integer(12),
111    user_id: Integer(1),
112    ad: Integer(2),
113    title: String("Abdu5 12"),
114    description: String(
115      "Et esse eius totam. Et laborum aperiam et maiores est doloremque ut hic. Quis et nesciunt reprehenderit velit."
116    ),
117  };

```

Gambar 4.19: Hasil data tabel *posts* yang telah diformat menggunakan *prettier* (*Extension Visual Studio Code* pada Javascript)

```

.... Test Below.....
Database berhasil terhubung
[{"first_name": "Farhan", "last_name": "String("Abdu5"), "id": Integer(0), ("id": Integer(1), "first_name": String("Abdu5"), "last_name": String("Malana")), ("last_name": "Rudapanya", "first_name": String("Abdu5"))}, {"first_name": "Rudapanya", "last_name": "String("Abdu5")"}, {"first_name": "Wijaya", "last_name": "String("Abdu5")"}]

successes:
test

```

Gambar 4.20: Hasil data tabel *users* telah berhasil terbuat

```

1  [
2    { first_name: String("Farhan"), last_name: String("Abdul"), id: Integer(0) },
3    { id: Integer(1), first_name: String("Akbar"), last_name: String("Maulana") },
4    { last_name: String("Haryadi"), id: Integer(2), first_name: String("Daffa") },
5    {
6      last_name: String("Ramadhan"),
7      first_name: String("Hanif"),
8      id: Integer(3),
9    },
10   {
11     id: Integer(4),
12     first_name: String("Rudiansyah"),
13     last_name: String("Wijaya"),
14   },
15 ];
16

```

Gambar 4.21: Hasil data tabel *users* yang telah diformat menggunakan *prettier* (*Extension Visual Studio Code* pada Javascript)

7. test_update_data

Method pengujian ini akan melakukan pengubahan pada data yang telah disimpan di *database*. Pengujian ini membutuhkan *parameter* untuk menentukan data mana yang akan dihapus, dan data apa saja yang akan diubah. Untuk kasus uji kali ini, akan mengubah kolom pada tabel *users* yang memiliki *id* dengan value 0 dan *first_name* yang memiliki value "Farhan". Sementara data yang diubah adalah kolom *last_name*, yang akan diubah berisi "Abdul Hamid".

```

---- test stdout ----
Database berhasil terhubung
true
Berhasil mengubah data

successes:
  test

```

Gambar 4.22: Informasi data berhasil diubah

8. test_search_data

Pengujian *method* search data ini dilakukan dengan memasukkan kolom dan

value yang dituju. Data yang dicari adalah data pada tabel users yang memiliki id 0.

```
---- test stdout ----
Database berhasil terhubung
[{"last_name": String("Abdul Hamid"), "first_name": String("Farhan"), "id": Integer(0)}]

successes:
    test
```

Gambar 4.23: Informasi hasil pencarian

```
1  [
2  {
3      last_name: String("Abdul Hamid"),
4      first_name: String("Farhan"),
5      id: Integer(0),
6  },
7 ];
```

Gambar 4.24: Hasil data yang telah diformat menggunakan *prettier* (*Extension VSCode* pada Javascript)

9. test_join_data

Untuk menjalankan pengujian ini, kolom user_id pada tabel posts akan digunakan untuk menyocokan data ke tabel users. Tabel, kolom serta tipe *join* harus didefinisikan untuk menjalankan pengujian *join* ini. Pada saat melakukan pengujian ini, terdapat beberapa error yang dialami dikarenakan *index* yang tidak valid serta kesalahan dalam pengondisionan *join* nya. Namun untuk gambar dibawah, merupakan hasil dari implementasi yang telah diperbaiki.

```
successes:
---- test stdout ----
Database berhasil terhubung
[{"last_name": String("Abdul Hamid"), "title": String("Judul 0"), "description": String("Maxime
rum. Magni voluptate laudantium ipsam est vero expedita aspernatur cum. Quas neque dolores sapie
rst_name": String("Farhan")), {"first_name": String("Farhan"), "last_name": String("Abdul Hamid")
equatur aut deserunt libero voluptas beatae recusandae excepturi libero. Quas dolorem impedit et
"id": Integer(0)}, {"user_id": Integer(1), "description": String("Sei architecto consequuntur r
licabo sed nostrum quia laudantium quia."), "id": Integer(1), "title": String("Judul 2"), "first
"description": String("Cum et autem ut laudantium eum quas. Veritatis molestias aliquam est ut
```

Gambar 4.25: Informasi hasil *inner join*

```

1 | /**
2 |  * User entity
3 |  */
4 | @Entity
5 | @Table(name = "User")
6 | public class User {
7 |     @Id
8 |     @GeneratedValue(strategy = GenerationType.IDENTITY)
9 |     private Long id;
10 |     private String first_name;
11 |     private String last_name;
12 |     private Integer user_id;
13 |     private String title;
14 |     private String description;
15 |     private String bio;
16 |     private String email;
17 |     private String phone;
18 |     private String address;
19 |     private Boolean is_deleted;
20 |     private Boolean is_active;
21 |     private Boolean is_email_verified;
22 |     private Boolean is_phone_verified;
23 |     private Boolean is_password_reset;
24 |     private Boolean is_email_change;
25 |     private Boolean is_phone_change;
26 |     private Boolean is_email_change;
27 |     private Boolean is_password_change;
28 |     private Boolean is_email_change;
29 |     private Boolean is_password_change;
30 |     private Boolean is_email_change;
31 |     private Boolean is_password_change;
32 |     private Boolean is_email_change;
33 |     private Boolean is_password_change;
34 |     private Boolean is_email_change;
35 |     private Boolean is_password_change;
36 |     private Boolean is_email_change;
37 |     private Boolean is_password_change;
38 |     private Boolean is_email_change;
39 |     private Boolean is_password_change;
40 |     private Boolean is_email_change;
41 |     private Boolean is_password_change;
42 |     private Boolean is_email_change;
43 |     private Boolean is_password_change;
44 |     private Boolean is_email_change;
45 |     private Boolean is_password_change;
46 |     private Boolean is_email_change;
47 |     private Boolean is_password_change;
48 |     private Boolean is_email_change;
49 |     private Boolean is_password_change;
50 |     private Boolean is_email_change;
51 |     private Boolean is_password_change;
52 |     private Boolean is_email_change;
53 |     private Boolean is_password_change;
54 |     private Boolean is_email_change;
55 |     private Boolean is_password_change;
56 |     private Boolean is_email_change;
57 |     private Boolean is_password_change;
58 |     private Boolean is_email_change;
59 |     private Boolean is_password_change;
60 |     private Boolean is_email_change;
61 |     private Boolean is_password_change;
62 |     private Boolean is_email_change;
63 |     private Boolean is_password_change;
64 |     private Boolean is_email_change;
65 |     private Boolean is_password_change;
66 |     private Boolean is_email_change;
67 |     private Boolean is_password_change;
68 |     private Boolean is_email_change;
69 |     private Boolean is_password_change;
70 |     private Boolean is_email_change;
71 |     private Boolean is_password_change;
72 |     private Boolean is_email_change;
73 |     private Boolean is_password_change;
74 |     private Boolean is_email_change;
75 |     private Boolean is_password_change;
76 |     private Boolean is_email_change;
77 |     private Boolean is_password_change;
78 |     private Boolean is_email_change;
79 |     private Boolean is_password_change;
80 |     private Boolean is_email_change;
81 |     private Boolean is_password_change;
82 |     private Boolean is_email_change;
83 |     private Boolean is_password_change;
84 |     private Boolean is_email_change;
85 |     private Boolean is_password_change;
86 |     private Boolean is_email_change;
87 |     private Boolean is_password_change;
88 |     private Boolean is_email_change;
89 |     private Boolean is_password_change;
90 |     private Boolean is_email_change;
91 |     private Boolean is_password_change;
92 |     private Boolean is_email_change;
93 |     private Boolean is_password_change;
94 |     private Boolean is_email_change;
95 |     private Boolean is_password_change;
96 |     private Boolean is_email_change;
97 |     private Boolean is_password_change;
98 |     private Boolean is_email_change;
99 |     private Boolean is_password_change;
100 |     private Boolean is_email_change;
101 |     private Boolean is_password_change;
102 |     private Boolean is_email_change;
103 |     private Boolean is_password_change;
104 |     private Boolean is_email_change;
105 |     private Boolean is_password_change;
106 |     private Boolean is_email_change;
107 |     private Boolean is_password_change;
108 |     private Boolean is_email_change;
109 |     private Boolean is_password_change;
110 |     private Boolean is_email_change;
111 |     private Boolean is_password_change;
112 |     private Boolean is_email_change;
113 |     private Boolean is_password_change;
114 |     private Boolean is_email_change;
115 |     private Boolean is_password_change;
116 |     private Boolean is_email_change;
117 |     private Boolean is_password_change;
118 |     private Boolean is_email_change;
119 |     private Boolean is_password_change;
120 |     private Boolean is_email_change;
121 |     private Boolean is_password_change;
122 |     private Boolean is_email_change;
123 |     private Boolean is_password_change;
124 |     private Boolean is_email_change;
125 |     private Boolean is_password_change;
126 |     private Boolean is_email_change;
127 |     private Boolean is_password_change;
128 |     private Boolean is_email_change;
129 |     private Boolean is_password_change;
130 |     private Boolean is_email_change;
131 |     private Boolean is_password_change;
132 | }

```

Gambar 4.26: Hasil data yang telah diformat menggunakan *prettier* (*Extension VSCode pada Javascript*)

```
---- test stdout ----
Database berhasil terhubung
[{"id": Integer(0), "first name": String("Farhan"), "user id": Integer(0), "last name": String("Sed architecto consequuntur reatum beatae. Inventore et porrumpit omnis eos nam sunt ut quo aut debitis totam. Et laborum aperient et maiores est dolorem hic. Quod last name": String("Haryadi"), "user id": Integer(2), "first name": String("Daffa")), {"title": "String("Totam quidem cum reprehenderit rerum et consequatur soluta. Repudianduugiat. Maiores expedita quaerat odit minus nulla. Et reprehenderit repellendus eos temporibus : String("Wijaya"), "id": Integer(4), "first name": String("Rudiansyah")}]
```

Gambar 4.27: Informasi hasil *left join*

```

2   [
3     {
4       id: Integer(0),
5       first_name: String("Farhan"),
6       user_id: Integer(0),
7       last_name: String("Abdul Hamid"),
8       title: String("Judul 0"),
9       description: String("Maxime fugit voluptatem dolor et qui voluptate. Atque veritatis velit modi reciendis rerum. Magni voluptate laudantium ipsum est vero expedita aspernatur cum. Quas neque dolores sapiente sequi et velit quia sapiente."),
10      },
11    ],
12   [
13     {
14       id: Integer(1),
15       title: String("Judul 2"),
16       last_name: String("Maulan"),
17       first_name: String("Akbar"),
18       user_id: Integer(1),
19       description: String("Sed architecto consequatur rerum beatae. Inventore et parva ulma omnis eos nam sit id. Provident ducimus explicabo sed nostrum quia laudantium quia."),
20     },
21   ],
22   [
23     {
24       description: String("Ut quo aut debitis totam. Et laborum aperiam et maiores est doloresque ut hic. Quo et nescient reprehenderit velit."),
25       id: Integer(1),
26       title: String("Judul 3"),
27       id: Integer(1),
28       last_name: String("Mulyadi"),
29       user_id: Integer(2),
30       first_name: String("Daffa"),
31     },
32   ],
33   [
34     {
35       title: String("Judul 4"),
36       last_name: String("Wulan"),
37       first_name: String("Wulan"),
38       description: String("Reprehenderit reprenderit rerum et consequatur soluta. Repudiandae dolares assumenda sed ex quo. Aspernatur quis perspicillat omnis tempore optio fugiat. Maiores expedita querat edit minus nulla. Et reprehenderit repellendus eos temporibus reprehenderit nam libero."),
39     },
40   ],
41   [
42     {
43       last_name: String("Wijaya"),
44       id: Integer(4),
45       first_name: String("Nodiansyah"),
46     },
47   ];

```

Gambar 4.28: Hasil data yang telah diformat menggunakan *prettier (Extension VSCode pada Javascript)*

```

successes:
---- test stdout ----
Database berhasil terhubung
[{"description": String("Maxime fugit voluptatem dolor et qui voluptate. Atque veritatis velit ur cum. Quas neque dolores sapiente sequi et velit quia sapiente."), "user_id": Integer(0), "f ame": String("Abdul Hamid")}, {"user_id": Integer(0), "first_name": String("Farhan"), "title": ion": String("Consequatur aut deserunt libero voluptas beatae recusandae excepturi libero. Qua r sunt dolorem."), "id": Integer(2), "title": String("Judul 2"), "description": String("Sed . Provident ducimus explicabo sed nostrum quia laudantium quia."), "last_name": String("Maulan 3"), "description": String("Ut quo aut debitis totam. Et laborum aperiam et maiores est dolor")];

```

Gambar 4.29: Informasi hasil *right join*

Gambar 4.30: Hasil data yang telah diformat menggunakan *prettier* (*Extension VSCode pada Javascript*)

10. test_delete_data

Pengujian *method* delete data membutuhkan data mana saja yang akan dihapus, tentunya dengan menggunakan kolom tersedia di dalam tabel. Untuk kasus kali ini, data pada tabel users akan dicoba untuk dihapus dengan ketentuan kolom first_name yang memiliki value "Rudiansyah".



Gambar 4.31: Informasi hasil penghapusan data

```

    test
    database berikut terhubung
    [{"id": Integer(0), "first_name": String("Farhan"), "last_name": String("Abdul Hamid")}, {"id": Integer(1), "first_name": String("Akbar"), "last_name": String("Maulana")}, {"id": Integer(2), "last_name": String("Haryadi"), "first_name": String("Daffa")}, {"id": Integer(3), "first_name": String("Hanif")}]
}
successes:
Test

```

Gambar 4.32: Data tabel *users* setelah value "Rudiansyah" dihapus

```

1  [
2    {
3      id: Integer(0),
4      first_name: String("Farhan"),
5      last_name: String("Abdul Hamid"),
6    },
7    { id: Integer(1), first_name: String("Akbar"), last_name: String("Maulana") },
8    { id: Integer(2), last_name: String("Haryadi"), first_name: String("Daffa") },
9    {
10      last_name: String("Ramadhan"),
11      id: Integer(3),
12      first_name: String("Hanif"),
13    },
14  ];
15

```

Gambar 4.33: Data tabel *users* yang telah diformat menggunakan *prettier* (*Extension Visual Studio Code* pada Javascript)

Return pada setiap *method* didalam *DatabaseInterface* pada pengujian diatas tidak memengaruhi hasil pengujian karena untuk menampilkan data, pengujian menggunakan *method* print dari struct yang telah dibuat. Lalu tidak semua *method* yang ada pada *class TestDatabaseInterface* digunakan selama pengujian.

4.2.2 Pengujian Eksternal

Untuk melakukan pengujian eksternal diperlukan sebuah *client* untuk memanggil *interface* D-Bus yang telah dibuka. Pada pengujian kali ini, bahasa pemrograman python akan digunakan beserta dengan library dbus-next. Pengujian akan dilakukan pada proses pembuatan, pengubahan dan penghapusan. Untuk pengambilan data masih belum bisa diuji sebagaimana yang telah di kutip pada sub bab 4.1.11. Berikut adalah isian dari program *client* menggunakan bahasa pemrograman python.

```

1  from dbus_next.aio import MessageBus
2
3  import asyncio
4
5  loop = asyncio.get_event_loop()
6
7  async def main():
8      bus = await MessageBus().connect()
9      introspection = await bus.introspect('org.two.DatabaseConnection', '/org/two/DatabaseConnection')
10     obj = bus.get_proxy_object('org.two.DatabaseConnection', '/org/two/DatabaseConnection', introspection)
11     connection = obj.get_interface('org.two.DatabaseConnection')
12
13     res = await connection.call_create_database("tests") # Membuat database test
14     res = await connection.call_select_database("tests") # Select database test
15     res = await connection.call_drop_database("tests") # Mencoba menghapus database test
16
17     res = await connection.call_create_database("library") # Membuat database
18     res = await connection.call_select_database("library") # Select database
19     res = await connection.call_create_table("books") # Membuat table
20     res = await connection.call_list_table() # Melihat list table
21     res = await connection.call_add_column("books", "id", "integer") # Membuat Kolom baru
22     res = await connection.call_add_column("books", "name", "string") # Membuat Kolom baru
23     res = await connection.call_delete_column("books", "author", "string") # Membuat Kolom baru
24     res = await connection.call_list_column("books", "author") # Membuat kolom author sebagai bahan uji penghapusan kolom
25     res = await connection.call_list_column("books") # Melihat list kolom pada tabel book
26     res = await connection.call_add_data("books", {'id': '0', "name": "Farhan"}) # Membuat data baru pada tabel books
27
28     print(res)
29
30 loop.run_until_complete(main())

```

Gambar 4.34: Isi program pengujian *client* dengan bahasa pemrograman Python



```

successes:

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

()

```

Gambar 4.35: Informasi hasil pengujian client

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*.

4.2.3 Pengujian Internal dengan Data Hasil *Crawling*

Database Engine yang dikembangkan pada saat ini ditujukan untuk pengembangan sistem *crawling* pada penelitian terakhir (Rizqillah 2024). Karena alasan tersebut, pengujian menggunakan data hasil *crawling* pada penelitian sebelumnya (Rizqillah 2024) akan dilakukan. Untuk pengujian tidak semua tabel akan digunakan, hanya 2 tabel hasil *crawling* saja yaitu tabel bernama *page_information* dan *crawling*. Dalam tabel tersebut terdapat berbagai macam data, namun dikarenakan pada database saat ini hanya mendukung 2 tipe data yaitu *string* dan *integer*, maka untuk tipe data lain diluar dari 2 tipe data tersebut, akan dirubah menjadi *string*. Sebanyak 10 baris dari tabel *page_information* akan digunakan, data tersebut dapat dilihat pada file *crawl_information.json* dalam folder *test-data*. Proses pengujian dilakukan dengan tahapan berikut:

1. Membuat *database*
2. Membuat *table* *crawling* dengan *table* *page_information*
3. Melengkapi kolom pada *table* *crawling*
4. Mengisi data pada *table* *crawling*
5. Melengkapi kolom pada *table* *page_information*
6. Mengisi data pada *table* *page_information*
7. Menampilkan data yang telah terisi

Dari hasil pengujian tersebut didapatkan bahwa, penyimpanan dan pengambilan dapat berjalan dengan baik. Namun hal tersebut hanya berlaku untuk huruf latin, untuk huruf-huruf bahasa lain seperti bahasa china, jepang, arab dan lainnya masih belum dapat tersimpan dengan benar. Permasalahan ini harus ditelusuri lebih lanjut, namun ada kemungkinan bahwa penyimpanan 1 huruf selain huruf latin memiliki ukuran yang berbeda dengan huruf latin. Berikut adalah contoh kesalahan penyimpanan dan penampilan data pada bahasa non latin:

```

2024-02-04 22:21:46
Data 12:
18511
1
"https://developer.mozilla.org/en-US/docs/Glossary/ATAG"
"ATAG - MDN Web Docs Glossary: Definitions of Web-related terms | MDN"
"ATAG (Authoring Tool Accessibility Guidelines) is a W3C recommendation for building accessible-authoring tools that produce accessible contents."
"ATAG ATAG (Authoring Tool Accessibility Guidelines) is a W3C recommendation for building accessible-authoring tools that produce accessible contents. See also ATAG as part of the Web Accessibility Initiative on Wikipedia Authoring Tool Accessibility Guidelines (ATAG) Overview The ATAG 2.0 recommendation Found a content problem with this page? Edit the page on GitHub . Report the content issue . View the source on GitHub . Want to get more involved? Learn how to contribute . This page was last modified on Jun 8, 2023 by MDN contributors ."
109081
"8F5 crawling"
"0:00:10.891102"
"2024-02-04 22:21:46"
()
```

Gambar 4.36: Informasi hasil pengujian data *crawling*

4.2.4 Hasil Implementasi

Dari implementasi dan pengujian yang dilakukan pada sub bab sebelumnya, maka 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. Pemrosesan tersebut mulai dari mengambil data, menyimpan data, mengubah data dan menghapus data. Dari hasil pengujian pun dapat terlihat bahwa *database* telah berhasil disimpan secara persisten dalam sebuah sebuah *file*. Tidak hanya itu, pola dan metode penyimpanan pun dapat terlihat dari urutan *byte* yang disimpan pada *file*. Dengan begitu pengembang ke depannya dapat mengubah dan menyesuaikan pola pemrosesan data di dalam *database* baru ini jika menemukan pola atau algoritma yang lebih efisien. Terlebih jika di kedepan hari *database engine* ini akan diimplementasikan pada *distributed database*, maka algoritma sinkronisasi dapat di terapkan langsung di dalam *class Schema* yang telah dibuat. Algoritma yang dipilih dikembalikan kepada pengembang *database engine* ini di kemudian hari.

Meski demikian masih terdapat banyak fitur dari *database* ini yang belum berjalan dengan baik dan memerlukan penyempurnaan. Berikut adalah hal-hal yang dapat disempurnakan pada fitur *database engine* ini, yaitu:

1. Melengkapi metode-metode eksternal untuk *client*

Belum semua *method* yang ada pada *DatabaseInterface* terimplementasi pada *DatabaseConnection*, sehingga membuat *client* belum bisa menggunakan *database engine* sepenuhnya.

2. Memperbaiki return *indexing* pada *join*

Dengan menerapkan *indexing* yang ada pada *database engine* saat ini, maka fitur *join* tabel masih belum bisa berjalan dikarenakan tidak dapat

mengembalikan data ke *DatabaseConnection*.

3. Menambahkan utilitas pada fitur *join*

Membuat fitur *joining* untuk 3 tabel atau lebih, serta membuat constraint pada column yang menjadi key antar tabel.

4. Melihat metadata pada column di *database*

Saat ini fitur untuk melihat list column, baru mengembalikan nama kolom saja dari tabel.

5. Menambahkan metadata pada *database* dan *table*

Metadata yang umumnya ada pada fitur *database management* seperti `created_at`, `increment`, `uniqueId` dan lainnya.

6. Menambahkan tipe data lain

Menambahkan tipe data lain karena saat ini hanya bisa mendukung 2 tipe data yaitu `String` dan `Integer`.

7. Pengubahan default value

Menambahkan perubahan nilai default pada cell yang ada di *database* agar nilai default dari sebuah cell dapat diubah.

8. Membuat multi connection

Saat ini koneksi yang dapat ditangani hanya 1 koneksi, jika digunakan untuk 2 koneksi *client*, maka dikhawatirkan terdapat data yang tidak sinkron.

9. Membuat kompatibilitas terhadap bahasa non latin

Pada sistem *database engine* yang berjalan saat ini, masih belum dapat menyimpan dan menampilkan data dengan bahasa non latin. Kemungkinan hal ini terjadi dikarenakan bahasa non latin memiliki jumlah *byte* yang berbeda untuk satu karakter dibanding dengan satu karakter huruf latin.

10. Pembuatan *folder schema* dan *index* secara otomatis

Setelah melakukan proses *build*, jika hasil *build database engine* ingin dikirim ke perangkat lain, maka untuk saat ini *folder schema* dan *folder index* harus dibuat manual.

11. Memperbaiki dan menyempurnakan penanganan *error*

Error yang ada pada saat ini hanya bisa dilihat dari server dan belum di handle

dengan baik pada *client*. Hal ini dapat menyebabkan *client* tidak tau pasti penyebab error. Terdapat beberapa *error* lain seperti saat hendak melakukan pembuatan database tanpa adanya *folder schema* dan *index*. Jika tidak ada kedua *folder* tersebut maka database tidak terbuat, dan langsung memunculkan pesan "*Database Created!*". Tentunya pesan yang disampaikan tidaklah sesuai, hal ini dikarenakan penanganan *error* di *Database Interface* masih belum baik. Tidak menutup kemungkinan bahwa masih terdapat penanganan *error* lain yang harus ditingkatkan seperti jika path dari D-Bus sudah dipakai oleh sistem lain dan lainnya.

4.3 Sumber dan Orisinalitas *code*

Dalam pengembangan *database engine*, berbagai macam sumber *code* pihak lain digunakan sebagai referensi. Sumber-sumber tersebut dari beberapa *website* serta dokumentasi langsung bahasa pemrograman dan *library* yang digunakan selama pengembangan. Salah satu bagian penting pada pengembangan yang didapatkan dari referensi lain adalah proses perubahan dari sebuah data menjadi bentuk byte serta mengembalikannya dalam bahasa pemrograman rust. Hasil akhir dari *source code* dapat dilihat pada *link* berikut.

<https://github.com/arhandev11/database-engine>

Selain itu, sumber untuk melakukan penerapan metode *waterfall* juga digunakan sebagai pembanding dengan penulisan yang dibuat. Hal ini dilakukan karena selama pengembangan berlangsung masih belum menggunakan metode *waterfall*. Sumber lain seperti dokumentasi resmi postgresql dan lainnya digunakan sebagai sumber pengetahuan dan pertimbangan dalam pengembangan *database engine*. Penggunaan *Artificial Intelligence* dengan model *Gemini 2.5 pro* dan *Chatgpt (Versi o4-mini-high, 4.1 dan 5)* juga digunakan dalam memahami isi *paper* serta dalam membantu membuat penulisan dan *formatting* menggunakan *latex*.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Berdasarkan hasil implementasi dan pengujian yang sudah dijalankan maka terdapat beberapa kesimpulan sebagai berikut:

1. Untuk membuat *database engine* yang menjaga integritas data, maka harus memikirkan bagaimana cara database tersebut disimpan pada *filesystem*. Jenis data yang disimpan juga harus dipikirkan untuk memastikan data yang telah berubah menjadi *byte* dapat kembali menjadi nilai asli dan tipe aslinya walaupun pada fitur saat ini masih memiliki keterbatasan dalam pengembalian datanya.
2. Penyimpanan data pada *filesystem* harus konsisten agar tidak terdapat data yang hilang. Karena berbeda 1 urutan saja dapat menghancurkan data-data yang tersimpan diurutan lainnya.
3. Fitur-fitur untuk mengolah data yang disimpan pada *filesystem* dapat dikelola oleh atribut dan *method* pada bahasa pemrograman yang digunakan. Sehingga harus mempertimbangkan *method* yang mana saja yang dapat di akses oleh *client*.
4. Tipe data yang terdapat pada *database* harus kompatibel untuk berbagai bahasa untuk memastikan antara server dan *client* dapat saling tukar menukar data.
5. Implementasi algoritma sinkronisasi sangat memungkinkan untuk diterapkan di dalam sistem *database* karena alur awal penerimaan data sampai penyimpanan data dapat dilihat secara langsung. Tak hanya algoritma, metode penyimpanan pada *filesystem* dan *index* juga bisa lebih dikembangkan lagi untuk mencari performa yang paling optimal.
6. Penanganan penyimpanan data untuk bahasa non latin memiliki tahapan yang berbeda dikarenakan cara yang diterapkan pada saat ini masih belum berhasil.

5.2 Saran

Berdasarkan hasil implementasi dan pengujian yang sudah dijalankan, terdapat beberapa saran untuk menyempurnakan *database engine* di kemudian hari, yaitu:

1. Mencoba memperbaiki kekurangan pada *database engine* sesuai dengan yang dibahas pada sub bab 4.2.4. Tujuannya adalah agar *database engine* yang telah dibuat saat ini dapat diimplementasikan pada sistem lain yang sudah berjalan.
2. Mengimplementasikan fitur distribusi *database*, agar dapat segera digunakan pada sistem *crawling* untuk *search engine*. Salah satu langkah utamanya adalah dengan mencari algoritma sinkronisasi yang paling optimal dan efisien yang dapat diterapkan pada *database engine* saat ini.
3. Implementasikan lebih dalam fitur-fitur koneksi yang ada pada D-Bus agar komunikasi ke *client* dapat berjalan lebih baik.
4. Sebelum melanjutkan pengembangan *database*, sangat disarankan untuk mempelajari konsep dan paradigma yang ada di bahasa pemrograman rust agar dapat mengimplementasikan fitur-fitur lain menjadi lebih baik ke depannya.

DAFTAR PUSTAKA

- Ahmed, I. (2025). *Convert a struct to byte array and back in rust*. URL: <https://stackoverflow.com/questions/66922989/convert-a-struct-to-byte-array-and-back-in-rust> (Diakses pada 08/12/2025).
- Bincode* (2025). URL: <https://docs.rs/bincode/latest/bincode/> (Diakses pada 08/12/2025).
- dbus-next* (2025). URL: <https://github.com/altdesktop/python-dbus-next> (Diakses pada 08/12/2025).
- DeBarros, A. (2022). *Practical SQL, 2nd edition*.
- Faker API* (2025). URL: <https://fakerapi.it/> (Diakses pada 08/12/2025).
- Feature Matrix* (2025). URL: <https://www.postgresql.org/about/featurematrix/> (Diakses pada 08/12/2025).
- freedesktop (2022). *dbus*. URL: <https://www.freedesktop.org/wiki/Software/dbus/>.
- Garcia-Molina, H., J. D. Ullman, and J. Widom (2008). *Database System - The Complete Book - Second Edition*.
- Gyorodi, C. et al., (June 2015). “A Comparative Study: MongoDB vs. MySQL”. In: Institute of Electrical and Electronics Engineers.
- Győrödi, C. A. et al., (Sept. 2023). “Implementing a Synchronization Method between a Relational and a Non-Relational Database”. In: *Big Data and Cognitive Computing* 7 (3).
- Intersection of Hashsets* (2025). URL: <https://users.rust-lang.org/t/intersection-of-hashsets/32351/2> (Diakses pada 08/12/2025).
- Khatulistiwa, L. (2023). “Perancangan Arsitektur Search Engine dengan Mengintegrasikan Web Crawler, Algoritma Page Ranking, dan Document Ranking”. In.
- Limited, I. E. S. (2008). *Introduction to Database System*.
- Liu, Y., Y. Wang, and Y. Jin (2012). “Research on The Improvement of MongoDB Auto-Sharding in Cloud Environment”. In.
- Meilinaeka (2025). *Metode Waterfall dalam Pengembangan Perangkat Lunak*. URL: <https://it.telkomuniversity.ac.id/metode-waterfall-dalam-pengembangan-perangkat-lunak/> (Diakses pada 08/12/2025).
- Michael T. Goodrich, R. T. (2014). *Data Structures and Algorithms in Java, 6th Edition*.
- MongoDB Licensing* (2025). URL: <https://www.mongodb.com/legal/licensing/community-edition> (Diakses pada 08/12/2025).
- Ntoso, M. K. et al., (2025). *Computer Systems A - Data Representation and Computer Structure*. URL: https://curriculumresources.edu.gh/wp-content/uploads/2024/10/Computing-Section-1_-TV.pdf (Diakses pada 08/12/2025).

- Petrox, A. (2019). *Database Internals*.
- Postgresql Source Code* (2025). URL: <https://github.com/postgres/postgres/tree/master> (Diakses pada 08/12/2025).
- Python Dictionaries Official Documentation* (2025). URL: <https://docs.python.org/3/tutorial/datastructures.html#dictionaries> (Diakses pada 08/12/2025).
- Qoriiba Muhammad, F. (2021). “Perancangan Crawler Sebagai Pendukung pada Search Engine”. In.
- Returning different types in match arms* (2025). URL: <https://users.rust-lang.org/t/returning-different-types-in-match-arms/73508/3> (Diakses pada 08/12/2025).
- Rizqillah, R. (2024). “IMPROVISASI CRAWLING PADA PETA WEB MENGGUNAKAN ALGORITMA TERDISTRIBUSI DENGAN MODEL KOORDINASI BERBASIS SOCKET PROGRAMMING”. In.
- Rust Official Documentation* (2025). URL: <https://doc.rust-lang.org/stable/reference/> (Diakses pada 08/12/2025).
- Rust Official Website* (2025). URL: <https://www.rust-lang.org/> (Diakses pada 08/12/2025).
- Serde* (2025). URL: <https://serde.rs/> (Diakses pada 08/12/2025).
- Shepmaster (2025). *How to convert 'struct' to '&[u8]'?* URL: <https://stackoverflow.com/questions/28127165/how-to-convert-struct-to-u8> (Diakses pada 08/12/2025).
- Tomar, P. and Megha (2014). “An Overview of Distributed Databases”. In: 4.2, pp. 207–214.
- Watt, A. (2014). *Database Design - 2nd Edition*.
- Wengrow, J. (2020). *A Common-Sense Guide to Data Structures and Algorithms, Second Edition, 2nd Edition*.
- Why am I Getting a "bad file descriptor" error when writing file in rust?* (2025). URL: <https://stackoverflow.com/questions/65862510/why-am-i-getting-a-bad-file-descriptor-error-when-writing-file-in-rust> (Diakses pada 08/12/2025).
- Xu, F. et al., (Dec. 2019). “Design and implementation of heterogeneous relational database synchronization mechanism based on tree distribution architecture”. In: *Proceedings - 2019 6th International Conference on Information Science and Control Engineering, ICISCE 2019*. Institute of Electrical and Electronics Engineers, pp. 234–240.
- zbus* (2025). URL: <https://dbus2.github.io/zbus/> (Diakses pada 08/12/2025).
- Zhang, Y. et al., (Oct. 2022). “Towards Understanding the Runtime Performance of Rust”. In: Institute of Electrical and Electronics Engineers.

LAMPIRAN A

Dokumentasi Source Code

```
1 use std::{
2     collections::{HashMap, HashSet},
3     fs::File,
4     future::pending,
5     io::{ErrorKind, Write},
6     result,
7 };
8
9 use crate::database::{
10     column::Column,
11     database_connection::DatabaseConnection,
12     database_interface::DatabaseInterface,
13     schema::Schema,
14     table::{self, Table},
15     test_interface::TestDatabaseInterface,
16     utils::{self, DataType, InputDataEnum},
17 };
18 use zbus::{interface, Connection, Result};
19
20 mod database;
```

Gambar 1.1: Import dalam file main.rs

```

1  #[tokio::main]
2  async fn main() -> Result<()> {
3      let database_connection: DatabaseConnection = DatabaseConnection {
4          db_interface: DatabaseInterface {
5              is_connect: false,
6              database: None,
7          },
8      };
9
10     let connection: Connection = Connection::session().await?;
11
12     connection
13         .object_server()
14         .at("/org/two/DatabaseConnection", database_connection)
15         .await?;
16
17     connection
18         .request_name("org.two.DatabaseConnection")
19         .await?;
20
21     loop {
22         pending:::<()>().await;
23     }
24 }
25
26 // DAFTAR PUSTAKA CODE / REFERENSI YANG DIGUNAKAN DALAM CODE
27 // https://stackoverflow.com/questions/28127165/how-to-convert-struct-to-u8
28 // https://stackoverflow.com/questions/42499049/how-to-transmute-a-u8-buffer-to-struct-in-rust
29 // https://stackoverflow.com/questions/25410028/how-to-read-a-struct-from-a-file-in-rust
30 // https://stackoverflow.com/questions/66922989/convert-a-struct-to-byte-array-and-back-in-rust
31 // https://stackoverflow.com/questions/59600739/access-struct-field-by-variable
32 // https://curriculumresources.edu.gihwp-content/uploads/2024/10/Computing-Section-1_TV.pdf
33 // https://users.rust-lang.org/t/returning-different-types-in-match-arms/73508
34 // https://doc.rust-lang.org/reference/items/enumerations.html
35 // https://users.rust-lang.org/t/intersection-of-hashesets/32351/2
36 // https://stackoverflow.com/questions/65862510/why-am-i-getting-a-bad-file-descriptor-error-when-writing-file-in-rust
37 // https://docs.python.org/3/tutorial/datastructures.html#dictionaries
38

```

Gambar 1.2: Function main dalam file main.rs

Gambar 1.3: Function test dalam file main.rs bagian 1

Gambar 1.4: Function test dalam file main.rs bagian 2

```

1 fn test_data_crawling() {
2     TestDatabaseInterface::test.create_database("crawling".to_string());
3     TestDatabaseInterface::test.select_database("crawling".to_string());
4     TestDatabaseInterface::test.create_table("crawling",to_string(), "crawling".to_string());
5     TestDatabaseInterface::test.create_table("crawling",to_stringto_string());
6
7     // Table Crawling
8     TestDatabaseInterface::test.add_column(
9         "crawling",to_string(),
10        "crawling".to_string(),
11        "id_crawling".to_string(),
12        "int".to_string(),
13        "string".to_string());
14     TestDatabaseInterface::test.add_column(
15        "crawling",to_string(),
16        "crawling".to_string(),
17        "crawl_id".to_string(),
18        "int".to_string(),
19        "string".to_string());
20     TestDatabaseInterface::test.add_column(
21        "crawling",to_string(),
22        "crawling".to_string(),
23        "crawl_time".to_string(),
24        "keyword".to_string(),
25        "string".to_string());
26     TestDatabaseInterface::test.add_column(
27        "crawling",to_string(),
28        "crawling".to_string(),
29        "total_page".to_string(),
30        "integer".to_string());
31     TestDatabaseInterface::test.add_column(
32        "crawling",to_string(),
33        "crawling".to_string(),
34        "duration_crawl".to_string(),
35        "string".to_string());
36     TestDatabaseInterface::test.add_column(
37        "crawling",to_string(),
38        "crawling".to_string(),
39        "url".to_string(),
40        "crawling".to_string(),
41        "crawling".to_string(),
42        "crawl_id".to_string(),
43        "string".to_string());
44     TestDatabaseInterface::test.add_column(
45        "crawling",to_string(),
46        "crawling".to_string(),
47        "crawl_id".to_string(),
48        "page_information".to_string(),
49        "id_crawl".to_string(),
50        "integer".to_string());
51     TestDatabaseInterface::test.add_column(
52        "crawling",to_string(),
53        "crawling".to_string(),
54        "page_information".to_string(),
55        "crawl_id".to_string(),
56        "integer".to_string());
57     TestDatabaseInterface::test.add_column(
58        "crawling",to_string(),
59        "crawling".to_string(),
60        "page_information".to_string(),
61        "url".to_string(),
62        "string".to_string());
63     TestDatabaseInterface::test.add_column(
64        "crawling",to_string(),
65        "crawling".to_string(),
66        "page_information".to_string(),
67        "htmls".to_string(),
68        "text".to_string());
69     TestDatabaseInterface::test.add_column(
70        "crawling",to_string(),
71        "crawling".to_string(),
72        "page_information".to_string(),
73        "title".to_string(),
74        "string".to_string());
75     TestDatabaseInterface::test.add_column(
76        "crawling",to_string(),
77        "crawling".to_string(),
78        "page_information".to_string(),
79        "description".to_string(),
80        "string".to_string());
81     TestDatabaseInterface::test.add_column(
82        "crawling",to_string(),
83        "crawling".to_string(),
84        "page_information".to_string(),
85        "keywords".to_string(),
86        "string".to_string());
87     TestDatabaseInterface::test.add_column(
88        "crawling",to_string(),
89        "crawling".to_string(),
90        "page_information".to_string(),
91        "content_text".to_string(),
92        "text".to_string());
93     TestDatabaseInterface::test.add_column(
94        "crawling",to_string(),
95        "crawling".to_string(),
96        "page_information".to_string(),
97        "log".to_string(),
98        "integer".to_string());
99     TestDatabaseInterface::test.add_column(
100        "crawling",to_string(),
101        "crawling".to_string(),
102        "page_information".to_string(),
103        "size_bytes".to_string(),
104        "integer".to_string());
105     TestDatabaseInterface::test.add_column(
106        "crawling",to_string(),
107        "crawling".to_string(),
108        "page_information".to_string(),
109        "model_crawl".to_string(),
110        "string".to_string());
111     TestDatabaseInterface::test.add_column(
112        "crawling",to_string(),
113        "crawling".to_string(),
114        "page_information".to_string(),
115        "parse_error".to_string(),
116        "error_type".to_string(),
117        "string".to_string());
118     TestDatabaseInterface::test.add_column(
119        "crawling",to_string(),
120        "page_information".to_string(),
121        "crawl_at".to_string(),
122        "string".to_string());
123 }

```

Gambar 1.5: Function `test_data_crawling` dalam file `main.rs`. Keseluruhan function dapat dilihat di *source code* langsung

```

1 use std::{
2     collections::HashMap,
3     fmt::format,
4     fs::File,
5     io::{Read, Write},
6     result,
7 };
8
9 use crate::database::utils::{self, DataType, InputDataEnum};
10
11 pub struct Cell {
12     pub data_type: DataType,
13     pub data_value: Vec<u8>,
14 }
15
16 impl Cell {
17     pub fn print(&self) {
18         match self.data_type {
19             DataType::String => {
20                 let result = utils::bytes_to_string(self.data_value.clone());
21                 println!("{}: {}", self.data_type, result);
22             }
23             DataType::Integer => {
24                 let result = utils::bytes_to_integer(self.data_value.clone());
25                 println!("{}: {}", self.data_type, result);
26             }
27             DataType::Null => {
28                 println!("{}: NULL", self.data_type);
29             }
30         }
31     }
32
33     pub fn value(&self) -> InputDataEnum {
34         match self.data_type {
35             DataType::String => InputDataEnum::String(utils::bytes_to_string(self.data_value.clone())),
36             DataType::Integer => InputDataEnum::Integer(utils::bytes_to_integer(self.data_value.clone())),
37             DataType::Null => InputDataEnum::Null,
38         }
39     }
40
41     pub fn change_value(&mut self, data: Vec<u8>) {
42         self.data_value = data;
43     }
44 }
45

```

Gambar 1.6: File cell.rs

```

1  use std::{
2      collections::HashMap,
3      fmt::format,
4      fs::File,
5      io::{Read, Write},
6      result,
7  };
8
9  use crate::database::{
10     cell::Cell,
11     utils::({self, bytes_to_string, DataType, InputDataEnum},
12   });
13
14 pub struct Column {
15     pub name: String,
16     pub data_type: DataType,
17     pub rows: Vec<Cell>,
18 }
19
20 impl Column {
21     // fn get_name(&self) -> String {
22     //     self.name.clone()
23     // }
24     // fn get_type(&self) -> DataType {
25     //     self.data_type
26     // }
27
28     pub fn get_name(&self) -> String {
29         self.name.clone()
30     }
31
32     pub fn get_data_type(&self) -> DataType {
33         match self.data_type {
34             DataType::String => DataType::String,
35             DataType::Integer => DataType::Integer,
36             DataType::Null => DataType::Null,
37         }
38     }
39
40     pub fn print_column(&self) {
41         for i in &self.rows {
42             i.print();
43         }
44     }
45
46     pub fn insert_data(&mut self, input: &InputDataEnum) {
47         let result = match input {
48             InputDataEnum::String(word) => utils::string_to_bytes(word.to_string()),
49             InputDataEnum::Integer(number) => utils::integer_to_bytes(*number).to_vec(),
50             InputDataEnum::Null => [0; (isize::BITS / 8) as usize].to_vec(),
51         };
52         let new_cell: Cell = Cell {
53             data_type: self.get_data_type(),
54             data_value: result,
55         };
56         self.rows.push(new_cell);
57     }
58
59     pub fn insert_default_data(&mut self) {
60         let result = match self.data_type {
61             DataType::Integer => utils::integer_to_bytes(0).to_vec(),
62             DataType::String => utils::string_to_bytes(".".to_owned()),
63             DataType::Null => {
64                 panic!("Not Supported Yet!")
65             }
66         };
67         let new_cell: Cell = Cell {
68             data_type: self.get_data_type(),
69             data_value: result,
70         };
71         self.rows.push(new_cell);
72     }
73
74     pub fn update_data(&mut self, index: usize, input: &InputDataEnum) {
75         let byte = match input {
76             InputDataEnum::String(word) => utils::string_to_bytes(word.to_string()),
77             InputDataEnum::Integer(num) => utils::integer_to_bytes(*num).to_vec(),
78             InputDataEnum::Null => vec![],
79         };
80         self.rows[index].change_value(byte);
81     }
82
83     pub fn search_for_index(&self, value: String) -> Vec<usize> {
84         let mut res: Vec<usize> = Vec::new();
85         let mut index = 0;
86         for row in &self.rows {
87             match row.value() {
88                 InputDataEnum::String(word) => {
89                     if value == word {
90                         if value.as_str().parse::<isize>().unwrap() == num {
91                             res.push(index);
92                         }
93                     }
94                     InputDataEnum::Integer(num) => {
95                         if value.as_str().parse::<isize>().unwrap() == num {
96                             res.push(index);
97                         }
98                     }
99                     InputDataEnum::Null => {}
100                }
101            index += 1;
102        }
103        res
104    }
105}

```

Gambar 1.7: File column.rs bagian 1

```

1  pub fn <deleter>(self, index: usize) {
2    self.rows.remove(index);
3  }
4
5
6  pub fn to_vec(self) -> Vec<Vec<u8>> {
7    let mut result: Vec<Vec<u8>> = Vec::new();
8    // Parsing columns row length
9    let column_data_length = util::string_to_bytes(self.row.len()) as isize;
10   result.append(&mut column_name_str[0..len]);
11
12   // Parsing Columns name
13   let mut column_name_len: u32 = util::string_to_bytes(self.name.len()) as isize;
14   result.append(&mut column_name_str[0..len]);
15   let mut column_name_bytes = util::string_to_bytes(self.name.clone());
16   column_name_bytes.append(&mut column_name_len);
17
18   match self.data_type {
19     Datatype::String => {
20       result.push(0);
21     }
22     Datatype::Integer => {
23       result.push(1);
24     }
25     Datatype::Null => {}
26   };
27
28   for column_data in Asset::rows {
29     match column_data.data_type {
30       Datatype::String => {
31         let mut str_len = 0;
32         while util::string_to_bytes(column_data.value[0..len]) as isize == 0 {
33           result.push(column_data.value[0..len]);
34           str_len += len;
35         }
36         Datatype::Integer => {}
37         Datatype::Null => {}
38       }
39     }
40     result.push(0);
41   }
42 }
43
44 pub fn to_data(buf: &mut Vec<u8>) -> Column {
45   // Parsing bytes to row length first
46   let rest_of_bytes = buf[0..vec];
47   let (col_name_len, rest_of_bytes) = rest_of_bytes.split_at((useisze::BITS / 8) as usize);
48   let col_name_len_usize: usize = from_le_bytes(col_name_len[0..8]);
49   let col_name = bytes_to_string(name_u8_to_vec());
50   *buf = vec;
51
52   // Parsing bytes to column name length first
53   let rest_of_bytes = buf[0..vec];
54   let (col_name_len, rest_of_bytes) = rest_of_bytes.split_at((useisze::BITS / 8) as usize);
55   let col_name_len_usize: usize = from_le_bytes(col_name_len[0..8]);
56   let col_name = bytes_to_string(name_u8_to_vec());
57   *buf = vec;
58
59   // Parsing bytes to column name value first
60   let rest_of_bytes = buf[0..vec];
61   let (col_name, rest_of_bytes) = rest_of_bytes.split_at((useisze::BITS / 8) as usize);
62   let col_name_bytes = bytes_to_string(name_u8_to_vec());
63   *buf = vec;
64
65   // Parsing Data Type
66   let (data_type, rest_of_bytes) = rest_of_bytes.split_at(1);
67   let data_type: Data_type = rest_of_bytes[0] as Data_type;
68   *buf = vec;
69
70   let mut column = match data_type.0 {
71     0 => Column {
72       data_type: Datatype::String,
73       name: col_name,
74       rows: Vec::new(),
75     },
76     1 => Column {
77       data_type: Datatype::Integer,
78       name: col_name,
79       rows: Vec::new(),
80     },
81     _ => panic!("Something went wrong on parsing bytes to Table")
82   };
83
84   }
85
86   for i in 0..col_name_len_usize {
87     match data_type.0 {
88       0 => {
89         let rest_of_bytes = buf[0..vec];
90         let (cell_len, rest_of_bytes) = rest_of_bytes.split_at((useisze::BITS / 8) as usize);
91         let (cell_val, rest_of_bytes) = rest_of_bytes.split_at((useisze::BITS / 8) as usize);
92         let cell_len_usize: usize = from_le_bytes(cell_len[0..8]);
93         let cell_val_bytes = bytes_to_string(data_val_u8_to_vec());
94         *buf = vec;
95
96         // Parsing bytes to table cell value first
97         let (cell_val, rest_of_bytes) = rest_of_bytes.split_at((useisze::BITS / 8) as usize);
98         let (cell_u8, rest_of_bytes) = rest_of_bytes.split_at(1);
99         let new_cell = Cell {
100           data_type: Data_type::String,
101           data_val: cell_val[0] as Data_type,
102           column: column,
103         };
104         column.rows.push(new_cell);
105         *buf = vec;
106       }
107       1 => {
108         // Parsing bytes to table cell value first
109         let (cell_val, rest_of_bytes) = rest_of_bytes.split_at((useisze::BITS / 8) as usize);
110         let (cell_u8, rest_of_bytes) = rest_of_bytes.split_at(1);
111         let new_cell = Cell {
112           data_type: Data_type::Integer,
113           data_val: cell_val[0] as Data_type,
114           column: column,
115         };
116         column.rows.push(new_cell);
117         *buf = vec;
118       }
119     } => panic!("Something went wrong on parsing bytes to table");
120   }
121 }
122
123 column
124 }
125 }
```

Gambar 1.8: File column.rs bagian 2

```

1  use std::collections::{HashMap, HashSet};
2
3  use crate::database::{
4      cell::Cell,
5      column::Column,
6      utils::{self, bytes_to_string, DataType, InputDataEnum},
7  };
8
9  pub struct Table {
10     pub name: String,
11     pub columns: Vec<Column>,
12     pub length: usize,
13 }
14
15 impl Table {
16     fn get_type(&self, name: String) -> DataType {
17         for column in &self.columns {
18             if column.name == name {
19                 return column.get_data_type();
20             }
21         }
22         return DataType::Null;
23     }
24
25     fn get_data_by_column(&self) -> HashMap<String, Vec<InputDataEnum>> {
26         let mut result: HashMap<String, Vec<InputDataEnum>> = HashMap::new();
27         for column in &self.columns {
28             let mut column_data: Vec<InputDataEnum> = Vec::new();
29             for cell in &column.rows {
30                 column_data.push(cell.value());
31             }
32             result.insert(column.get_name(), column_data);
33         }
34     }
35     result
36 }
37
38     pub fn get_column_names(&self) -> Vec<String> {
39         self.columns.iter().map(|col| col.get_name()).collect()
40     }
41
42     pub fn get_data(&self) -> Vec<HashMap<String, InputDataEnum>> {
43         let mut result: Vec<HashMap<String, InputDataEnum>> = Vec::new();
44         for index_row in 0..self.length {
45             let mut row_data: HashMap<String, InputDataEnum> = HashMap::new();
46             for column in &self.columns {
47                 let cell: &Cell = &column.rows[index_row];
48                 row_data.insert(column.get_name(), cell.value());
49             }
50             result.push(row_data);
51         }
52     }
53     result
54 }
55
56     fn get_data_by_index(&self, index: usize) -> HashMap<String, InputDataEnum> {
57         let mut row_data: HashMap<String, InputDataEnum> = HashMap::new();
58         for column in &self.columns {
59             let cell: &Cell = &column.rows[index];
60             row_data.insert(column.get_name(), cell.value());
61         }
62     }
63     row_data
64 }
65

```

Gambar 1.9: File table.rs bagian 1

```

1  pub fn search_by_column(
2      &self,
3      column_name: String,
4      search: String,
5  ) -> Vec<HashMap<String, InputDataEnum>> {
6      let mut result: Vec<HashMap<String, InputDataEnum>> = Vec::new();
7      let selected_column = self.search_column(column_name);
8      for index in 0..self.length {
9          match selected_column.rows[index].value() {
10              InputDataEnum::String(word) => {
11                  if search == word {
12                      result.push(self.get_data_by_index(index));
13                      true
14                  } else {
15                      false
16                  }
17              }
18              InputDataEnum::Integer(num) => {
19                  if search.as_str().parse::<i32>().unwrap() == num {
20                      result.push(self.get_data_by_index(index));
21                      true
22                  } else {
23                      false
24                  }
25              }
26              InputDataEnum::Null => false,
27          };
28      }
29      result
30  }
31
32
33
34  pub fn search_column(&self, name: String) -> &Column {
35      let mut selected_column: Option<&Column> = None;
36      for column in &self.columns {
37          if column.name == name {
38              selected_column = Some(column);
39          }
40      }
41
42      match selected_column {
43          Some(column) => column,
44          None => {
45              panic!("{:?}, Column not Found", name)
46          }
47      }
48  }
49
50
51  pub fn print(&self) {
52      println!("====Table {:?}====", self.name);
53      println!("Column: {:?}", self.get_column_names());
54      for index_data in 0..self.length {
55          println!("Data {:?};", index_data + 1);
56          for column in self.columns.iter() {
57              column.rows[index_data].print();
58          }
59          println!();
60      }
61      println!();
62  }
63
64  pub fn add_column(&mut self, column: Column) {
65      if self.check_column_index(column.name.clone()) != -1 {
66          panic!("Column Already Exists");
67      }
68      self.columns.push(column);
69  }
70
71  pub fn check_column_index(&self, name: String) -> isize {
72      let mut index = -1;
73
74      let mut loop_index = 0;
75      for column in &self.columns {
76          if column.name == name {
77              index = loop_index;
78              break;
79          }
80          loop_index += 1;
81      }
82      index
83  }
84
85

```

Gambar 1.10: File table.rs bagian 2

```

1  pub fn add_data_column(&mut self, name: String, input: InputDataEnum) {
2      for column in &mut self.columns {
3          if column.name == name {
4              column.insert_data(&input);
5              self.length += 1;
6              break;
7          }
8      }
9  }
10 }
11
12 pub fn add_data(&mut self, input_data: HashMap<String, String>) {
13     for item in self.columns.iter_mut() {
14         match input_data.get(&item.get_name()) {
15             Some(val) => {
16                 let value_enum = match item.get_data_type() {
17                     DataType::String => InputDataEnum::String(val.to_owned()),
18                     DataType::Integer => {
19                         InputDataEnum::Integer(val.as_str().parse::<isize>().unwrap())
20                     }
21                     DataType::Null => {
22                         panic!("Not Supported Yet!")
23                     }
24                 };
25                 item.insert_data(&value_enum);
26             }
27             None => {
28                 item.insert_default_data();
29             }
30         };
31     }
32     self.length += 1;
33 }
34
35 pub fn update(&mut self, index: usize, map: &HashMap<String, String>) {
36     for (key, value) in map.iter() {
37         for column in &mut self.columns {
38             if column.name == *key {
39                 let value_enum = match column.get_data_type() {
40                     DataType::String => InputDataEnum::String(value.to_owned()),
41                     DataType::Integer => {
42                         InputDataEnum::Integer(value.as_str().parse::<isize>().unwrap())
43                     }
44                     DataType::Null => {
45                         panic!("Not Supported Yet!")
46                     }
47                 };
48                 column.update_data(index, &value_enum);
49                 break;
50             }
51         }
52     }
53 }
54
55 pub fn delete_column(&mut self, column_name: String) {
56     let column_index = self.check_column_index(column_name);
57
58     if column_index == -1 {
59         panic!("Column not Found");
60     }
61
62     self.columns.remove(column_index as usize);
63 }
64
65 pub fn delete(&mut self, index: usize) {
66     for col in &mut self.columns {
67         col.delete(index);
68     }
69     self.length -= 1;
70 }
71
72

```

Gambar 1.11: File table.rs bagian 3

```

1  pub fn update_data(
2      &mut self,
3      where_data: HashMap<String, String>,
4      updated_data: HashMap<String, String>,
5  ) -> bool {
6      let mut updated_column_index_set: HashSet<usize> = HashSet::new();
7      for (column_key, column_val) in where_data.iter() {
8          let column = self.search.column(column_key.to_string());
9          let updated_column_index = column.search_for_index(column_val.to_string());
10         let mut current_updated_column_index_set: HashSet<usize> = HashSet::new();
11         updated_column_index.iter().for_each(|i| {
12             current_updated_column_index_set.insert(*i);
13         });
14         if updated_column_index_set.is_empty() {
15             updated_column_index_set = current_updated_column_index_set;
16         } else {
17             // https://users.rust-lang.org/t/intersection-of-hashes/32351/2
18             updated_column_index_set = updated_column_index_set
19                 .intersection(&current_updated_column_index_set)
20                 .copied()
21                 .collect();
22         }
23         for i in updated_column_index {
24             // pengurangan dilakukan agar index yang terpilih sesuai dengan data yang dituju
25             // karena setiap delete akan menggeser data yang ada
26             self.update(i, &updated_data);
27         }
28     }
29 }
30 true
31 }
32
33 pub fn delete_data(&mut self, where_data: HashMap<String, String>) -> bool {
34     let mut deleted_column_index_set: HashSet<usize> = HashSet::new();
35     for (column_key, column_val) in where_data.iter() {
36         let column = self.search.column(column_key.to_string());
37         let deleted_column_index = column.search_for_index(column_val.to_string());
38         let mut current_deleted_column_index_set: HashSet<usize> = HashSet::new();
39         deleted_column_index.iter().for_each(|i| {
40             current_deleted_column_index_set.insert(*i);
41         });
42         if deleted_column_index_set.is_empty() {
43             deleted_column_index_set = current_deleted_column_index_set;
44         } else {
45             // https://users.rust-lang.org/t/intersection-of-hashes/32351/2
46             deleted_column_index_set = deleted_column_index_set
47                 .intersection(&current_deleted_column_index_set)
48                 .copied()
49                 .collect();
50         }
51         let mut index = 0;
52         for i in deleted_column_index {
53             // pengurangan dilakukan agar index yang terpilih sesuai dengan data yang dituju
54             // karena setiap delete akan menggeser data yang ada
55             self.delete(i - index);
56             index += 1;
57         }
58     }
59     true
60 }
61
62 pub fn to_bytes(&self) -> Vec<u8> {
63     let mut bytes: Vec<u8> = Vec::new();
64     let mut name_str_len = utils::integer_to_bytes(self.name.len() as isize).to_vec();
65     bytes.append(&mut name_str_len);
66     let mut name_bytes = utils::string_to_bytes(self.name.clone());
67     bytes.append(&mut name_bytes);
68
69     let mut column_len = utils::integer_to_bytes(self.columns.len() as isize).to_vec();
70     bytes.append(&mut column_len);
71
72     let mut table_len = utils::integer_to_bytes(self.length as isize).to_vec();
73     bytes.append(&mut table_len);
74
75     for column in self.columns.iter() {
76         let mut col_bytes = column.to_bytes();
77         bytes.append(&mut col_bytes);
78     }
79
80 }
81

```

Gambar 1.12: File table.rs bagian 4

```

1  pub fn to_data(buf_u8: &mut Vec<u8>) -> Table {
2      // Parsing bytes to table name length first
3      let rest_of_bytes = buf_u8.to_vec();
4      let (name_len, rest_of_bytes) = rest_of_bytes.split_at((usize::BITS / 8) as usize);
5      let name_len_u8: [u8; 8] = name_len.try_into().unwrap();
6      let name_len_usize = usize::from_le_bytes(name_len_u8);
7      *buf_u8 = rest_of_bytes.to_vec();
8
9
10     // Parsing bytes to the value of table name
11     let rest_of_bytes = buf_u8.to_vec();
12     let (name_u8, rest_of_bytes) = rest_of_bytes.split_at(name_len_usize);
13     let name = bytes_to_string(name_u8.to_vec());
14     *buf_u8 = rest_of_bytes.to_vec();
15
16     // Parsing bytes to table length
17     let rest_of_bytes = buf_u8.to_vec();
18     let (table_column_len, rest_of_bytes) = rest_of_bytes.split_at((usize::BITS / 8) as usize);
19     let table_column_len_u8: [u8; 8] = table_column_len.try_into().unwrap();
20     let table_column_len_usize = usize::from_le_bytes(table_column_len_u8);
21
22     *buf_u8 = rest_of_bytes.to_vec();
23
24     // Parsing bytes to table length
25     let rest_of_bytes = buf_u8.to_vec();
26     let (table_len, rest_of_bytes) = rest_of_bytes.split_at((usize::BITS / 8) as usize);
27     let table_len_u8: [u8; 8] = table_len.try_into().unwrap();
28     let table_len_usize = usize::from_le_bytes(table_len_u8);
29     *buf_u8 = rest_of_bytes.to_vec();
30
31     // Start Parsing the column
32     let mut columns: Vec<Column> = Vec::new();
33
34     for i in 0..table_column_len_usize {
35         let new_col = Column::to_data(buf_u8);
36         columns.push(new_col);
37     }
38
39     Table {
40         name: name,
41         length: table_len_usize,
42         columns: columns,
43     }
44     // println!("{}:{?}{:?}", i, arr_values);
45
46     pub fn name_to_bytes(&self) -> Vec<u8> {
47         let mut vec: Vec<u8> = Vec::new();
48         for n in self.name.chars() {
49             vec.push(n as u8);
50         }
51         return vec;
52     }
53 }
54

```

Gambar 1.13: File table.rs bagian 5

```

1  use std::{
2      collections::HashMap,
3      fs::File,
4      io::{Read, Write},
5  };
6
7  use crate::database::{
8      cell::Cell,
9      column::Column,
10     table:: {Self, Table},
11     utils:: {Self, bytes_to_string, DataType, InputDataEnum},
12 };
13
14 pub struct Schema {
15     pub name: String,
16     pub tables: Vec<Table>,
17     pub index: HashMap<String, Vec<HashMap<String, InputDataEnum>>,
18 }
19
20 impl Schema {
21     pub fn to_bytes(&self) -> Vec<u8> {
22         let mut bytes: Vec<u8> = Vec::new();
23         let mut name_str_len = utils::integer_to_bytes(self.name.len() as isize).to_vec();
24         bytes.append(&mut name_str_len);
25         let mut name_bytes = utils::string_to_bytes(self.name.clone());
26         bytes.append(&mut name_bytes);
27
28         let mut table_len = utils::integer_to_bytes(self.tables.len() as isize).to_vec();
29         bytes.append(&mut table_len);
30
31         for table in self.tables.iter() {
32             bytes.append(&mut table.to_bytes());
33         }
34         bytes
35     }
36
37     pub fn to_data(buf_u8: &mut Vec<u8>) -> Schema {
38         // Parsing bytes to table name length first
39         let rest_of_bytes = buf_u8.to_vec();
40         let (name_len, rest_of_bytes) = rest_of_bytes.split_at((usize::BITS / 8) as usize);
41         let name_len_u8: [u8; 8] = name_len.try_into().unwrap();
42         let name_len_usize = usize::from_le_bytes(name_len_u8);
43         *buf_u8 = rest_of_bytes.to_vec();
44
45         // Parsing bytes to the value of table name
46         let rest_of_bytes = buf_u8.to_vec();
47         let (name_u8, rest_of_bytes) = rest_of_bytes.split_at(name_len_usize);
48         let name = bytes_to_string(name_u8.to_vec());
49         *buf_u8 = rest_of_bytes.to_vec();
50
51         // Parsing bytes to table length
52         let rest_of_bytes = buf_u8.to_vec();
53         let (table_len, rest_of_bytes) = rest_of_bytes.split_at((usize::BITS / 8) as usize);
54         let table_len_u8: [u8; 8] = table_len.try_into().unwrap();
55         let table_len_usize = usize::from_le_bytes(table_len_u8);
56         *buf_u8 = rest_of_bytes.to_vec();
57
58         // Start Parsing the column
59         let mut tables: Vec<Table> = Vec::new();
60
61         for i in 0..table_len_usize {
62             let new_col = Table::to_data(buf_u8);
63             tables.push(new_col);
64         }
65
66         let mut schema = Schema {
67             name,
68             tables,
69             index: HashMap::new(),
70         };
71         schema.build_index();
72
73         schema
74     }
75 }
76
77

```

Gambar 1.14: File schema.rs bagian 1

```

1  pub fn save(&mut self) -> std::io::Result<()> {
2      let path: String = "schema".to_owned() + &self.name;
3      let mut file = File::create(path)?;
4      let buf_result = self.to_bytes();
5      file.write_all(&buf_result)?;
6      self.clear_index();
7      Ok(())
8  }
9
10 pub fn create_table(&mut self, table: Table) {
11     if self.check_table_index(table.name.clone()) != -1 {
12         panic!("Table Already Exists");
13     }
14     self.tables.push(table);
15     let _ = self.save();
16 }
17
18 pub fn add_column_to_table(&mut self, table_name: String, mut column: Column) -> bool {
19     let table = self.search_table(table_name);
20     match column.data_type {
21         DataType::String => {
22             column.rows = (1..=table.length)
23                 .map(|_| Cell {
24                     data_type: DataType::String,
25                     data_value: utils::string_to_bytes("".to_owned()),
26                 })
27                 .collect();
28         }
29         DataType::Integer => {
30             column.rows = (1..=table.length)
31                 .map(|_| Cell {
32                     data_type: DataType::String,
33                     data_value: utils::integer_to_bytes(0).to_vec(),
34                 })
35                 .collect();
36         }
37         DataType::Null => {}
38     }
39     table.add_column(column);
40     let _ = self.save();
41     true
42 }
43
44 pub fn list_column_on_table(&mut self, name: String) -> Vec<String> {
45     let table = self.search_table(name);
46     table.get_column_names()
47 }
48
49 pub fn delete_column_on_table(&mut self, table_name: String, column_name: String) -> bool {
50     let table = self.search_table(table_name);
51
52     table.delete_column(column_name);
53     let _ = self.save();
54
55     true
56 }
57
58 pub fn search_table(&mut self, name: String) -> &mut Table {
59     let mut selected_table: Option<&mut Table> = None;
60     for table in &mut self.tables {
61         if table.name == name {
62             selected_table = Some(table);
63         }
64     }
65
66     match selected_table {
67         Some(table) => table,
68         None => {
69             panic!("Table not Found");
70         }
71     }
72 }
73
74 pub fn check_table_index(&self, name: String) -> isize {
75     let mut index = -1;
76
77     let mut loop_index = 0;
78     for table in &self.tables {
79         if table.name == name {
80             index = loop_index;
81             break;
82         }
83         loop_index += 1;
84     }
85     index
86 }
87
88
89

```

Gambar 1.15: File schema.rs bagian 2

```

1  pub fn drop_table(&mut self, table_name: String) {
2      let table_index = self.check_table_index(table_name);
3
4      if table_index == -1 {
5          panic!("Table not Found");
6      }
7
8      self.tables.remove(table_index as usize);
9      let _ = self.save();
10 }
11
12 pub fn add_data(&mut self, table_name: String, data: HashMap<String, String>) -> bool {
13     let table = self.search_table(table_name);
14     table.add_data(data);
15     let _ = self.save();
16     true
17 }
18
19 pub fn get_data(&mut self, table_name: String) -> Vec<HashMap<String, InputDataEnum>> {
20     let table = self.search_table(table_name);
21     let result = table.get_data();
22     result
23 }
24
25 pub fn search_data(
26     &mut self,
27     table_name: String,
28     column_name: String,
29     value: String,
30 ) -> Vec<HashMap<String, InputDataEnum>> {
31     let table = self.search_table(table_name);
32     let result = table.search_by_column(column_name, value);
33     result
34 }
35
36 pub fn update_data(
37     &mut self,
38     table_name: String,
39     where_data: HashMap<String, String>,
40     updated_data: HashMap<String, String>,
41 ) -> bool {
42     let table = self.search_table(table_name);
43     let result = table.update_data(where_data, updated_data);
44     let _ = self.save();
45     result
46 }
47
48 pub fn delete_data(&mut self, table_name: String, where_data: HashMap<String, String>) -> bool {
49     let table = self.search_table(table_name);
50     let result = table.delete_data(where_data);
51     let _ = self.save();
52     result
53 }
54
55 pub fn join_table(
56     &mut self,
57     table_name: String,
58     column_name: String,
59     table_join: String,
60     column_join: String,
61     join_type: String,
62 ) -> Vec<HashMap<String, InputDataEnum>> {
63     let result: Vec<HashMap<String, InputDataEnum>> = match join_type.as_str() {
64         "inner" => self.join_inner_table(table_name, column_name, table_join, column_join),
65         "left" => self.left_right_join_table(table_name, column_name, table_join, column_join),
66         "right" => self.left_right_join_table(table_join, column_join, table_name, column_name),
67         _ => {
68             panic!("Invalid Join Type")
69         }
70     };
71     result
72 }
73
74
75

```

Gambar 1.16: File schema.rs bagian 3

```

1 pub fn left_right_join_table(
2     self: &Self,
3     table_name: String,
4     column_name: String,
5     table_join: String,
6     join: String,
7 ) -> Vec<HashMap<String, InputDataEnum>> {
8     let selected_table = self.search_table(table_name.to_string());
9     let mut res: Vec<HashMap<String, InputDataEnum>> = selected_table.get_data();
10    let selected_join_table = self.search_table(table_join.to_string());
11
12    let mut result: Vec<HashMap<String, InputDataEnum>> = Vec::new();
13
14    for item in &mut left_join {
15        let val = match item.get(&column_name) {
16            Some(input) => input,
17            None => panic!("Key tidak ditemukan"),
18        };
19
20        let join_result = match val {
21            InputDataEnum::String(word) => {
22                selected_join_table.search_by_column(column_join.clone(), word.clone())
23            }
24            InputDataEnum::Integer(num) => {
25                selected_join_table.search_by_column(column_join.clone(), num.to_string())
26            }
27            InputDataEnum::Null => Vec::new(),
28        };
29
30        let mut res: HashMap<String, InputDataEnum> = HashMap::new();
31        if join_result.len() > 0 {
32            match join_result.first() {
33                Some(item) => {
34                    for k in item {
35                        let value = match v {
36                            InputDataEnum::Integer(num) => InputDataEnum::Integer(*num),
37                            InputDataEnum::String(word) => InputDataEnum::String(word.to_string()),
38                            InputDataEnum::Null => InputDataEnum::Null,
39                        };
40                        res.insert(k.to_string(), value);
41                    }
42                    res.insert(k.to_string(), value);
43                }
44                None => {}
45            }
46        }
47    }
48    for (k, v) in item {
49        let value = match v {
50            InputDataEnum::Integer(num) => InputDataEnum::Integer(*num),
51            InputDataEnum::String(word) => InputDataEnum::String(word.to_string()),
52            InputDataEnum::Null => InputDataEnum::Null,
53        };
54        res.insert(k.to_string(), value);
55    }
56    res.push(res);
57 }
58
59 }
60
61 result
62
63 pub fn join_inner_table(
64     &mut self,
65     table_name: String,
66     column_name: String,
67     table_join: String,
68     column_join: String,
69 ) -> Vec<HashMap<String, InputDataEnum>> {
70     let selected_table = self.search_table(table_name.to_string());
71     let left_join: Vec<HashMap<String, InputDataEnum>> = selected_table.get_data();
72     let selected_join_table = self.search_table(table_join.to_string());
73
74     let mut result: Vec<HashMap<String, InputDataEnum>> = Vec::new();
75
76     for item in &left_join {
77         let val = match item.get(&column_name) {
78             Some(input) => input,
79             None => panic!("Key tidak ditemukan"),
80         };
81
82         let join_result = match val {
83             InputDataEnum::String(word) => {
84                 selected_join_table.search_by_column(column_join.clone(), word.clone())
85             }
86             InputDataEnum::Integer(num) => {
87                 selected_join_table.search_by_column(column_join.clone(), num.to_string())
88             }
89             InputDataEnum::Null => Vec::new(),
90         };
91
92         for res_search in join_result {
93             let mut res: HashMap<String, InputDataEnum> = HashMap::new();
94             for (k, v) in res_search {
95                 res.insert(k, v);
96             }
97             for (k, v) in item {
98                 let value = match v {
99                     InputDataEnum::Integer(num) => InputDataEnum::Integer(*num),
100                     InputDataEnum::String(word) => InputDataEnum::String(word.to_string()),
101                     InputDataEnum::Null => InputDataEnum::Null,
102                 };
103                 res.insert(k.to_string(), value);
104             }
105             res.push(res);
106         }
107     }
108 }
109
110 result
111

```

Gambar 1.17: File schema.rs bagian 4

```

1  pub fn build_index(&mut self) {
2      let index_file_name: String = "index/.to_owned() + &self.name;
3      let mut file = Schema::get_file(index_file_name);
4      let mut buf = Vec::new();
5      file.read_to_end(&mut buf);
6
7      // If file empty then assign an empty map
8      let decoded_index: HashMap<String, Vec<HashMap<String, InputDataEnum>>;
9      if buf.len() == 0 {
10         decoded_index = HashMap::new();
11     } else {
12         decoded_index = bincode::deserialize(&buf..).unwrap();
13     }
14     self.index = decoded_index;
15 }
16
17 pub fn save_index(
18     &mut self,
19     key: String,
20     result: Vec<HashMap<String, InputDataEnum>>;
21 ) -> std::io::Result<()> {
22     let index_file_name: String = "index/.to_owned() + &self.name;
23     let mut file = File::create(index_file_name)?;
24     // If file empty then assign an empty map
25     self.index.insert(key.clone(), result);
26     let index_u8: Vec = bincode::serialize(&self.index).unwrap();
27     file.write_all(&index_u8)?;
28     Ok(())
29 }
30
31 pub fn clear_index(&mut self) -> std::io::Result<()> {
32     let index_file_name: String = "index/.to_owned() + &self.name;
33     let mut file = File::create(index_file_name)?;
34     // If file empty then assign an empty map
35     let decoded_index: HashMap<String, Vec<HashMap<String, InputDataEnum>> = HashMap::new();
36     let index_u8: Vec = bincode::serialize(&decoded_index).unwrap();
37     file.write_all(&index_u8)?;
38     self.index = decoded_index;
39     Ok(())
40 }
41
42 pub fn print(&self) {
43     println!("Database {:?}", self.name);
44     for table in self.tables.iter() {
45         table.print();
46     }
47     println!();
48     println!();
49 }
50
51 pub fn list_all_table(&self) -> Vec<String> {
52     let mut table_names: Vec<String> = Vec::new();
53     for table in self.tables.iter() {
54         table_names.push(table.name.clone());
55     }
56     table_names
57 }
58
59 fn get_file(name: String) -> File {
60     let read_file_res = File::open(name.as_str());
61     let file = match read_file_res {
62         Ok(read_file) => read_file,
63         Err(err) => {
64             let created_file_res = File::create(name.as_str());
65             match created_file_res {
66                 Ok(created_file) => created_file,
67                 Err(err) => panic!("Failed Open file: {:?}", err),
68             };
69         };
70         // After Create File we need to open it again so its not failing
71         let read_file_res = File::open(name.as_str());
72         let file = match read_file_res {
73             Ok(read_file) => read_file,
74             Err(err) => {
75                 panic!("Failed Open file: {:?}", err)
76             };
77         };
78         file
79     };
80     file
81 };
82 file
83 }
84 }
```

Gambar 1.18: *File schema.rs bagian 5*

```

1  use std::collections::HashMap;
2
3  use zbus::interface;
4
5  use crate::database::database_interface::DatabaseInterface;
6
7  pub struct DatabaseConnection {
8      pub db_interface: DatabaseInterface,
9  }
10 #[interface(name = "org.two.DatabaseConnection")]
11 impl DatabaseConnection {
12     async fn select_database(&mut self, database_name: &str) -> String {
13         let res = self
14             .db_interface
15             .select_database(&database_name.to_string());
16         match res {
17             true => "Connected to database".to_string(),
18             false => "Database not found".to_string(),
19         }
20     }
21
22     async fn create_database(&mut self, database_name: &str) -> String {
23         let res = self
24             .db_interface
25             .create_database(&database_name.to_string());
26         match res {
27             true => "Database Created!".to_string(),
28             false => "Database Failed to create".to_string(),
29         }
30     }
31
32     async fn drop_database(&mut self, database_name: &str) -> String {
33         let res = self
34             .db_interface
35             .drop_database(&database_name.to_string());
36         match res {
37             true => "Database Dropped!".to_string(),
38             false => "Failed to Drop Database".to_string(),
39         }
40     }
41
42     async fn list_table(&mut self) -> Vec<String> {
43         let res = self.db_interface.list_all_table();
44         res
45     }
46
47     async fn create_table(&mut self, table_name: &str) -> String {
48         let column = vec![];
49         let res = self
50             .db_interface
51             .create_table(&table_name.to_string(), column);
52         match res {
53             true => "Table Created".to_string(),
54             false => "Failed to create table".to_string(),
55         }
56     }
57
58     async fn drop_table(&mut self, table_name: &str) -> String {
59         let res = self.db_interface.drop_table(&table_name.to_string());
60         match res {
61             true => "Table Dropped".to_string(),
62             false => "Failed to drop table".to_string(),
63         }
64     }
65
66     async fn add_column(&mut self, table_name: &str, name: &str, data_type: &str) -> String {
67         let res = self.db_interface.add_column_to_table(
68             &table_name.to_string(),
69             name.to_string(),
70             data_type.to_string(),
71         );
72         match res {
73             true => "Column Created".to_string(),
74             false => "Failed to create column".to_string(),
75         }
76     }
77
78     async fn list_column(&mut self, table_name: &str) -> Vec<String> {
79         let res = self
80             .db_interface
81             .list_column_on_table(table_name.to_string());
82         res
83     }
84
85     async fn delete_column(&mut self, table_name: &str, column_name: &str) -> String {
86         let res = self
87             .db_interface
88             .delete_column_on_table(table_name.to_string(), column_name.to_string());
89         match res {
90             true => "Column Deleted".to_string(),
91             false => "Failed to delete column".to_string(),
92         }
93     }
94
95     async fn add_data(&mut self, table_name: &str, data: HashMap<String, String>) -> String {
96         let res = self.db_interface.add_data(&table_name.to_string(), data);
97         match res {
98             true => "Data Created".to_string(),
99             false => "Failed to create data".to_string(),
100        }
101    }
102
103
104

```

Gambar 1.19: File database_connection.rs bagian 1

```

1  async fn update_data(
2      &mut self,
3      table_name: &str,
4      where_data: HashMap<String, String>,
5      updated_data: HashMap<String, String>,
6  ) -> String {
7      let res = self
8          .db_interface
9          .update_data(&table_name.to_owned(), where_data, updated_data);
10     match res {
11         true => "Data Updated".to_string(),
12         false => "Failed to update data".to_string(),
13     }
14 }
15
16 async fn delete_data(
17     &mut self,
18     table_name: &str,
19     where_data: HashMap<String, String>,
20 ) -> String {
21     let res = self
22         .db_interface
23         .delete_data(&table_name.to_owned(), where_data);
24     match res {
25         true => "Data Deleted".to_string(),
26         false => "Failed to delete data".to_string(),
27     }
28 }
29
30 // TODO
31 // async fn get_data(&mut self, table_name: &str) -> Vec<HashMap<String, String>> {
32 //     let res = self.db_interface.get_data(&table_name.to_string());
33 //     res
34 // }
35 // // // TODO
36 // async fn search_data(
37 //     &mut self,
38 //     table_name: &str,
39 //     column_name: String,
40 //     value: String,
41 // ) -> Vec<HashMap<String, String>> {
42 //     let res = self.db_interface.get_data(&table_name.to_string());
43 //     res
44 // }
45 // // // TODO
46 // async fn join_table(
47 //     &mut self,
48 //     table_name: &str,
49 //     column_name: &str,
50 //     table_join: &str,
51 //     column_join: &str,
52 //     join_type: &str,
53 // ) -> Vec<HashMap<String, String>> {
54 //     let res = self.db_interface.join_table(
55 //         table_name.to_string(),
56 //         column_name.to_string(),
57 //         table_join.to_string(),
58 //         column_join.to_string(),
59 //         join_type.to_string(),
60 //     );
61 //     res
62 // }
63
64 // #[zbus(property)]
65 // async fn greeter_name(&self) -> &str{
66 //     // &self.name
67 //     "str"
68 // }
69 // #[zbus(property)]
70 // async fn set_greeter_name(&mut self, name: &str){
71 //     // self.name = name;
72 // }
73 }
74

```

Gambar 1.20: File database_connection.rs bagian 2

```

1  use std::{
2      collections::HashMap,
3      fs::{self, File},
4      io::{Error, ErrorKind, Read},
5  };
6
7  use crate::database::{
8      column::Column,
9      schema::Schema,
10     table::Table,
11     utils::{parse_new_column, DataType, InputDataEnum},
12 };
13
14 pub struct DatabaseInterface {
15     pub is_connect: bool,
16     pub database: Option<Schema>,
17 }
18
19 impl DatabaseInterface {
20     pub fn show_databases(&self) -> Vec<String> {
21         let files = fs::read_dir("./schema");
22
23         let result: Vec<String> = match files {
24             Ok(dir) => dir
25                 .map(|dir_entry| match dir_entry {
26                     Ok(file) => {
27                         let res: String = match file.file_name().to_str() {
28                             Some(name) => name.to_string(),
29                             None => {
30                                 panic!("Something went Wrong")
31                             }
32                         );
33                         res
34                     }
35                     Err(_) => {
36                         panic!("Something went Wrong")
37                     }
38                 })
39                 .collect(),
40             Err(_) => {
41                 panic!("Something went Wrong!")
42             }
43         };
44         result
45     }
46     pub fn select_database(&mut self, database_name: &String) -> bool {
47         let path = "schema/".to_owned() + database_name;
48         let open_file = File::open(path);
49
50         match open_file {
51             Ok(mut file) => {
52                 let mut buf = Vec::new();
53                 let read_res = file.read_to_end(&mut buf);
54                 match read_res {
55                     Ok(_) => (),
56                     Err(err) => {
57                         panic!("Error: {:?}", err)
58                     }
59                 };
59                 let parsed_schema = Schema::to_data(&mut buf);
60                 self.database = Some(parsed_schema);
61                 self.is_connect = true;
62                 self.build_index();
63                 true
64             }
65             Err(err) => match err.kind() {
66                 ErrorKind::NotFound => false,
67                 _ => panic!("Something went wrong"),
68             },
69         },
70     }
71 }
72
73 fn build_index(&mut self) {
74     match &mut self.database {
75         Some(schema) => {
76             let build_index = schema.build_index();
77         }
78         None => {}
79     }
80 }
81

```

Gambar 1.21: File database_interface.rs bagian 1

```

1 pub fn create_database(&self, database_name: &String) -> bool {
2     let mut new_schema = Schema {
3         name: database_name.clone(),
4         tables: Vec::new(),
5         index: HashMap::new(),
6     };
7     let _ = new_schema.save();
8     true
9 }
10
11 pub fn drop_database(&mut self, database_name: &String) -> bool {
12     let path = "schemas/".to_owned() + database_name;
13     let result = fs::remove_file(path);
14     match result {
15         Ok(_) => {
16             match &mut self.database {
17                 Some(database) => {
18                     if database.name == *database_name {
19                         self.database = None;
20                         self.is_connect = false;
21                     }
22                 }
23                 None => {}
24             };
25             let path_index = "index/".to_owned() + database_name;
26             let result = fs::remove_file(path_index);
27             match result {
28                 Ok(_) => true,
29                 Err(_) => false,
30             }
31         }
32         Err(_) => false,
33     }
34 }
35
36 pub fn list_all_table(&self) -> Vec<String> {
37     let check_database = match &self.database {
38         Some(database) => database,
39         None => {
40             panic!("Please select database first!")
41         }
42     };
43     check_database.list_all_table()
44 }
45
46
47 pub fn create_table(
48     &mut self,
49     table_name: &String,
50     columns: Vec<HashMap<String, String>>,
51 ) -> bool {
52     let mut table = Table {
53         name: table_name.clone(),
54         columns: Vec::new(),
55         length: 0,
56     };
57     for column in columns {
58         let col = parse_new_column(column);
59         table.add_column(col);
60     }
61     match &mut self.database {
62         Some(database) => {
63             database.create_table(table);
64             true
65         }
66         None => {
67             panic!("Please Connect to database")
68         }
69     }
70 }
71
72 pub fn drop_table(&mut self, table_name: &String) -> bool {
73     let check_database = match &mut self.database {
74         Some(database) => database,
75         None => {
76             panic!("Please select database first!")
77         }
78     };
79     check_database.drop_table(table_name.to_owned());
80     true
81 }
82
83 }
84

```

Gambar 1.22: File database_interface.rs bagian 2

```

1  pub fn add_column_to_table(
2      &mut self,
3      table_name: &String,
4      name: String,
5      data_type: String,
6  ) -> bool {
7      let check_database = match &mut self.database {
8          Some(database) => database,
9          None => {
10              panic!("Please select database first!")
11          }
12      };
13  }
14
15  let mut map_column: HashMap<String, String> = HashMap::new();
16  map_column.insert("name".to_string(), name);
17  map_column.insert("type".to_string(), data_type);
18
19  let column = parse_new_column(map_column);
20
21  check_database.add_column_to_table(table_name.to_owned(), column);
22  true
23 }
24
25 pub fn list_column_on_table(&mut self, table_name: String) -> Vec<String> {
26     let check_database = match &mut self.database {
27         Some(database) => database,
28         None => {
29             panic!("Please select database first!")
30         }
31     };
32
33     let column_names = check_database.list_column_on_table(table_name);
34     column_names
35 }
36
37 pub fn delete_column_on_table(&mut self, table_name: String, column_name: String) -> bool {
38     let check_database = match &mut self.database {
39         Some(database) => database,
40         None => {
41             panic!("Please select database first!")
42         }
43     };
44
45     check_database.delete_column_on_table(table_name, column_name);
46     true
47 }
48
49 pub fn add_data(&mut self, table_name: &String, data: HashMap<String, String>) -> bool {
50     let check_database = match &mut self.database {
51         Some(database) => database,
52         None => {
53             panic!("Please select database first!")
54         }
55     };
56
57     check_database.add_data(table_name.to_owned(), data);
58
59     true
60 }
61
62 pub fn get_data(&mut self, table_name: &String) -> Vec<HashMap<String, InputDataEnum>> {
63     let check_database = match &mut self.database {
64         Some(database) => database,
65         None => {
66             panic!("Please select database first!")
67         }
68     };
69     let result = check_database.get_data(table_name.to_owned());
70     println!("{:?}", result);
71     result
72 }
73
74 pub fn search_data(
75     &mut self,
76     table_name: &String,
77     column_name: String,
78     value: String,
79 ) -> Vec<HashMap<String, InputDataEnum>> {
80     let check_database = match &mut self.database {
81         Some(database) => database,
82         None => {
83             panic!("Please select database first!")
84         }
85     };
86     let result = check_database.search_data(table_name.to_owned(), column_name, value);
87     println!("{:?}", result);
88     result
89 }
90

```

Gambar 1.23: File database_interface.rs bagian 3

```

1  pub fn update_data(
2    &mut self,
3    table_name: &String,
4    where_data: HashMap<String, String>,
5    updated_data: HashMap<String, String>,
6  ) -> bool {
7    let check_database = match &mut self.database {
8      Some(database) => database,
9      None => {
10        panic!("Please select database first!")
11      }
12    };
13
14    let result = check_database.update_data(table_name.to_owned(), where_data, updated_data);
15
16    println!("({:?}", result);
17
18    true
19  }
20
21  pub fn delete_data(
22    &mut self,
23    table_name: &String,
24    where_data: HashMap<String, String>,
25  ) -> bool {
26    let check_database = match &mut self.database {
27      Some(database) => database,
28      None => {
29        panic!("Please select database first!")
30      }
31    };
32    let result = check_database.delete_data(table_name.to_owned(), where_data);
33
34    println!("({:?}", result);
35
36    true
37  }
38
39  pub fn join_table(
40    &mut self,
41    table_name: String,
42    column_name: String,
43    table_join: String,
44    column_join: String,
45    join_type: String,
46  ) {
47    let check_database = match &mut self.database {
48      Some(database) => database,
49      None => {
50        panic!("Please select database first!")
51      }
52    };
53
54    let key = format!(
55      "{}_{}_{}_{}",
56      table_name, column_name, table_join, column_join, join_type
57    );
58
59    match check_database.index.get(&key) {
60      Some(_) => {
61        // panic!("Index Saved")
62      }
63      None => {
64        let result = check_database.join_table(
65          table_name,
66          column_name,
67          table_join,
68          column_join,
69          join_type,
70        );
71        let _ = check_database.save_index(key.clone(), result);
72      }
73    };
74
75    let result = match check_database.index.get(&key) {
76      Some(val) => val,
77      None => &vec![],
78    };
79
80    println!("({:?}", result);
81  }
82
83  pub fn print(&mut self) {
84    let check_database = match &mut self.database {
85      Some(database) => database,
86      None => {
87        panic!("Please select database first!")
88      }
89    };
90    check_database.print();
91  }
92}
93
94

```

Gambar 1.24: File database_interface.rs bagian 4

```
1 pub mod cell;
2 pub mod column;
3 pub mod database_interface;
4 pub mod schema;
5 pub mod table;
6 pub mod test_interface;
7 pub mod utils;
8 pub mod database_connection;
9
```

Gambar 1.25: *File mod*

```

1 use std::collections::HashMap;
2 use crate::database::{ 
3     column::Column,
4     database_interface::DatabaseInterface,
5     schema::Schema,
6     table::Table,
7     utils::{parse_new_column, DataType, InputDataEnum},
8 };
9
10 pub struct TestDatabaseInterface {}
11
12 impl TestDatabaseInterface {
13     pub fn test_show_database() -> Vec<String> {
14         let db_interface = DatabaseInterface {
15             is_connect: false,
16             database: None,
17         };
18         let res = db_interface.show_databases();
19         println!("({})", res);
20         res
21     }
22 }
23 pub fn print(database_name: String) {
24     let mut db_interface = DatabaseInterface {
25         is_connect: false,
26         database: None,
27     };
28     let res = db_interface.select_database(&database_name);
29     match res {
30         true => {
31             println!("Database berhasil terhubung")
32         }
33         false => println!("Database tidak ditemukan"),
34     };
35     let res_columns = db_interface.print();
36     println!("({})", res_columns);
37 }
38
39 pub fn test_select_database(database_name: String) {
40     let mut db_interface = DatabaseInterface {
41         is_connect: false,
42         database: None,
43     };
44     let res = db_interface.select_database(&database_name);
45     match res {
46         true => {
47             println!("Database berhasil terhubung")
48         }
49         false => println!("Database tidak ditemukan"),
50     };
51 }
52
53 pub fn test_create_database(database_name: String) {
54     let mut db_interface = DatabaseInterface {
55         is_connect: false,
56         database: None,
57     };
58     let res = db_interface.create_database(&database_name);
59     match res {
60         true => {
61             println!("Database berhasil terbuat")
62         }
63         false => println!("Terjadi suatu error"),
64     };
65 }
66
67 pub fn test_drop_database(database_name: String) {
68     let mut db_interface = DatabaseInterface {
69         is_connect: false,
70         database: None,
71     };
72     let res = db_interface.drop_database(&database_name);
73     match res {
74         true => {
75             println!("Database berhasil dihapus")
76         }
77         false => println!("Database tidak ditemukan"),
78     };
79 }
80

```

Gambar 1.26: File test_interface.rs bagian 1

```

1  pub fn test_list_table(database_name: String) {
2    let mut db_interface = DatabaseInterface {
3      is_connect: false,
4      database: None,
5    };
6    let res = db_interface.select_database(&database_name);
7    match res {
8      true => {
9        println!("Database berhasil terhubung")
10       } false => println!("Database tidak ditemukan"),
11    };
12  };
13  let res_tables = db_interface.list_all_table();
14  println!("({})", res_tables);
15  }
16  }
17  }
18  }
19  pub fn test_create_table(database_name: String, table_name: String) {
20    let mut db_interface = DatabaseInterface {
21      is_connect: false,
22      database: None,
23    };
24    let res = db_interface.select_database(&database_name);
25    match res {
26      true => {
27        println!("Database berhasil terhubung")
28       } false => println!("Database tidak ditemukan"),
29    };
30    let columns = vec![];
31    let res_tables = db_interface.create_table(&table_name, columns);
32    match res_tables {
33      true => {
34        println!("Berhasil membuat tabel")
35       } _ => {
36         panic!("Something went wrong")
37       }
38    };
39  };
40  }
41  }
42  }
43  pub fn test_create_table_with_column(database_name: String, table_name: String) {
44    let mut db_interface = DatabaseInterface {
45      is_connect: false,
46      database: None,
47    };
48    let res = db_interface.select_database(&database_name);
49    match res {
50      true => {
51        println!("Database berhasil terhubung")
52       } false => println!("Database tidak ditemukan"),
53    };
54    let mut column1: HashMap<String, String> = HashMap::new();
55    column1.insert("name".to_string(), "id".to_string());
56    column1.insert("type".to_string(), "integer".to_string());
57    let mut column2: HashMap<String, String> = HashMap::new();
58    column2.insert("name".to_string(), "string".to_string());
59    column2.insert("type".to_string(), "string".to_string());
60    let mut column3: HashMap<String, String> = HashMap::new();
61    column3.insert("name".to_string(), "last_name".to_string());
62    column3.insert("type".to_string(), "string".to_string());
63    let mut columns = vec![];
64    columns.push(column1);
65    columns.push(column2);
66    columns.push(column3);
67    let res_tables = db_interface.create_table(&table_name, columns);
68    println!("({})", res_tables);
69  };
70  }
71  pub fn test_drop_table(database_name: String, table_name: String) {
72    let mut db_interface = DatabaseInterface {
73      is_connect: false,
74      database: None,
75    };
76    let res = db_interface.select_database(&database_name);
77    match res {
78      true => {
79        println!("Database berhasil terhubung")
80       } false => println!("Database tidak ditemukan"),
81    };
82    db_interface.drop_table(&table_name);
83  };
84  }
85  }
86  }

```

Gambar 1.27: File test_interface.rs bagian 2

```

1 pub fn test_add_column(
2     database_name: String,
3     table_name: String,
4     name: String,
5     data_type: String,
6 ) {
7     let mut db_interface = DatabaseInterface {
8         is_connect: false,
9         database: None,
10    };
11    let res = db_interface.select_database(&database_name);
12    match res {
13        true => {
14            println!("Database berhasil terhubung")
15        }
16        false => println!("Database tidak ditemukan"),
17    };
18    let res = db_interface.add_column_to_table(&table_name, name, data_type);
19    match res {
20        true => {
21            println!("Berhasil membuat column")
22        }
23        _ => {
24            panic!("Something went wrong")
25        }
26    };
27 };
28 }
29 }
30
31 pub fn test_list_column_on_table(database_name: String, table_name: String) {
32     let mut db_interface = DatabaseInterface {
33         is_connect: false,
34         database: None,
35    };
36     let res = db_interface.select_database(&database_name);
37     match res {
38        true => {
39            println!("Database berhasil terhubung")
40        }
41        false => println!("Database tidak ditemukan"),
42    };
43    let res_columns = db_interface.list_column_on_table(table_name);
44    println!("{:?}", res_columns);
45 }
46
47 pub fn test_delete_column_on_table(
48     database_name: String,
49     table_name: String,
50     column_name: String,
51 ) {
52     let mut db_interface = DatabaseInterface {
53         is_connect: false,
54         database: None,
55    };
56    let res = db_interface.select_database(&database_name);
57    match res {
58        true => {
59            println!("Database berhasil terhubung")
60        }
61        false => println!("Database tidak ditemukan"),
62    };
63    let res_columns = db_interface.delete_column_on_table(table_name, column_name);
64    println!("{:?}", res_columns);
65 }
66
67 pub fn test_add_data(database_name: String, table_name: String, data: HashMap<String, String>) {
68     let mut db_interface = DatabaseInterface {
69         is_connect: false,
70         database: None,
71    };
72    let res = db_interface.select_database(&database_name);
73    match res {
74        true => {
75            println!("Database berhasil terhubung")
76        }
77        false => println!("Database tidak ditemukan"),
78    };
79    let res = db_interface.add_data(&table_name, data);
80    match res {
81        true => {
82            println!("Berhasil membuat data")
83        }
84        _ => {
85            panic!("Something went wrong")
86        }
87    };
88 };
89 }
90 }
91 }
```

Gambar 1.28: File test_interface.rs bagian 3

```

1  pub fn test_get_data(database_name: String, table_name: String) {
2      let mut db_interface = DatabaseInterface {
3          is_connect: false,
4          database: None,
5      };
6      let res = db_interface.select_database(&database_name);
7      match res {
8          true => {
9              println!("Database berhasil terhubung")
10         }
11         false => println!("Database tidak ditemukan"),
12     };
13 }
14 db_interface.get_data(&table_name);
15 }
16 }
17
18 pub fn test_search_data(
19     database_name: String,
20     table_name: String,
21     column_name: String,
22     value: String,
23 ) {
24     let mut db_interface = DatabaseInterface {
25         is_connect: false,
26         database: None,
27     };
28     let res = db_interface.select_database(&database_name);
29     match res {
30         true => {
31             println!("Database berhasil terhubung")
32         }
33         false => println!("Database tidak ditemukan"),
34     };
35     db_interface.search_data(&table_name, column_name, value);
36 }
37 }
38
39 pub fn test_update_data(
40     database_name: String,
41     table_name: String,
42     where_data: HashMap<String, String>,
43     updated_data: HashMap<String, String>,
44 ) {
45     let mut db_interface = DatabaseInterface {
46         is_connect: false,
47         database: None,
48     };
49     let res = db_interface.select_database(&database_name);
50     match res {
51         true => {
52             println!("Database berhasil terhubung")
53         }
54         false => println!("Database tidak ditemukan"),
55     };
56     let res = db_interface.update_data(&table_name, where_data, updated_data);
57     match res {
58         true => {
59             println!("Berhasil mengubah data")
60         }
61         false => panic!("Gagal mengubah data"),
62     };
63 }
64 }
```

Gambar 1.29: File test_interface.rs bagian 4

```

1  pub fn test_delete_data(
2      database_name: String,
3      table_name: String,
4      where_data: HashMap<String, String>,
5  ) {
6      let mut db_interface = DatabaseInterface {
7          is_connect: false,
8          database: None,
9      };
10     let res = db_interface.select_database(&database_name);
11     match res {
12         true => {
13             println!("Database berhasil terhubung")
14         }
15         false => println!("Database tidak ditemukan"),
16     };
17
18     let res = db_interface.delete_data(&table_name, where_data);
19     match res {
20         true => {
21             println!("Berhasil menghapus data")
22         }
23         false => panic!("Gagal menghapus data"),
24     };
25 }
26
27 pub fn test_create_join_data() {
28     TestDatabaseInterface::test_create_database("articles".to_string());
29     TestDatabaseInterface::test_create_table("articles".to_string(), "users".to_string());
30     TestDatabaseInterface::test_add_column(
31         "articles".to_string(),
32         "users".to_string(),
33         "id".to_string(),
34         "integer".to_string(),
35     );
36     TestDatabaseInterface::test_add_column(
37         "articles".to_string(),
38         "users".to_string(),
39         "first_name".to_string(),
40         "string".to_string(),
41     );
42     TestDatabaseInterface::test_add_column(
43         "articles".to_string(),
44         "users".to_string(),
45         "last_name".to_string(),
46         "string".to_string(),
47     );
48
49     let data: Vec<HashMap<String, String>> = vec![
50         HashMap::from([
51             ("id".to_string(), "0".to_string()),
52             ("first_name".to_string(), "Farhan".to_string()),
53             ("last_name".to_string(), "Abdul".to_string()),
54         ]),
55         HashMap::from([
56             ("id".to_string(), "1".to_string()),
57             ("first_name".to_string(), "Akbar".to_string()),
58             ("last_name".to_string(), "Maulana".to_string()),
59         ]),
60         HashMap::from([
61             ("id".to_string(), "2".to_string()),
62             ("first_name".to_string(), "Daffa".to_string()),
63             ("last_name".to_string(), "Haryadi".to_string()),
64         ]),
65         HashMap::from([
66             ("id".to_string(), "3".to_string()),
67             ("first_name".to_string(), "Hanif".to_string()),
68             ("last_name".to_string(), "Ramadhan".to_string()),
69         ]),
70         HashMap::from([
71             ("id".to_string(), "4".to_string()),
72             ("first_name".to_string(), "Rudiantyah".to_string()),
73             ("last_name".to_string(), "Wijaya".to_string()),
74         ]),
75     ];
76     for item in data {
77         TestDatabaseInterface::test_add_data("articles".to_string(), "users".to_string(), item);
78     }
79 }
80

```

Gambar 1.30: File test_interface.rs bagian 5

```

1 // DatabaseInterface trait yang digunakan untuk mengeksekusi query ke database
2 trait DatabaseInterface {
3     type Row = Map[String, String]
4     type Table = Map[String, List[Row]]
5
6     def select_database(database_name: String): Unit
7     def insert_table(table_name: String, column_name: String, table_join: String, column_join: String, join_type: String): Unit
8     def drop_table(table_name: String): Unit
9 }
10
11 trait DatabaseImplementation extends DatabaseInterface {
12     var db_map: Map[String, Table] = Map()
13
14     def select_database(database_name: String): Unit = {
15         if (db_map.contains(database_name)) {
16             db_map.get(database_name).foreach { row: Row =>
17                 println(s"Row: $row")
18             }
19         } else {
20             println("Database tidak ditemukan")
21         }
22     }
23
24     def insert_table(table_name: String, column_name: String, table_join: String, column_join: String, join_type: String): Unit = {
25         if (db_map.contains(table_name)) {
26             db_map.get(table_name) match {
27                 case Some(table: Table) => {
28                     val new_table = table ++ Map(column_name -> List("New Data"))
29                     db_map.update(table_name, new_table)
30                 }
31             }
32         } else {
33             db_map += (table_name -> Map(column_name -> List("New Data")))
34         }
35     }
36
37     def drop_table(table_name: String): Unit = {
38         db_map.get(table_name) match {
39             case Some(table: Table) => db_map -= table_name
40         }
41     }
42 }
43
44 class TestInterface {
45     def main(args: Array[String]): Unit = {
46         val db_interface: DatabaseInterface = new DatabaseImplementation
47
48         db_interface.select_database("test")
49
50         db_interface.insert_table("test", "id", "test", "test", "test")
51
52         db_interface.select_database("test")
53
54         db_interface.drop_table("test")
55
56     }
57 }

```

Gambar 1.31: File test_interface.rs bagian 6

```

1 pub fn test_join_table(
2     database_name: String,
3     table_name: String,
4     column_name: String,
5     table_join: String,
6     column_join: String,
7     join_type: String,
8 ) {
9     let mut db_interface = DatabaseInterface {
10         is_connect: false,
11         database: None,
12     };
13
14     let res = db_interface.select_database(&database_name);
15     match res {
16         true => {
17             println!("Database berhasil terhubung")
18         }
19         false => println!("Database tidak ditemukan"),
20     };
21     db_interface.join_table(table_name, column_name, table_join, column_join, join_type);
22 }
23
24

```

Gambar 1.32: File test_interface.rs bagian 7

```

1 use std::collections::HashMap;
2
3 use serde::Deserialize, Serialize;
4
5 use crate::database::{cell::Cell, column::Column, table::Table, utils};
6
7 pub enum DataType {
8     String,
9     Integer,
10    Null,
11 }
12
13 #[derive(Debug, Serialize, Deserialize)]
14 pub enum InputDataNum {
15     String,
16     Integer(isize),
17     Null,
18 }
19
20 pub fn bytes_to_string(arr_vec: Vec<u8>) -> String {
21     let mut res_str = "".to_string();
22     for i in arr_vec {
23         res_str.push(*i as char);
24     }
25     return res_str;
26 }
27
28 pub fn string_to_bytes(word: String) -> Vec<u8> {
29     let mut vec: Vec<u8> = Vec::new();
30     for n in word.chars() {
31         vec.push(n as u8);
32     }
33     return vec;
34 }
35
36 pub fn integer_to_bytes(num: isize) -> [u8; (isize::BITS / 8) as usize] {
37     let bytes: [u8; (isize::BITS / 8) as usize] = num.to_le_bytes();
38     return bytes;
39 }
40
41 fn string_array_to_bytes(words: Vec<String>) -> Vec<u8> {
42     let mut vec: Vec<u8> = Vec::new();
43     for word in words {
44         let len_bytes = word.len().to_bytes();
45         vec.append(&len_bytes.to_vec());
46         let mut byte_string = word.to_string().to_bytes();
47         vec.append(&byte_string);
48     }
49     vec
50 }
51
52 fn integer_array_to_bytes(numbers: Vec<isize>) -> Vec<u8> {
53     let mut vec: Vec<u8> = Vec::new();
54     for item in numbers {
55         let len_bytes = item.to_le_bytes();
56         vec.append(&len_bytes.to_vec());
57     }
58     return vec;
59 }
60
61 pub fn bytes_to_integer(arr_vec: Vec<u8>) -> isize {
62     let number = isize::from_le_bytes(arr_vec.try_into().unwrap());
63     return number;
64 }
65
66 fn bytes_to_integer_array(buf_str: &mut Vec<u8>, length: usize) -> Vec<isize> {
67     let mut arr_int = Vec::new();
68     for i in 0..length {
69         let (word_len, arr_values) = buf_str.split_at((usize::BITS / 8) as usize);
70         println!("{}?", word_len);
71         let word_len_u8: [u8; 8] = word_len.try_into().unwrap();
72         let word_len_isize: isize = from_le_bytes(word_len_u8);
73         let (current, rest_of_values) = arr_values.split_at(word_len_isize);
74         let current_arr_buff = current.to_vec();
75         buf_str = rest_of_values.to_vec();
76         let string_result = bytes_to_integer(current_arr_buff);
77         arr_int.push(string_result);
78     }
79     return arr_int;
80 }
81
82 fn bytes_to_string_array(buf_str: &mut Vec<u8>, length: usize) -> Vec<String> {
83     let mut arr_str = Vec::new();
84     for j in 0..length {
85         let (word_len, arr_values) = buf_str.split_at((usize::BITS / 8) as usize);
86         println!("{}?", word_len);
87         let word_len_u8: [u8; 8] = word_len.try_into().unwrap();
88         let word_len_isize: isize = from_le_bytes(word_len_u8);
89         let (current, rest_of_values) = arr_values.split_at(word_len_isize);
90         let current_arr_buff = current.to_vec();
91         buf_str = rest_of_values.to_vec();
92         let string_result = bytes_to_string(current_arr_buff);
93         arr_str.push(string_result);
94         arr_str.push(string_result);
95     }
96     return arr_str;
97 }
98
99 pub fn parse_new_column(column: HashMap<String, String>) -> Column {
100     let name = match column.get("name") {
101         Some(val) => val.to_owned(),
102         None => panic!("Please input key name!"),
103     };
104     let data_type = match column.get("type") {
105         Some(val) => {
106             if val == "string" {
107                 String
108             } else if val == "integer" {
109                 DataType::Integer
110             } else {
111                 panic!("Incorrect Type")
112             }
113         }
114         None => panic!("Please input key data type!"),
115     };
116     Column {
117         name: name,
118         data_type: data_type,
119         rows: Vec::new(),
120     }
121 }
122 }
```

Gambar 1.33: File utils.rs

DAFTAR RIWAYAT HIDUP



FARHAN DEWANTA SYAHPUTRA. Lahir di Jakarta, 11 Oktober 2001. Anak Ketiga dari pasangan Bapak Yusirwansyah dan Ibu Dewi Banyuwati. Saat ini penulis tinggal di Jl. Teratai Putih 1 Nomor 161 RT 004/RW 004, Kelurahan Malaka Sari, Kecamatan Duren Sawit, Kota Jakarta Timur, Provinsi DKI Jakarta.

No. Ponsel : 081291335215

Email : farhan.abdulhamid11@gmail.com

Riwayat Pendidikan : Penulis mengikuti pendidikan sekolah dasar di SDS Tadika Puri pada tahun 2007 - 2013.

Setelah itu, penulis melanjutkan pendidikan di SMPN 213 Jakarta pada tahun 2013 - 2016. Kemudian penulis melanjutkan pendidikan di SMAN 12 Jakarta pada tahun 2016 - 2019. Setelah lulus, penulis melanjutkan perkuliahan di Universitas Negeri Jakarta pada tahun 2019.

Riwayat Organisasi : Selama di bangku perkuliahan, penulis terlibat dalam organisasi Default Program Studi Ilmu Komputer sebagai Staff dan Ketua Departemen Mobile Development periode 2021-2022.

METADATA

Judul : Perancangan dan Implementasi Prototype
Database Engine Berbasis Structure Oriented Programming Menggunakan Rust

Nama : Farhan Dewanta Syahputra

NIM : 1313619017

Pembimbing I : Muhammad Eka Suryana, M.Kom

Pembimbing II : Med Irzal, M.Kom

Keyword : 1. Database
 2. Index
 3. Basis Data
 4. Sistem Terdistribusi