ITS
Institut
Teknologi
Sepuluh Nopember

TUGAS AKHIR - SM234801

PENINGKATAN KINERJA KUERI DATA WAREHOUSE MENGGUNAKAN MEKANISME PARTISI MEMORY-AWARE BERBASIS PROADAPT

HAFIDZ MULIA

NRP 5002221022

Dosen Pembimbing

Dr. Budi Setiyono, S.Si., M.T.

NIP 19720207 199702 1 001

Program Studi Sarjana

Departemen Matematika

Fakultas Sains dan Analitika Data

Institut Teknologi Sepuluh Nopember

Surabaya

2026

TUGAS AKHIR - SM234801

PENINGKATAN KINERJA KUERI DATA WAREHOUSE MENGGUNAKAN MEKANISME PARTISI MEMORY-AWARE BERBASIS PROADAPT

HAFIDZ MULIA
NRP 5002221022

Dosen Pembimbing
Dr. Budi Setiyono, S.Si., M.T.
NIP 19720207 199702 1 001

Program Sarjana
Departemen Matematika
Fakultas Sains dan Analitika Data
Institut Teknologi Sepuluh Nopember
Surabaya
2026

FINAL PROJECT - SM234801

PERFORMANCE IMPROVEMENT OF DATA WAREHOUSE QUERIES USING PROADAPT-BASED MEMORY-AWARE PARTITIONING MECHANISM

HAFIDZ MULIA

NRP 5002221022

Supervisor

Dr. Budi Setiyono, S.Si., M.T.

NIP 19720207 199702 1 001

Bachelor Program

Department of Mathematics

Faculty of Scientics

Institut Teknologi Sepuluh Nopember

Surabaya

2026

LEMBAR PENGESAHAN

PENINGKATAN KINERJA KUERI *DATA WAREHOUSE* MENGGUNAKAN MEKANISME PARTISI *MEMORY-AWARE* BERBASIS PROADAPT

TUGAS AKHIR

Diajukan untuk memenuhi salah satu syarat
memperoleh gelar Sarjana Matematika pada
Program Studi S-1 Matematika
Departemen Matematika
Fakultas Sains dan Analitika Data
Institut Teknologi Sepuluh Nopember

Oleh: **Hafidz Mulia**
NRP 5002221022

Surabaya, Juli 2026
Disetujui oleh Tim Penguji Tugas Akhir:

Pembimbing:

1. Dr. Budi Setiyono, S.Si., M.T. (.....)
   NIP 19720207 199702 1 001

Penguji:

1. Dr. Drs. Bandung Arry S., MI.Komp (.....)
   NIP 19630605 198903 1 003
2. Dr. Darmaji, S.Si, M.T. (.....)
   NIP 19691015 199412 1 001
3. Dr. Dieky Adzkiya, S.Si, M.Si. (.....)
   NIP 19830517 200812 1 003

Mengetahui,
Kepala Departemen Matematika,
Fakultas Sains dan Analitika Data,

Dr. Didik Khusnul Arif, S.Si., M.Si.
NIP 19730930 199702 1 001

APPROVAL SHEET

PERFORMANCE IMPROVEMENT OF DATA WAREHOUSE QUERIES USING PROADAPT-BASED MEMORY-AWARE PARTITIONING MECHANISM

FINAL PROJECT

Submitted to fulfill one of the requirements
for obtaining a degree Bachelor of Mathematics at
Bachelor Program of Mathematics
Department of Mathematics
Faculty of Science and Data Analytics
Institut Teknologi Sepuluh Nopember

By: **HAFIDZ MULIA**
NRP 5002221022

Surabaya, July 2026
Approved By:

Advisor:

1. Dr. Budi Setiyono, S.Si., M.T. (.....)
   NIP 19720207 199702 1 001

Examiners:

1. Dr. Drs. Bandung Arry S., MI.Komp (.....)
   NIP 19630605 198903 1 003
2. Dr. Darmaji, S.Si, M.T. (.....)
   NIP 19691015 199412 1 001
3. Dr. Dieky Adzkiya, S.Si, M.Si. (.....)
   NIP 19830517 200812 1 003

Acknowledged by,
Head of Mathematics Department,
Faculty of Science and Data Analytics,

Dr. Didik Khusnul Arif, S.Si., M.Si.
NIP 19730930 199702 1 001

PERNYATAAN ORISINALITAS

Yang bertanda tangan di sini:

Nama Mahasiswa / NRP	:	Hafidz Mulia / 5002221022
Departemen	:	Matematika
Dosen Pembimbing	:	Dr. Budi Setiyono, S.Si., M.T. / 19720207 199702 1 001

dengan ini menyatakan bahwa Tugas Akhir dengan judul "**PENINGKATAN KINERJA KUERI *DATA WAREHOUSE* MENGGUNAKAN MEKANISME PARTISI *MEMORY-AWARE* BERBASIS PROADAPT**" adalah hasil karya sendiri, bersifat orisinal, dan ditulis dengan mengikuti kaidah penulisan ilmiah.

Bilamana di kemudian hari ditemukan ketidaksesuaian dengan pernyataan ini, maka saya bersedia menerima sanksi sesuai dengan ketentuan yang berlaku di Institut Teknologi Sepuluh Nopember.

Surabaya, 22 Juli 2026

Mengetahui,
Dosen Pembimbing,

Dr. Budi Setiyono, S.Si., M.T.
NIP. 19720207 199702 1 001

Mahasiswa,

Hafidz Mulia
NRP. 5002221022

STATEMENT OF ORIGINALITY

The undersigned below:

Student's Name/NRP	:	Hafidz Mulia / 5002221022
Department	:	Mathematics
Advisor	:	Dr. Budi Setiyono, S.Si., M.T. / 19720207 199702 1 001

Hereby declare that the Final Project with the title of "**Performance Improvement of Data Warehouse Queries Using PROADAPT-Based Memory-Aware Partitioning Mechanism**" is the result of my own work, is original, and is written by following the rules of scientific writing.

If in the future there is a discrepancy with this statement, then I am willing to accept sanctions in accordance with the provisions that apply at Institut Teknologi Sepuluh Nopember.

Surabaya, July 22, 2026

Acknowledged,
Advisor

Dr. Budi Setiyono, S.Si., M.T.
NIP 19720207 199702 1 001

Student

Hafidz Mulia
NRP 5002221022

ABSTRAK

Peningkatan Kinerja Kueri *Data Warehouse* Menggunakan Mekanisme Partisi *Memory-Aware* Berbasis PROADAPT

Nama Mahasiswa / NRP	:	Hafidz Mulia / 5002221022
Departemen	:	Matematika, Fakultas Sains dan Analitika Data-ITS
Dosen Pembimbing	:	Dr. Budi Setiyono, S.Si., M.T.

Abstrak

Pertumbuhan data historis meningkatkan kebutuhan analitik jangka panjang pada *data warehouse*. Kinerja kueri dipengaruhi oleh konfigurasi *chunk time interval* dan kebijakan kompresi yang menentukan susunan data historis pada TimescaleDB. Penelitian ini mengadaptasi PROADAPT pada PostgreSQL/TimescaleDB *single-VM*. Tahap *offline* menilai kandidat konfigurasi menggunakan fungsi utilitas yang menggabungkan latensi, efektivitas *chunk pruning*, *memory score* berbasis BMR dan WSS, *storage size*, serta penalti *overhead*. Tahap *online* menjalankan lima *window* dengan persentase *predefined query* sebesar 100%, 75%, 50%, 25%, dan 0%. Hasil tahap *offline* memilih interval 3 hari pada Skenario 1 dengan utilitas 0,4056, kombinasi interval 14 hari dan `compress_after` 7 hari pada Skenario 2 dengan utilitas 0,4700, serta kombinasi interval 7 hari dan `compress_after` 3 hari pada Skenario 3 dengan utilitas 0,3065. Baseline menghasilkan latensi rata-rata 503,562 ms. Tahap *online* pada Skenario 1, Skenario 2, dan Skenario 3 menghasilkan latensi rata-rata 522,526 ms, 397,256 ms, dan 377,641 ms. BMR ketiga skenario mendapat hasil yang lebih rendah daripada baseline.

Kata kunci: *Data Warehouse; TimescaleDB; PROADAPT; Partisi Adaptif; Kompresi Memory-Aware.*

ABSTRACT

Performance Improvement of Data Warehouse Queries Using PROADAPT-Based Memory-Aware Partitioning Mechanism

Student Name / ID	:	Hafidz Mulia / 5002221022
Department	:	Mathematics, Faculty of Science and Data Analytics-ITS
Advisor	:	Dr. Budi Setiyono, S.Si., M.T.

Abstract

The growth of historical data increases the demand for long-term analytics in data warehouses. Query performance is affected by the *chunk time interval* and compression policy that determine how historical data are organized in TimescaleDB. This study adapts PROADAPT to a PostgreSQL/TimescaleDB *single-VM* environment. The *offline* stage evaluates configuration candidates using a utility function that combines latency, chunk-pruning effectiveness, a BMR- and WSS-based memory score, storage size, and an overhead penalty. The *online* stage executes five windows containing 100%, 75%, 50%, 25%, and 0% predefined queries. The *offline* stage selects a 3-day interval in Scenario 1 with a utility of 0.4056, a 14-day interval with a 7-day `compress_after` in Scenario 2 with a utility of 0.4700, and a 7-day interval with a 3-day `compress_after` in Scenario 3 with a utility of 0.3065. The baseline produces an average latency of 503.562 ms. The *online* stages over Scenarios 1, 2, and 3 produce average latencies of 522.526 ms, 397.256 ms, and 377.641 ms. All three scenarios produce lower BMR and higher WSS than the baseline. These results show that the configuration changes have different effects on latency and block usage as the query composition changes.

Keywords: *Data Warehouse, TimescaleDB, PROADAPT, Adaptive Partitioning, Memory-Aware Compression.*

KATA PENGANTAR

Puji syukur penulis panjatkan kepada Tuhan Yang Maha Esa karena atas rahmat dan karunia-Nya tugas akhir ini dapat diselesaikan. Tugas akhir ini berjudul:

"PENINGKATAN KINERJA KUERI *DATA WAREHOUSE* MENGGUNAKAN MEKANISME PARTISI *MEMORY-AWARE* BERBASIS PROADAPT"

Tugas akhir ini disusun untuk memenuhi salah satu syarat kelulusan pada Program Sarjana Departemen Matematika, Fakultas Sains dan Analitika Data, Institut Teknologi Sepuluh Nopember.

Selama proses penyusunan, penulis memperoleh dukungan, arahan, dan bantuan dari berbagai pihak. Oleh karena itu, penulis menyampaikan terima kasih kepada:

1. Keluarga yang telah memberikan dukungan, nasihat, dan motivasi sehingga penulis dapat menyelesaikan tugas akhir ini.
2. Bapak Dr. Didik Khusnul Arif, M.Si selaku Kepala Departemen Matematika Institut Teknologi Sepuluh Nopember.
3. Bapak Dr. Budi Setiyono, S.Si., M.T. selaku dosen pembimbing yang telah memberikan arahan dengan penuh kesabaran kepada penulis selama penyusunan tugas akhir ini.
4. Bapak Dr. Drs. Bandung Arry S., MI.Komp, Bapak Dr. Darmaji, S.Si, M.T., dan Bapak Dr. Dieky Adzkiya, S.Si, M.Si. dosen penguji yang telah memberikan arahan, kritik, saran, dan masukan yang membangun dalam pengerjaan tugas akhir ini.
5. Ibu Endah Rokhmati Merdika Putri, S.Si., M.T., Ph.D selaku dosen wali yang telah memberikan arahan dan bantuan kepada penulis selama masa perkuliahan.
6. Bapak/Ibu dosen pengajar yang tidak dapat penulis sebutkan satu per satu, serta segenap karyawan, tenaga kependidikan, dan keluarga besar Departemen Matematika Institut Teknologi Sepuluh Nopember atas dukungan dan bantuannya.

Penulis menyadari bahwa tugas akhir ini masih dapat disempurnakan. Oleh karena itu, masukan yang membangun tetap penulis harapkan untuk perbaikan penulisan selanjutnya. Semoga tugas akhir ini dapat memberikan manfaat bagi pengembangan kajian *data warehouse* dan optimasi kueri analitik.

Surabaya, 22 Juli 2026

Hafidz Mulia

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB I
PENDAHULUAN

Bab ini menjelaskan latar belakang dari permasalahan yang diangkat pada penelitian ini. Selain itu bab ini juga memaparkan rumusan masalah, batasan masalah, tujuan penelitian, serta manfaat penelitian yang diharapkan.

1.1 Latar Belakang

Perkembangan sistem digital menyebabkan volume data selalu terus bertambah. Data tersebut tidak hanya digunakan untuk melihat kondisi saat ini, tetapi juga data masa lalu dipertahankan sebagai riwayat untuk mendukung pelaporan, analisis, dan pengambilan keputusan. Pada konteks tersebut, *data warehouse* digunakan untuk mengintegrasikan data dari berbagai sumber dan menyimpan data historis secara terstruktur (Inmon, 2005). Seiring bertambahnya data yang disimpan, kueri analitik perlu memproses rentang data yang semakin luas sehingga kinerja kueri menjadi aspek penting dalam pengelolaan *data warehouse*. Salah satu strategi yang digunakan untuk mengurangi data yang perlu dibaca adalah partisi data. Namun, konfigurasi partisi yang ditetapkan pada inisialisasi pembangunan *data warehouse* dapat menjadi kurang sesuai ketika rentang waktu, bentuk agregasi, dan komposisi kueri mengalami perubahan (Benkrid, Bellatreche, Mestoui, dan Ordonez, 2022).

Kebutuhan pengelolaan data historis berbasis waktu dapat dijumpai pada layanan Data Online BMKG yang dikelola oleh Direktorat Data dan Komputasi BMKG. Layanan tersebut menyediakan data meteorologi, klimatologi, dan geofisika berdasarkan stasiun, parameter, serta waktu pengamatan (Direktorat Data dan Komputasi BMKG, 2024). Pada kasus yang serupa, administrator basis data dan *data engineer* perlu menjaga ketersediaan data historis sekaligus mempertahankan kinerja kueri ketika volume data dan kebutuhan analitik berkembang.

Sejumlah penelitian telah mengembangkan pengelolaan data berdasarkan pola kueri dan karakteristik sistem. AdaptDB menyesuaikan partisi secara bertahap untuk mengurangi pemindahan data pada proses penggabungan tabel dalam lingkungan terdistribusi (Lu, Shanbhag, Jindal, dan Madden, 2017). Martin dan Davis memanfaatkan riwayat kueri untuk menentukan penempatan data pada penyimpanan *hot*, *warm*, dan *cold* sesuai dengan kapasitas penyimpanan cepat yang tersedia (Martin dan Davis, 2021). Sementara itu, Lee dkk. menunjukkan bahwa konfigurasi TimescaleDB seperti interval *chunk*, penggunaan kompresi, jumlah *worker*, dan

perangkat penyimpanan dapat mempengaruhi kinerja kueri(Lee, Son, dan Kim, 2023).

PROADAPT diperkenalkan sebagai kerangka partisi adaptif yang bersifat proaktif dan memperhatikan pola kueri. Pada tahap offline, PROADAPT menggunakan riwayat kueri untuk membentuk dan menilai skema partisi awal. Setelah skema diterapkan, tahap online memantau kueri baru untuk menilai kesesuaian skema terhadap perubahan *workload*. PROADAPT juga memanfaatkan penulisan ulang kueri agar bagian data yang tidak berkaitan dengan kebutuhan kueri dapat diabaikan. Dengan alur tersebut, PROADAPT menyediakan mekanisme untuk mempertahankan kesesuaian konfigurasi partisi ketika pola akses data berubah (Benkrid dkk., 2022).

Meskipun menyediakan alur adaptasi yang lengkap, PROADAPT pada penelitian awalnya dirancang untuk partisi multidimensi dalam lingkungan paralel dan terdistribusi. Karakteristik tersebut berbeda dengan TimescaleDB yang mempartisi data berdasarkan waktu melalui *hypertable* dan *chunk*, serta menyediakan kebijakan kompresi untuk pengelolaan data historis (Tiger Data, 2026b, 2026a). Oleh karena itu, PROADAPT pada penelitian ini digunakan sebagai dasar prinsip dan alur mekanisme. Struktur yang dipertahankan meliputi seleksi konfigurasi tahap *offline*, penilaian berbasis utilitas, pemantauan *workload* pada tahap *online*, dan *query rewriter*. Implementasinya disesuaikan dengan objek partisi waktu, ruang kandidat *chunk time interval* dan `compress_after`, serta lingkungan PostgreSQL/TimescaleDB *single-VM*.

Pada penelitian ini, peningkatan kinerja kueri merujuk pada penurunan latensi kueri terhadap konfigurasi *baseline*. Kemudian *memory-aware* dibatasi pada perilaku akses blok terhadap memori utama (RAM) melalui *shared buffers* PostgreSQL yang dinilai menggunakan BMR dan WSS. Efektivitas *chunk pruning*, *storage size*, dan *overhead* digunakan sebagai komponen pendukung dalam fungsi utilitas untuk meningkatkan kinerja kueri.

Berdasarkan uraian tersebut, penelitian ini disusun dengan judul *Peningkatan Kinerja Kueri Data Warehouse Menggunakan Mekanisme Partisi Memory-Aware Berbasis PROADAPT*. Hasil penelitian diharapkan dapat menjadi acuan bagi administrator basis data dan *data engineer* dalam memilih konfigurasi partisi dan kompresi untuk pengelolaan data historis berbasis waktu pada lingkungan dengan sumber daya memori yang terbatas

1.2 Rumusan Masalah

Berdasarkan latar belakang yang telah diuraikan, diperoleh rumusan masalah pada penelitian ini sebagai berikut

1. Bagaimana mengadaptasi mekanisme berbasis PROADAPT agar dapat diterapkan pada *data warehouse* berbasis PostgreSQL/TimescaleDB dalam lingkungan *single-VM*?

2. Bagaimana merumuskan fungsi *utility* yang *memory-aware* untuk menilai dan

memilih konfigurasi partisi serta kebijakan kompresi pada TimescaleDB?

3. Sejauh mana pendekatan berbasis PROADAPT memberikan peningkatan kinerja kueri dibandingkan konfigurasi baseline yang statis berdasarkan metrik evaluasi yang ditetapkan?

1.3 Batasan Masalah

Berdasarkan rumusan masalah yang ada, ruang lingkup pada penelitian ini dibatasi dengan

1. Implementasi dilakukan pada lingkungan *single-VM* menggunakan PostgreSQL dengan ekstensi TimescaleDB.
2. Data yang digunakan adalah dataset sekunder yang berasal dari *Intel Berkeley Research Lab* (Intel Lab Data) dan diposisikan sebagai data historis detail dengan atribut waktu sebagai dasar analisis.
3. Ruang lingkup optimasi dibatasi pada konfigurasi partisi *chunk time interval* dan kebijakan kompresi pada *hypertable*.
4. Peningkatan kinerja kueri pada penelitian ini diukur berdasarkan latensi kueri.
5. *Memory-aware* pada penelitian ini dibatasi pada perilaku baca blok terhadap memori utama (RAM) melalui *shared buffers* PostgreSQL.

1.4 Tujuan Penelitian

Berdasarkan rumusan masalah yang ada, serta batasan masalah yang telah diuraikan, tujuan penelitian ini adalah

1. Mengadaptasi mekanisme berbasis PROADAPT agar dapat diterapkan pada *data warehouse* berbasis PostgreSQL/TimescaleDB dalam lingkungan *single-VM*.
2. Menyusun fungsi *utility* yang *memory-aware* sebagai dasar pemilihan konfigurasi partisi dan kebijakan kompresi pada TimescaleDB.
3. Mengevaluasi dan membandingkan kinerja konfigurasi berbasis PROADAPT dengan konfigurasi baseline yang bersifat statis menggunakan metrik yang ditetapkan.

1.5 Manfaat Penelitian

Dengan menjawab rumusan masalah yang telah diuraikan, penelitian ini diharapkan dapat memberikan manfaat sebagai berikut

1. Memberikan kontribusi dalam perancangan dan evaluasi mekanisme partisi adaptif *memory-aware* berbasis PROADAPT untuk meningkatkan kinerja kueri *data warehouse* pada PostgreSQL/TimescaleDB.

2. Memberikan acuan bagi administrator basis data dan *data engineer* dalam pemilihan konfigurasi partisi dan kebijakan kompresi yang seimbang antara kinerja kueri dan efisiensi sumber daya pada lingkungan *single-VM*.

BAB II
TINJAUAN PUSTAKA

Bab ini membahas landasan teori dan konsep dasar yang menjadi acuan penelitian ini. Penjelasan dimulai dengan tinjauan penelitian terdahulu yang relevan serta landasan dalam pemilihan metode yang digunakan.

2.1 Penelitian Terdahulu

Penelitian terkait peningkatan kinerja kueri melalui partisi data telah banyak dilakukan oleh para peneliti sebelumnya. Partisi digunakan untuk membagi data berukuran besar menjadi bagian yang lebih kecil agar kueri tidak selalu membaca seluruh isi tabel. Pada kondisi pola kueri yang stabil, konfigurasi partisi dapat ditentukan sejak awal dan digunakan dalam jangka waktu yang panjang. Namun, ketika kebutuhan analisis berubah, partisi yang semula sesuai dapat menyebabkan lebih banyak data dibaca atau dipindahkan. Penelitian mengenai partisi data kemudian berkembang dengan memanfaatkan riwayat kueri, perubahan *workload*, ketersediaan penyimpanan, dan kondisi perangkat yang digunakan oleh sistem.

Lu dkk. (2017) mengembangkan AdaptDB untuk meningkatkan kinerja kueri analitik yang melibatkan penggabungan (*join*) beberapa tabel. Pada sistem basis data terdistribusi, proses penggabungan tabel dapat membutuhkan pemindahan data antar *node* dalam jumlah besar. AdaptDB mengurangi biaya tersebut melalui *hyper-join*, yaitu proses penggabungan yang hanya membaca blok data yang saling berkaitan. Ketika pola penggabungan tabel berubah, AdaptDB tidak langsung membagi ulang seluruh data, tetapi memindahkan sebagian blok secara bertahap melalui *smooth repartitioning*. Pengujian menggunakan TPC-H dan data aktual pada Spark dan HDFS menunjukkan bahwa AdaptDB meningkatkan kinerja kueri sebesar 2–3 kali dibandingkan pendekatan yang menggunakan pemindaian data dan *shuffle join*. Meskipun demikian, AdaptDB berfokus pada biaya penggabungan tabel dan perpindahan data dalam lingkungan basis data terdistribusi (Lu dkk., 2017).

Penelitian yang dilakukan oleh Martin dan Davis (2021) meneliti seputar penggunaan riwayat kueri untuk mengatur penempatan data pada *logical data warehouse*, yaitu arsitektur yang menggabungkan akses terhadap beberapa sumber data tanpa memindahkan seluruh data ke satu tempat. Penelitian dilakukan pada data kesehatan berskala besar dengan mengumpulkan *workload* selama 100 hari. Setelah proses penyaringan, diperoleh 1.061 kueri OLAP yang menggunakan 4.824 kolom.

Kolom yang sering digunakan secara bersamaan dikelompokkan, kemudian ditempatkan pada penyimpanan *hot*, *warm*, atau *cold* sesuai tingkat kepentingannya menggunakan algoritma *Best-Worst Neighbor* menghasilkan *query coverage* sebesar 76%, yang berarti 76% kueri dapat dilayani menggunakan data yang tersedia pada penyimpanan cepat (*in-memory SAP HANA*). Ketika riwayat kueri selama 30 dan 70 hari digunakan untuk menentukan penempatan data pada periode berikutnya, cakupan yang diperoleh masing-masing sebesar 65% dan 70%. Penelitian ini menunjukkan bahwa pola kueri dapat digunakan untuk mengatur pemakaian penyimpanan cepat yang kapasitasnya terbatas, tetapi belum membahas perubahan interval partisi dan kebijakan kompresi (Martin dan Davis, 2021).

Benkrid dkk. (2022) mengusulkan PROADAPT untuk mengatasi permasalahan penurunan kinerja partisi ketika pola akses kueri berubah. PROADAPT menggunakan riwayat kueri (*query log*) untuk membentuk konfigurasi partisi awal, kemudian memantau kueri baru selama sistem digunakan. Kueri dengan pola yang serupa dikelompokkan, sedangkan kandidat partisi dinilai menggunakan fungsi utilitas dan dipilih menggunakan *Genetic Algorithm*. PROADAPT juga melakukan penulisan ulang kueri agar bagian data yang tidak berkaitan dengan kebutuhan kueri dapat diabaikan. Pengujian dilakukan menggunakan Star Schema Benchmark pada Postgres-XL dan Spark SQL. Hasilnya menunjukkan bahwa penulisan ulang kueri menghasilkan waktu eksekusi rata-rata 67% lebih baik dibandingkan Spark SQL dan 70% lebih baik dibandingkan Postgres-XL. Namun, PROADAPT dirancang untuk partisi multidimensi pada lingkungan paralel dan terdistribusi (Benkrid dkk., 2022).

Lee dkk. (2023) menganalisis kinerja TimescaleDB dalam mengelola data *time-series*. Pengujian dilakukan menggunakan Time Series Benchmark Suite dengan mengubah interval *chunk*, penggunaan kompresi, jumlah *worker*, dan jenis perangkat penyimpanan. Perangkat yang dibandingkan meliputi HDD, SATA SSD, dan NVMe SSD. Pada interval *chunk* satu menit, penggunaan NVMe SSD menghasilkan waktu eksekusi hingga 33,22 kali lebih cepat dibandingkan HDD. Penggunaan kompresi juga mengurangi ukuran data dari 8,2 GB menjadi 1,5 GB atau sekitar 18% dari ukuran sebelumnya. Hasil tersebut menunjukkan bahwa kinerja TimescaleDB dipengaruhi oleh hubungan antara interval *chunk*, kompresi, dan perangkat penyimpanan. Akan tetapi, konfigurasi pada penelitian tersebut ditentukan sebelum pengujian dan belum disesuaikan berdasarkan perubahan *workload* (Lee dkk., 2023). Ringkasan penelitian terdahulu tersebut disajikan pada Tabel 2.1.

Tabel 2.1: Penelitian terdahulu

Referensi	**Data/Platform**	**Metode**	**Hasil**
Lu dkk. (2017)	TPC-H dan data aktual pada Spark/HDFS	*Hyper-join* dan *smooth repartitioning*	Kinerja kueri meningkat 2–3 kali dibandingkan *scan* dan *shuffle join*.
Martin dan Davis (2021)	Workload OLAP kesehatan pada SQL Server dan SAP HANA	Pengelompokan data dan *Best-Worst Neighbor*	Mencapai 76% *query coverage* dan 65–70% pada pengujian workload berikutnya.
Benkrid dkk. (2022)	SSB pada Postgres-XL dan Spark SQL	Pengelompokan workload, fungsi utilitas, *Genetic Algorithm*, dan *query rewriter*	Waktu eksekusi membaik 67% dibandingkan Spark SQL dan 70% dibandingkan Postgres-XL.
Lee dkk. (2023)	TSBS dan TimescaleDB pada HDD, SATA SSD, dan NVMe SSD	Pengujian interval *chunk*, kompresi, jumlah *worker*, dan perangkat penyimpanan	Performa meningkat hingga 33,22 kali dan kompresi mengurangi data dari 8,2 GB menjadi 1,5 GB.

Dalam ruang lingkup empat penelitian tersebut, setiap penelitian membahas bagian yang berbeda dari permasalahan partisi data. AdaptDB menyesuaikan partisi secara bertahap untuk mengurangi biaya penggabungan tabel pada sistem basis data terdistribusi. Martin dan Davis menggunakan riwayat kueri (*query logs*) untuk menentukan data yang perlu ditempatkan pada penyimpanan cepat. PROADAPT menyediakan mekanisme pemilihan konfigurasi awal dan penyesuaian saat *workload* berubah, sedangkan Lee dkk. menunjukkan pengaruh interval *chunk*, kompresi, dan perangkat penyimpanan terhadap kinerja TimescaleDB. Oleh karena itu, penelitian ini mengadaptasi mekanisme partisi berbasis PROADAPT pada PostgreSQL/TimescaleDB dalam lingkungan *single-VM*. Pendekatan tersebut ditujukan untuk memilih konfigurasi partisi dan kompresi yang tetap sesuai ketika *workload* berubah (*workload shift*), serta menjaga keseimbangan antara kinerja kueri, penggunaan memori (RAM dan *storage disk*), serta biaya adaptasi.

Dalam ruang lingkup empat penelitian tersebut, setiap penelitian membahas bagian yang berbeda dari permasalahan partisi data. AdaptDB menyesuaikan partisi secara bertahap pada sistem terdistribusi, Martin dan Davis menggunakan riwayat kueri untuk menentukan penempatan data, PROADAPT menyediakan mekanisme pemilihan konfigurasi dan pemantauan perubahan *workload*, sedangkan Lee dkk. menunjukkan pengaruh interval *chunk*, kompresi, dan perangkat penyimpanan terhadap kinerja TimescaleDB.

Berdasarkan posisi tersebut, penelitian ini menggunakan PROADAPT sebagai dasar alur seleksi tahap *offline* dan pemantauan tahap *online*, kemudian menyesuaikannya dengan partisi berbasis waktu dan kebijakan kompresi selaku partisi berbasis kolom pada PostgreSQL/TimescaleDB *single-VM*. Penilaian konfigurasi membedakan latensi sebagai ukuran kinerja kueri, serta BMR dan WSS sebagai indikator *memory-aware*.

2.2 Data Warehouse

Data warehouse merupakan arsitektur penyimpanan data yang dirancang untuk mendukung kebutuhan analitik dan pengambilan keputusan. Berbeda dengan database operasional (OLTP) yang mengelola transaksi harian, data warehouse memiliki empat karakteristik, yaitu berorientasi subjek (*subject-oriented*), terintegrasi (*integrated*), memiliki rentang waktu (*time-variant*), dan bersifat permanen (*non-volatile*) (Inmon, 2005). Sifat *time-variant* dan *non-volatile* tersebut menunjukkan bahwa data warehouse berfungsi sebagai *tools* untuk menyimpan tumpukan data historis. Data historis adalah kumpulan rekaman informasi mengenai kejadian, transaksi, atau status suatu entitas di masa lalu yang sengaja dikumpulkan dengan atribut waktu tertentu (Snodgrass dan Ahn, 1985). Dalam lingkungan analitik, data historis ini dipertahankan keaslian data dan tidak akan ditimpa (*overwritten*) oleh data baru ketika terjadi perubahan status. Penyimpanan historis ini difokuskan secara khusus untuk kebutuhan analisis tren, komparasi antar periode, serta penelusuran riwayat dari waktu ke waktu (Chaudhuri dan Dayal, 1997).

Akumulasi data historis disimpan secara terpusat pada sebuah struktur yang dinamakan tabel fakta (*fact table*) (Kimball dan Ross, 2013). Mengingat fungsinya sebagai penyimpan utama data historis yang terus bertambah tanpa adanya penghapusan data, seiring berjalannya waktu, tabel fakta akan memiliki volume baris data yang sangat besar. Agar dapat mendapat informasi dari tumpukan data yang berskala masif tersebut, pengguna perlu menjalankan kueri analitik atau bisa disebut OLAP. Berbeda dengan kueri operasional yang hanya memanipulasi satu atau dua baris data secara spesifik, kueri analitik memiliki karakteristik komputasi yang berat karena sering kali harus memindai (*scan*) dan melakukan agregasi terhadap jutaan hingga miliaran baris data pada tabel fakta (Chaudhuri dan Dayal, 1997). Pemindaian data bervolume besar ini memberikan tekanan yang sangat tinggi pada I/O disk dan kapasitas memori utama sistem. Oleh karena itu, tumpukan data pada tabel fakta tersebut memerlukan strategi untuk mengorganisasi penyimpanan pada level fisik seperti mekanisme partisi data yang adaptif terhadap sumber daya atau memori yang tersedia agar eksekusi kueri analitik dapat terhindar dari pemindaian tabel secara penuh (*full table scan*) yang memicu tekanan terhadap memori dan tingginya latensi.

2.3 Partisi Data

Partisi data merupakan metode membagi sebuah struktur data yang berukuran besar, seperti tabel fakta, menjadi beberapa fragmen data yang lebih kecil dan mudah dikelola (Silberschatz dkk., 2020). Dalam konteks data warehouse yang menyimpan akumulasi data historis, metode partisi yang paling umum dan efektif digunakan adalah partisi berbasis waktu (*time-based partitioning*). Pendekatan ini merupakan bentuk dari partisi rentang (*range partitioning*), di mana setiap baris data dialokasikan ke dalam

fragmen data tertentu berdasarkan interval pada atribut waktu kejadiannya, seperti harian, mingguan, atau bulanan (Elmasri dan Navathe, 2015). Pengelompokan data berdasarkan rentang waktu ini selaras dengan sifat *time-variant* pada data warehouse, di mana laju penambahan data baru selalu berjalan seiring dengan bertambahnya waktu operasional sistem.

Secara matematis, tabel fakta dapat dipandang sebagai himpunan baris R, sedangkan hasil partisinya dinyatakan sebagai keluarga himpunan

$$\mathcal{P} = \{P_1, P_2, \ldots, P_n\}.$$

Setiap partisi memenuhi

$$P_i \subseteq R, \qquad P_i \cap P_j = \varnothing \quad \text{untuk } i \neq j, \qquad \bigcup_{i=1}^{n} P_i = R. \tag{2.1}$$

Pada partisi berbasis waktu, setiap partisi didefinisikan oleh rentang waktu tertentu, yaitu

$$P_i = \{r \in R \mid \tau_i \leq r.\texttt{time} < \tau_{i+1}\}, \tag{2.2}$$

dengan τ_i dan τ_{i+1} sebagai batas awal dan akhir rentang waktu partisi ke-i.

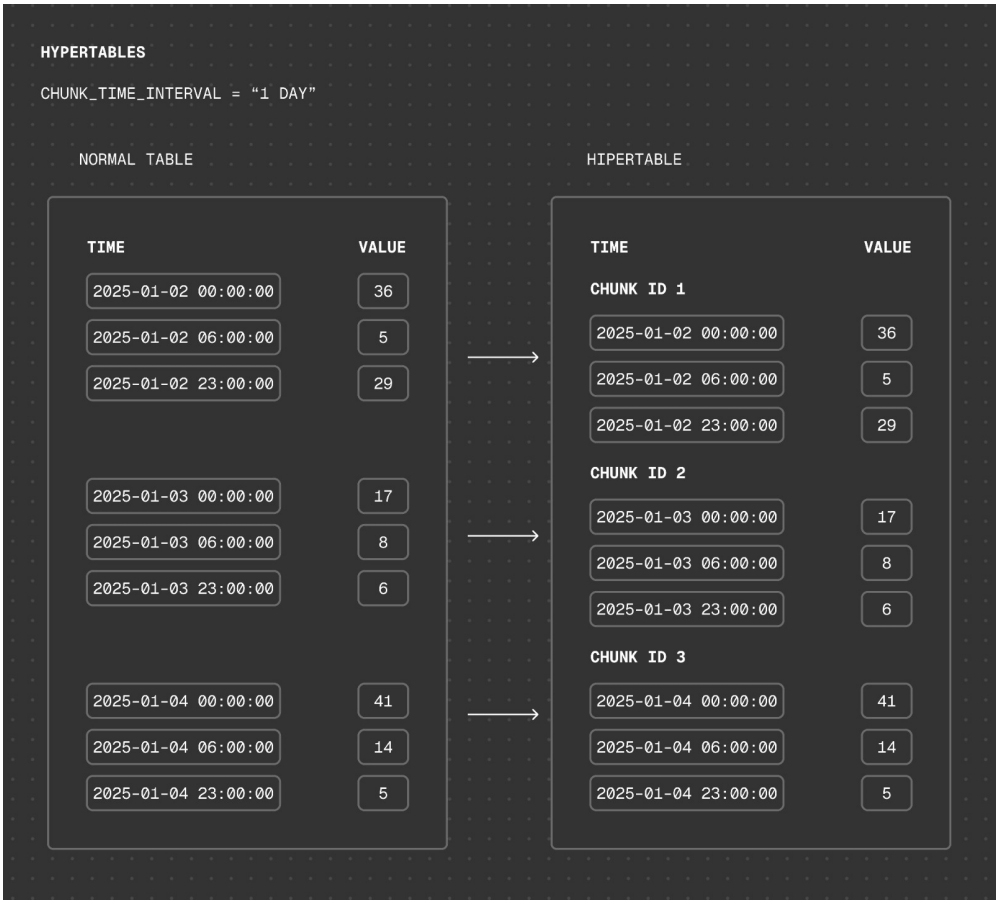
Salah satu kelebihan dari partisi berbasis waktu adalah kemampuan database untuk melakukan pemangkasan partisi (*partition pruning*). *Partition pruning* atau selanjutnya disebut *pruning* adalah mekanisme optimasi eksekusi kueri di mana *query planner* dapat menganalisis filter rentang waktu pada kueri analitik yang dilakukan *user*, dan secara otomatis mengabaikan atau memangkas partisi-partisi fisik yang tidak masuk dalam rentang waktu tersebut (Silberschatz dkk., 2020). Sebagai contoh, jika kueri analitik hanya meminta agregasi data penjualan pada bulan Desember, sistem hanya akan memuat partisi bulan Desember ke dalam memori (RAM) dan mengabaikan partisi dari bulan-bulan lainnya. Mekanisme pemangkasan ini dapat mengurangi jumlah data yang harus dibaca dari penyimpanan disk, sehingga menurunkan biaya I/O, meminimalkan tekanan memori, dan mempercepat waktu respons kueri. Meskipun konsep partisi dan *pruning* ini sangat efisien secara teoritis, cara mengelola interval partisi waktu dalam jumlah besar secara manual sering kali menimbulkan overhead bagi administrator database. Oleh karena itu, pendekatan modern untuk menangani beban kerja *time-series* yang masif biasanya diotomatisasi melalui arsitektur sistem database khusus seperti TimescaleDB.

2.4 TimescaleDB

TimescaleDB merupakan ekstensi dari PostgreSQL yang dirancang secara khusus untuk otomasi manajemen skala *time-series* data. Berbeda dengan data historis secara umum yang rentang waktu pencatatannya bisa terjadi secara acak, data deret waktu

adalah kumpulan rekaman nilai kuantitatif yang diamati, diukur, dan diurutkan secara ketat berdasarkan interval waktu yang sangat teratur dan terus-menerus (Jensen dkk., 2017). Penggunaan TimescaleDB ini menjadi solusi untuk menangani kompleksitas overhead yang muncul jika database administrator harus membuat dan mengelola partisi rentang waktu secara manual pada sistem data warehouse.

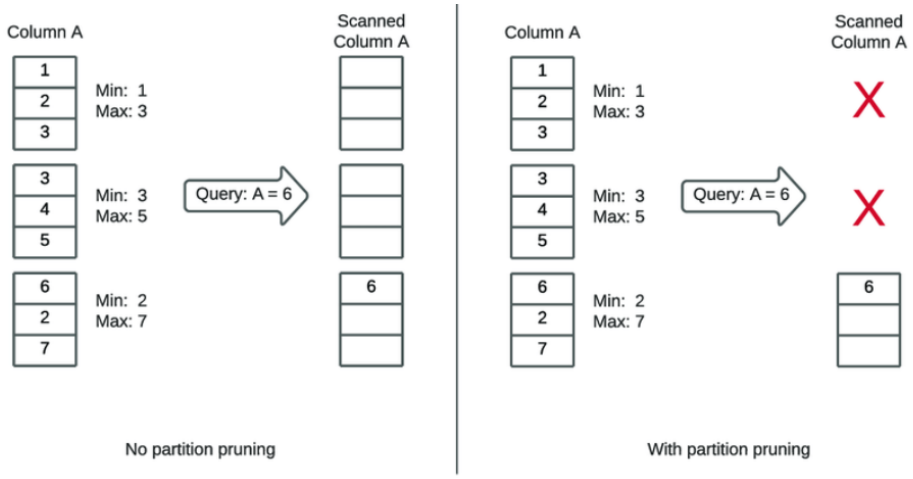
TimescaleDB mengimplementasikan arsitektur manajemen datanya melalui beberapa fitur utama, yaitu hypertable, chunk, *chunk time interval*, *chunk pruning*, dan kebijakan kompresi (*compression policy*). Dalam arsitektur ini, hypertable bertindak sebagai tabel logis (*logical table*) yang merepresentasikan tabel fakta (*fact table*) dari sebuah data warehouse. Melalui hypertable ini, seorang *user* dapat berinteraksi dan menjalankan kueri SQL seolah-olah sedang mengakses satu tabel yang tunggal. Namun, secara fisik, tabel logis tersebut terpartisi secara otomatis menjadi beberapa fragmen data yang disebut *chunk* (Timescale, 2023).



Gambar 2.1: Arsitektur partisi chunk pada hypertable TimescaleDB (Tiger Data)

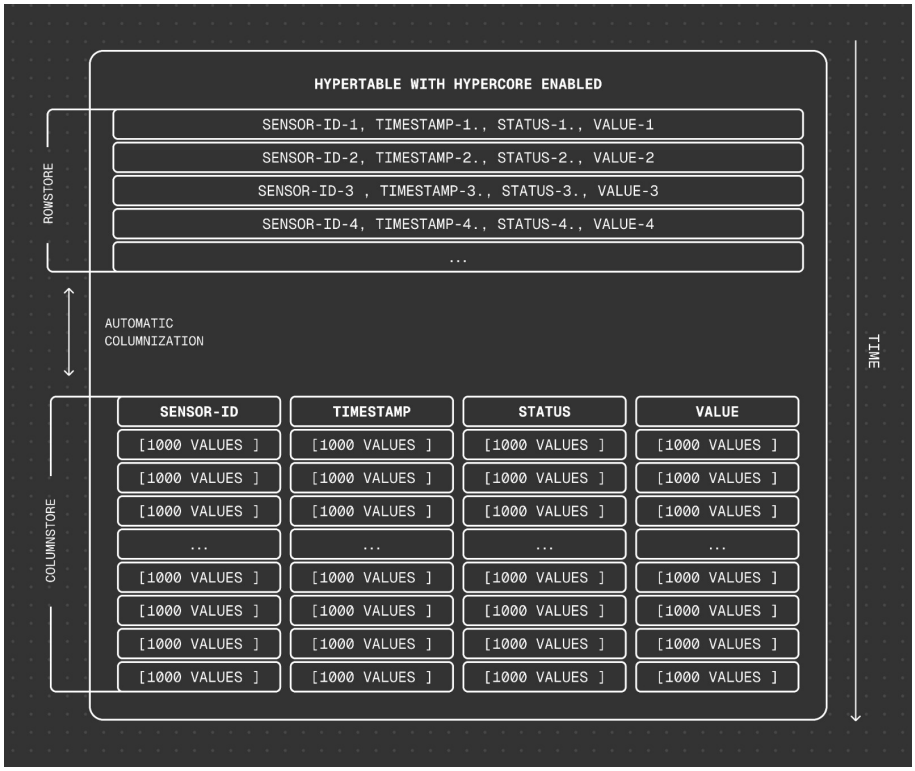
Chunk merupakan wujud partisi fisik dari hypertable yang dibuat oleh TimescaleDB secara otomatis. Proses pembentukan *chunk* secara dinamis dikendalikan oleh *chunk time interval*, yaitu sebuah parameter konfigurasi yang berfungsi untuk menentukan seberapa lebar rentang waktu data yang ditampung di dalam satu *chunk* fisik, sebagai

contoh dipartisi ke dalam interval harian, mingguan atau bulanan.(Timescale, 2023). Chunk pruning adalah mekanisme *pruning* partisi otomatis yang dapat dieksekusi oleh TimescaleDB sebagai salah satu keunggulan dari penggunaan *chunk*. Ketika sebuah kueri analitik dijalankan dengan filter rentang waktu tertentu, sistem akan langsung mengabaikan *chunk* yang tidak masuk dalam rentang waktu tersebut dari proses pembacaan disk, dan memastikan bahwa memori (RAM) yang digunakan hanya diisi oleh fragmen data yang relevan dengan kueri sehingga kinerja kueri dapat meningkat(Timescale, 2023).



Gambar 2.2: Perbandingan eksekusi kueri tanpa dan dengan pruning (Löser, 2021)

Kebijakan kompresi (*compression policy*) merupakan optimasi pada TimescaleDB yang mengubah format penyimpanan baris menjadi format berorientasi kolom (*columnar*) untuk partisi yang dianggap sudah lampau. Chunk yang berisi data *time series* yang sudah lampau dan sudah jarang menjadi target kueri dapat dikompresi sehingga ukurannya menyusut, pada akhirnya dapat menekan biaya I/O dan mempercepat waktu respons kueri saat partisi lampau tersebut dipindai kembali (Pelkonen dkk., 2015; Timescale, 2023).



Gambar 2.3: Ilustrasi hypercore selaku kebijakan kompresi pada TimescaleDB (Tiger Data)

Meskipun mekanisme otomasi partisi dan *pruning* ini menawarkan efisiensi tinggi, arsitektur bawaan TimescaleDB memiliki satu batasan mendasar di mana ukuran *chunk time interval* umumnya dikonfigurasi secara statis pada saat awal pembuatan Data Warehouse. Konfigurasi interval yang statis ini dibuat dengan asumsi bahwa pola akses pengguna akan selalu konstan. Namun kenyataannya, dalam lingkungan analitik yang dinamis, perilaku dan pola akses kueri seorang *user* terhadap rentang waktu tertentu dapat berubah-ubah. Pergeseran beban kerja (*workload*) ini menuntut sistem untuk mampu melakukan penyesuaian adaptif agar ukuran *chunk* yang statis tersebut tidak membebani memori ketika terjadi perubahan pola kueri.

2.5 Workload Kueri

Workload dalam lingkungan data warehouse merupakan kumpulan kueri yang dieksekusi dalam periode tertentu. Berbeda dengan basis data transaksional (OLTP) yang mengakses baris tunggal, kueri analitik (OLAP) dapat memindai dan mengagregasi data pada rentang waktu yang lebih panjang. Karakteristik kueri, rentang waktu, atribut yang dibaca, dan bentuk agregasi membentuk pola akses kueri (*query access pattern*).

Satu periode pengamatan terhadap sekumpulan kueri disebut *window*. Setiap *window* memuat komposisi kueri tertentu dan menghasilkan statistik eksekusi yang dibaca sebagai satu keadaan *workload*. Sebagai contoh, *window* pelaporan bulanan

dapat lebih banyak memuat kueri agregasi pada data terbaru, sedangkan *window* analisis historis dapat memuat kueri dengan rentang waktu yang lebih panjang.

Perubahan komposisi kueri antar-*window* disebut *workload shift*. Kondisi ini dapat terjadi ketika kebutuhan analitik berpindah dari kueri yang sudah dikenal (*predefined query*) menuju kueri baru (*ad-hoc query*). Pada data warehouse dengan interval partisi statis, perubahan rentang waktu dan bentuk agregasi dapat mengubah jumlah *chunk* yang dibaca, volume blok yang diproses, dan waktu eksekusi kueri.

Dalam mengevaluasi dampak dari *workload shift* terhadap kinerja data warehouse dengan memperhatikan penggunaan memory (*Memory-Aware*), diperlukan evaluasi yang menyeluruh. Pengujian ini diukur menggunakan metrik penilaian konfigurasi yang mencakup latensi p50 dan p95, efektivitas *pruning*, tekanan memori dan I/O *Cost*, hingga overhead adaptasi dari sistem.

2.6 Metrik Konfigurasi

Pada data warehouse dengan sumber daya memori RAM yang terbatas, konfigurasi *chunk time interval* dan kebijakan kompresi selalu melibatkan pertukaran (*trade-off*) antara performa kueri, efisiensi ruang yang digunakan, serta stabilitas sistem. Demi mengukur sejauh mana mekanisme partisi *memory-aware* berbasis PROADAPT mampu mengatasi dampak *workload shift*, diperlukan evaluasi. Penilaian pada penelitian ini menggunakan lima metrik utama yang merepresentasikan keseimbangan antara performa kinerja kueri dari sudut pandang *user* dan efisiensi sumber daya memori ram dari sudut pandang mesin. Setiap metrik dijelaskan lebih rinci sebagai berikut.

1. **Latensi kueri**

   Latensi kueri merupakan waktu yang dibutuhkan oleh database (TimescaleDB) untuk memproses kueri hingga mendapatkan hasil. Metrik ini digunakan untuk mengukur Pengukuran latensi dilakukan menggunakan persentil ke-50 (P50) yang mewakili hasil 50% dari total kueri, dan persentil ke-95 (P95) yang menunjukkan kecepatan kueri pada kondisi terburuknya (Beyer dkk., 2016).

2. **Efektivitas Chunk Pruning**

   Efektivitas *chunk pruning* merupakan metrik yang mengukur perbandingan jumlah *chunk* yang dibaca saat melakukan kueri dengan total *chunk* yang ada pada data warehouse. Sistem melakukan pemangkasan ini dengan dua hal, mengabaikan *chunk* yang tidak relevan dan hanya membaca kolom yang diminta oleh kueri (data yang dikompresi). Semakin banyak tumpukan data tidak relevan yang berhasil diabaikan (*skipped*), semakin sedikit pula data yang perlu diproses di memori (RAM), sehingga eksekusi kueri menjadi jauh lebih cepat.

3. **Storage Size**

Storage size merupakan total ukuran penyimpanan dari data warehouse didalam penyimpanan internal server. Metrik *storage size* mengevaluasi seberapa banyak ruang yang dihemat setelah data terkompresi. Meskipun penyusutan *storage* ini sangat menghemat penyimpanan server, sistem harus berhati-hati agar kompresi yang agresif malah membebani kerja prosesor (CPU) ketika pengguna tiba-tiba membutuhkan data historis yang sangat detail.

4. **Pengukuran *Memory-Aware***

   Pengukuran *memory-aware* menggunakan *Buffer Miss Ratio* (BMR) dan *Working Set Size* (WSS). BMR merupakan persentase seberapa sering PostgreSQL membaca data dari *storage disk* dikarenakan data yang diminta tidak ditemukan pada RAM (`shared_buffers`), sedangkan WSS menunjukkan estimasi ukuran blok data yang dibaca selama kueri dijalankan. Kedua indikator tersebut dihitung dari statistik blok PostgreSQL dan dijelaskan pada Subbab 2.8.2.

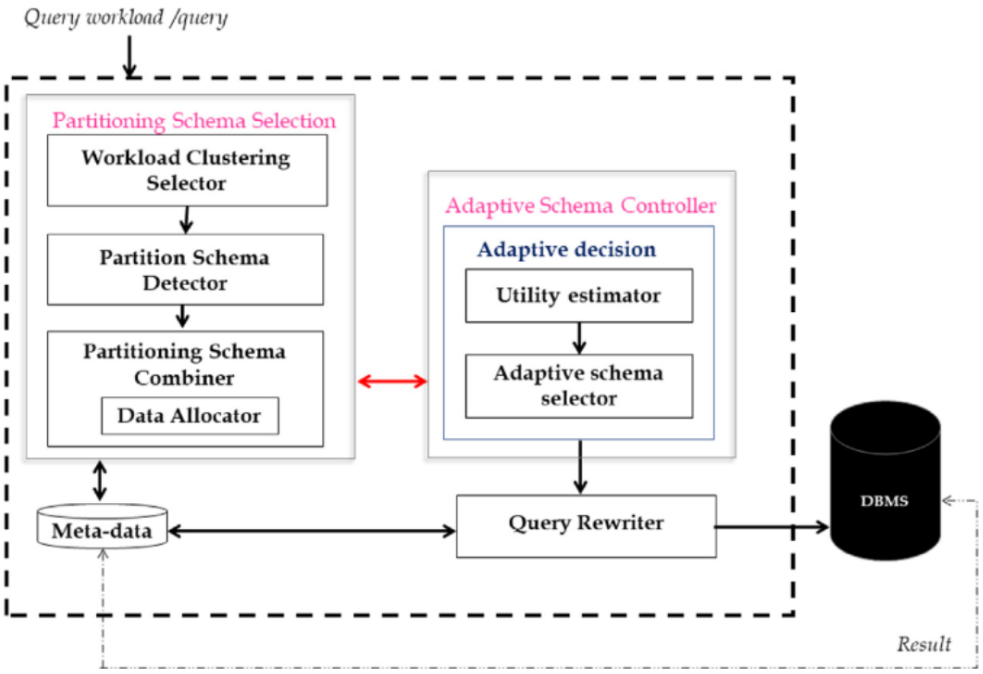
5. **Overhead Adaptasi**

   Overhead Adaptasi merupakan waktu atau biaya tambahan yang digunakan oleh sistem selama melakukan proses adaptasi atau penyesuaian baik tahap *offline* maupun tahap *online* dari mekanisme PROADAPT.

2.7 PROADAPT

PROADAPT (*Proactive Framework for Adaptive Partitioning*) merupakan kerangka partisi adaptif yang dirancang untuk mempertahankan kegunaan skema partisi ketika *workload data warehouse* berubah. Kerangka ini membentuk skema partisi awal berdasarkan riwayat kueri, memantau kueri baru selama sistem berjalan, dan melakukan penulisan ulang kueri agar bagian data yang tidak diperlukan dapat diabaikan (Benkrid dkk., 2022).

PROADAPT original diterapkan pada lingkungan paralel dan dengan sistem basis data terdistribusi dengan bentuk partisi multidimensi. Prinsip yang diimplementasikan pada penelitian ini meliputi seleksi konfigurasi berdasarkan *workload*, penilaian berbasis utilitas, pemantauan pada tahap *online*, dan penulisan ulang kueri. Prinsip tersebut selanjutnya disesuaikan dengan karakteristik TimescaleDB dalam lingkungan *single-VM*.



Gambar 2.4: Diagram Arsitektur dan Alur Kerja Mekanisme PROADAPT (Benkrid dkk., 2022)

Secara teknis, mekanisme kerja kerangka PROADAPT dibagi menjadi dua tahapan utama, *Partitioning Schema Selection* atau selanjutnya disebut tahap *offline* dan *Adaptive schema selector* atau selanjutnya disebut tahap *online*. Kedua tahap tersebut akan dijelaskan dibawah ini.

2.7.1 Tahap Offline

Pada tahap *offline*, PROADAPT menganalisis riwayat kueri yang telah dikumpulkan yang biasa disebut *workload query log*. Kemudian *workload clustering* yaitu Kueri dengan karakteristik yang serupa dikelompokkan, selanjutnya setiap kelompok digunakan untuk membentuk kandidat skema partisi (*partition candidate detector*). Kandidat tersebut dinilai dan digabungkan untuk memperoleh skema awal yang tetap berguna bagi kueri yang telah dikenal maupun kueri baru (Benkrid dkk., 2022).

2.7.2 Tahap Online

Setelah skema awal terbentuk, tahap *online* memantau kueri baru dan perubahan *workload* yang bisa disebut *workload monitoring*. Kueri baru terlebih dahulu dibandingkan dengan kelompok kueri yang telah terbentuk. Selama skema aktif masih memberikan manfaat atau utilitas yang memadai, kueri diteruskan ke *query rewriter*. Apabila manfaat skema menurun melewati batas yang ditetapkan, PROADAPT menjalankan kembali proses pemilihan skema dengan memanfaatkan hasil tahap *offline* (Benkrid dkk., 2022).

2.8 Fungsi Utilitas

PROADAPT menggunakan konsep manfaat atau *benefit* untuk mengukur keuntungan yang diberikan oleh suatu skema partisi terhadap eksekusi kueri. Keuntungan tersebut dihitung melalui selisih biaya kueri tanpa skema partisi dan biaya kueri ketika skema partisi digunakan (Benkrid dkk., 2022).

$$B_{q,SF} = cost(q) - cost(q \mid SF) \tag{2.3}$$

Pada Persamaan 2.3, $cost(q)$ merupakan biaya eksekusi kueri tanpa menggunakan skema partisi SF, sedangkan $cost(q \mid SF)$ merupakan biaya eksekusi kueri setelah skema tersebut digunakan. Nilai *benefit* yang lebih besar menunjukkan bahwa skema partisi mampu mengurangi biaya eksekusi kueri dengan lebih baik.

Nilai *benefit* kemudian digunakan oleh PROADAPT sebagai dasar untuk menilai apakah skema partisi yang sedang digunakan masih cukup sesuai ketika kueri baru dieksekusi. Skema yang masih memberikan manfaat yang memadai dapat dipertahankan sehingga perubahan skema pada data tidak perlu dilakukan terlalu sering.

Penelitian ini mengambil prinsip penilaian berbasis utilitas tersebut. Fungsi utilitas pada penelitian ini dikembangkan sebagai penilaian yang disesuaikan dengan TimescaleDB *single-VM*. Formulasi yang digunakan disajikan pada Bab IV.

2.8.1 Kinerja Kueri

Kinerja kueri bertindak sebagai variabel keuntungan (*reward*) di dalam fungsi utilitas. Kinerja ini dievaluasi berdasarkan estimasi penurunan latensi eksekusi kueri. Konfigurasi yang mampu menghasilkan efektivitas *pruning* tinggi dan meminimalkan operasi dekompresi akan menghasilkan latensi yang relatif rendah, sehingga memberikan nilai *Benefit* yang relatif tinggi. Waktu respons yang lebih rendah memberi kontribusi lebih besar pada perhitungan utilitas akhir.

2.8.2 Memory Cost (Memory-Aware)

Pada fungsi utilitas ini, kinerja kueri bertindak sebagai keuntungan (*benefit*), sedangkan penggunaan memori menjadi bagian dari biaya (*cost*). *Buffer* merupakan area RAM yang menyimpan blok data agar database tidak selalu membaca kembali data yang sama dari *storage*. Selinger dkk. menunjukkan bahwa pemilihan *access path* mempertimbangkan biaya pembacaan halaman data dalam penyusunan rencana eksekusi kueri (Selinger, Astrahan, Chamberlin, Lorie, dan Price, 1979). Chou dan DeWitt membahas pengelolaan *buffer pool* berdasarkan kebutuhan halaman aktif dari operasi basis data (Chou dan DeWitt, 1985).

PostgreSQL mencatat jumlah blok yang ditemukan di *shared buffer* melalui

`shared_blks_hit` dan jumlah blok yang dibaca melalui `shared_blks_read` (PostgreSQL Global Development Group, 2026a). Berdasarkan kedua statistik tersebut, *Buffer Miss Ratio* (BMR) menyatakan bagian akses blok yang memerlukan pembacaan dari *storage*. Nilai BMR dituliskan pada Persamaan berikut.

$$BMR = \frac{\texttt{shared\_blks\_read}}{\texttt{shared\_blks\_hit} + \texttt{shared\_blks\_read}} \tag{2.4}$$

Pada Persamaan 2.4, nilai yang mendekati 0 menunjukkan bahwa sebagian besar akses blok dapat dilayani dari *shared buffer*. Nilai yang lebih besar menunjukkan bahwa bagian pembacaan dari *storage* juga lebih besar.

Volume blok yang diakses selama kueri dijalankan dinyatakan sebagai *Working Set Size* (WSS). Pada penelitian ini, WSS merupakan estimasi operasional yang dihitung dari jumlah blok yang ditemukan di *shared buffer* dan blok yang dibaca dari *storage*. Ukuran blok PostgreSQL pada lingkungan penelitian adalah 8192 byte (PostgreSQL Global Development Group, 2026b). Nilai WSS dituliskan pada Persamaan berikut.

$$WSS = (\texttt{shared\_blks\_hit} + \texttt{shared\_blks\_read}) \times \textit{Block Size} \tag{2.5}$$

Pada Persamaan 2.5, WSS menunjukkan volume akses blok dalam satuan byte dan selanjutnya dikonversi menjadi megabyte. Nilai tersebut digunakan sebagai proksi luas data aktif yang diproses oleh kueri, bukan sebagai pengukuran langsung seluruh RAM yang dialokasikan oleh proses PostgreSQL.

Meskipun secara teoretis PROADAPT mengkalkulasi fungsi utilitas berdasarkan rasio keuntungan komputasi *cost*, penerapan perhitungan membutuhkan penyesuaian untuk mengakomodasi kapasitas memori (RAM). Fungsi utilitas harus memberikan penalti apabila ukuran tekanan memori (RAM) dan I/O *cost* dari konfigurasi tersebut melampaui batas kapasitas. Penalti ini merupakan mekanisme agar sistem terhindar dari lonjakan I/O akibat proses pertukaran data (*swapping*).

2.8.3 Waktu Adaptasi

Parameter terakhir yang menjadi faktor penentu adalah waktu adaptasi atau *overhead* adaptasi. Sebuah kandidat konfigurasi mungkin menawarkan latensi rendah dengan penggunaan memori yang aman, namun jika waktu untuk adaptasi yang terlalu lama, nilai utilitasnya akan berkurang. Sistem hanya akan diizinkan untuk mengubah konfigurasi ukuran *chunk* apabila nilai utilitas kueri tersebut turun di bawah ambang batas toleransi minimum (*utility threshold / u*) yang telah ditetapkan.

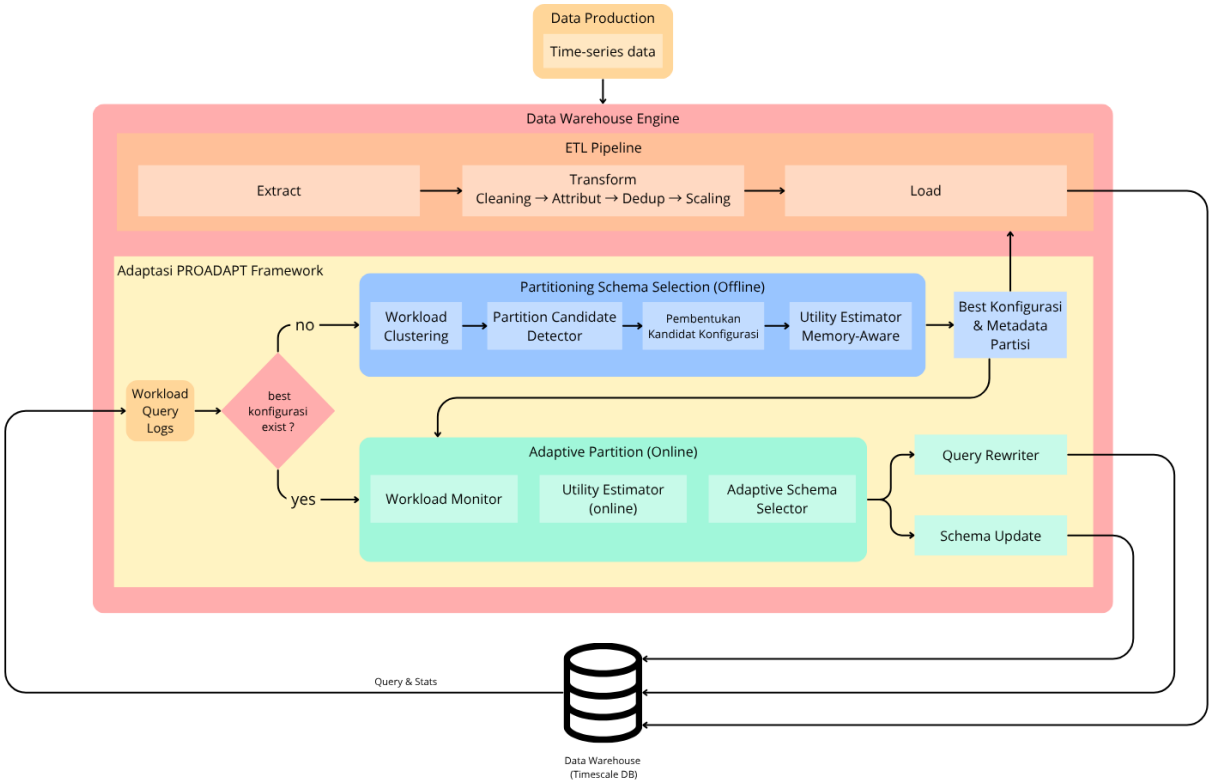
Meskipun secara teoretis *framework* PROADAPT menghitung fungsi utilitas berdasarkan rasio keuntungan biaya komputasi, penerapan model evaluasi ini di lingkungan data warehouse memiliki kasus yang berbeda-beda, sehingga membutuhkan penyesuaian. Oleh karena itu, pada penelitian ini, fungsi utilitas tersebut diadaptasi dan

disesuaikan lebih lanjut. Penyesuaian ini dilakukan untuk mengalokasikan kapasitas memori (RAM) secara eksplisit, serta mengintegrasikan hasil dari algoritma pengelompokan kueri (*workload clustering*) agar evaluasi kandidat konfigurasi *chunk time interval* dan kebijakan kompresi menjadi lebih relevan dengan sumber daya (memori) yang terbatas.

BAB III
METODOLOGI PENELITIAN

Pada bab ini dijelaskan tahapan penelitian untuk meningkatkan kinerja kueri *data warehouse* menggunakan mekanisme partisi *memory-aware* berbasis PROADAPT pada PostgreSQL/TimescaleDB. Metodologi penelitian mencakup tahap *offline* untuk memilih konfigurasi awal terbaik dari kandidat yang tersedia, serta tahap *online* untuk memantau perubahan pola kueri dan menjalankan adaptasi minor secara terukur.

Alur penelitian ditunjukkan pada 3.1. Proses dimulai dari *data production* dan *ETL pipeline*, dilanjutkan dengan penerapan PROADAPT pada *data warehouse engine* hingga data tersimpan pada TimescaleDB.



Gambar 3.1: Blok Diagram Penelitian

Blok *Partitioning Schema Selection (Offline)* pada 3.1 menunjukkan bahwa pemilihan konfigurasi dilakukan pada tahap *offline*. Hasil dari tahap ini menjadi dasar untuk tahap *online*. Setelah konfigurasi awal terpilih, tahap *online* memantau perubahan pola kueri dan menjalankan adaptasi minor agar stabilitas sistem tetap terjaga dan *overhead* adaptasi tetap terkendali.

3.1 Studi Literatur

Studi literatur dilakukan untuk membangun landasan konseptual penelitian terkait *data warehouse*, konsep partisi dan *pruning*, karakteristik TimescaleDB, serta prinsip dasar PROADAPT berbasis *workload*. Kajian ini digunakan untuk menyusun rancangan metodologi, menetapkan batasan tahap *offline* dan *online*, menyusun kandidat konfigurasi yang realistis untuk diimplementasikan, serta merumuskan metrik evaluasi yang relevan dengan tujuan peningkatan kinerja kueri dan aspek *memory-aware*.

3.2 Pengumpulan Data dan Spesifikasi Dataset

Data yang digunakan merupakan data sekunder publik dari *Intel Berkeley Research Lab* (IBRL). Pada penelitian ini, dataset tersebut diposisikan sebagai sumber data historis detail yang kemudian dibentuk menjadi data acuan DW. Karakteristik berorientasi waktu pada dataset ini relevan untuk mengevaluasi keputusan partisi, granularity baca, dan kebijakan kompresi pada data historis. Spesifikasi dataset ditunjukkan pada Tabel 3.1.

Tabel 3.1: Karakteristik data sumber penelitian

Parameter	**Keterangan**
Sumber data	Intel Berkeley Research Lab
Entitas pengamatan	54 titik pengamatan
Atribut utama	`date`, `time`, `epoch`, `moteid`, `temp`, `humidity`, `light`, `voltage`
Volume Asli	$\approx$ 2.3 Juta baris
Rentang Waktu	28 Februari 2004 – 5 April 2004
Frekuensi	Sampling setiap $\pm$ 31 detik

Untuk mendukung pengujian pada skala *data warehouse* ketika ukuran data melebihi kapasitas memori, penelitian ini menerapkan skenario *Data Scaling*. Data asli diduplikasi secara sintetis dengan teknik *timestamp-shifting* hingga mencapai volume target 50 juta baris agar kondisi *memory pressure* dapat diamati pada lingkungan uji yang tersedia.

3.3 Pra-Pemrosesan Data (ETL Pipeline)

Pra-pemrosesan data dilakukan melalui *ETL pipeline* untuk mengubah data mentah menjadi data terstruktur dan siap dianalisis pada *data warehouse*. Pada Bab III, tahapan ini dijelaskan pada tingkat metodologi sebagai alur pengolahan data yang menjaga keseragaman antara skenario baseline dan skenario PROADAPT. Realisasi program, rekaman ETL, dan bentuk data acuan yang benar-benar dipakai dijelaskan lebih rinci pada Bab IV.

3.3.1 *Extract*

Tahap *extract* berfokus pada pengambilan data mentah dari sumber dataset dan penyiapan format input yang konsisten untuk diproses pada tahap berikutnya. Pada tahap ini ditetapkan atribut utama yang akan dipertahankan, terutama atribut waktu, identitas entitas, dan nilai pengamatan, sehingga data siap diproses secara terstandar pada tahap transformasi.

3.3.2 *Transform*

Tahap *transform* mencakup pembersihan data, penyeragaman format waktu, penanganan nilai hilang dan duplikasi seperlunya, serta pembentukan struktur data yang mendukung analitik. Transformasi diarahkan untuk mendukung pola kueri agregasi dan pembacaan historis pada variasi rentang waktu, tanpa mengubah makna dasar data pada dataset sumber.

3.3.3 *Load*

Tahap *load* dilakukan dengan memuat hasil transformasi ke dalam skema *data warehouse* pada TimescaleDB, khususnya pada tabel fakta pengukuran yang kemudian dijadikan *hypertable*. Proses pemuatan data dirancang agar konsisten dengan skema fisik yang diuji dan mendukung pengukuran kinerja kueri pada tahap pengujian.

Validitas perbandingan antar skenario pengujian dijaga dengan menerapkan proses ETL yang identik pada skenario baseline dan skenario PROADAPT dari sisi sumber data, langkah transformasi, dan prosedur pemuatan. Dengan demikian, perbedaan hasil yang diamati dapat dikaitkan pada mekanisme partisi dan kompresi yang diterapkan.

3.4 Pembangunan Data Warehouse dan Lingkungan Eksperimen

Tahap pembangunan *data warehouse* diawali dengan penyiapan lingkungan komputasi yang terisolasi untuk memastikan pengukuran kinerja yang valid. Pada Bab III, bagian ini diposisikan sebagai rancangan metodologis mengenai lingkungan eksperimen dan batas sumber daya yang digunakan. Detail implementasi fisik, pembagian peran mesin pengendali dan target *warehouse*, serta verifikasi state server dijelaskan pada Bab IV. Spesifikasi lingkungan eksperimen dirinci pada Tabel 3.2.

Tabel 3.2: Spesifikasi Lingkungan Komputasi Eksperimen

Komponen	**Spesifikasi**
Platform Virtualisasi	OpenNebula (KVM)
Orkestrasi	Kubernetes
Alokasi CPU	4 vCPU
Alokasi RAM	**4 GB**
Sistem Operasi	Linux (Ubuntu Server 22.04 LTS)
Database Engine	PostgreSQL + TimescaleDB
Penyimpanan	SSD Block Storage

Pada lingkungan ini, tabel fakta historis dirancang untuk direalisasikan sebagai *hypertable*. Konfigurasi dasar PostgreSQL seperti `shared_buffers` dan `work_mem` diselaraskan dengan batasan RAM 4 GB agar tekanan memori dan I/O dapat diamati ketika beban kerja meningkat, sehingga efektivitas mekanisme PROADAPT dapat dievaluasi pada konteks *single-VM*.

3.5 Penerapan PROADAPT

Pada tahap ini, mekanisme PROADAPT diterapkan pada lingkungan *data warehouse* yang telah dibangun. Uraian pada Bab III menekankan alur metodologis dan batas ruang baca penelitian, sedangkan rancangan sistem dan implementasi skrip yang benar-benar dijalankan dipaparkan pada Bab IV.

3.5.1 Pengumpulan *Workload Query Log*

Setiap eksekusi kueri pada *data warehouse* direkam sebagai *workload query log*. Log memuat informasi kueri dan ringkasan metrik eksekusi yang digunakan sebagai masukan analisis pada tahap *offline* maupun *online*, termasuk informasi yang relevan untuk menilai latensi, efektivitas *chunk pruning*, serta indikator perilaku buffer dan I/O.

3.5.2 *Partitioning Schema Selection (Offline)*

Tahap *offline* pada penelitian ini merupakan tahap seleksi konfigurasi awal untuk mencari kandidat dengan nilai utilitas terbaik. Pada tahap ini, sejumlah kandidat dijalankan lebih dahulu, kemudian seluruh rekapan hasil pengukuran dikumpulkan dan dinilai dengan fungsi utilitas yang mempertimbangkan kinerja kueri dan aspek *memory-aware*. Hasil akhirnya adalah satu konfigurasi dengan utilitas terbaik yang dipilih sebagai acuan untuk tahap berikutnya. Dengan demikian, tahap *offline* menjadi dasar pemilihan konfigurasi awal yang akan dipakai pada implementasi tahap *online*. Rincian ruang kandidat, mekanisme seleksi, *Genetic Algorithm*, dan rekapan hasilnya dijelaskan pada Bab IV.

3.5.3 *Adaptive Partition (Online)*

Tahap *online* dijalankan setelah konfigurasi awal hasil tahap *offline* ditetapkan. Setiap *window* menjalankan sekumpulan kueri dengan komposisi tertentu. Sistem mencatat metrik eksekusi, mengestimasi utilitas, lalu melakukan adaptasi minor apabila diperlukan. Persentase *predefined query* diturunkan dari 100% menjadi 0% dengan perubahan 25% pada setiap *window*. Persentase sisanya merupakan *ad-hoc query*. Mekanisme *cooldown* mengatur jarak antaraksi, sedangkan *hysteresis* menahan perubahan ketika manfaat adaptasi belum mencapai batas yang ditetapkan. Implementasinya dipaparkan pada Bab IV dan hasilnya disajikan pada Bab V.

3.5.4 *Query Rewriting* dan Penyesuaian Kebijakan Kompresi

Penyesuaian pada tahap *online* difokuskan pada *query rewriter* dan penyesuaian kebijakan kompresi. *Query rewriter* diarahkan untuk memperjelas predikat waktu atau predikat baca utama sehingga *chunk pruning* lebih efektif dan akses data historis lebih terarah. Penyesuaian kebijakan kompresi dilakukan untuk mengendalikan ukuran data dan menekan jejak memori secara tidak langsung melalui perilaku buffer dan I/O. Jika adaptasi minor belum cukup menjaga utilitas tetap dekat dengan skenario *offline* terpilih, maka sistem mengambil keputusan untuk kembali ke tahap *offline*. Realisasi aturan *rewrite* dan perubahan pada kebijakan kompresi dipaparkan lebih rinci pada Bab IV.

BAB IV
PERANCANGAN DAN IMPLEMENTASI

Bab ini menjelaskan bagaimana penelitian ini direalisasikan mulai dari persiapan data sampai pengujian. Uraian pada bab ini mengikuti alur metodologi penelitian. Fokus pembahasan diarahkan pada pembentukan *data warehouse*, pemilihan konfigurasi awal pada tahap *offline*, dan pelaksanaan adaptasi minor pada tahap *online*.

4.1 Persiapan Data

Persiapan data dilakukan agar seluruh eksperimen menggunakan sumber data dan *workload* uji yang sama. Pada tahap ini, *raw data* dibaca dari file sumber, dipisahkan ke bentuk kolom, dirapikan, lalu dimuat ke tabel fakta yang digunakan pada penelitian ini.

4.1.1 Data Production

Data yang digunakan berasal dari *Intel Berkeley Research Lab* yang merupakan sumber data utama. Data ini berupa file `.txt` yang menyimpan catatan pengamatan sensor berdasarkan waktu dan id sensor. Setiap baris data berisi date, time, *epoch*, moteid, temperature, humidity, light, dan voltage.

Berikut contoh Lima baris awal data ditunjukkan pada Listing berikut 4.1.

Listing 4.1: lima baris awal data production

```
2004-03-31 03:38:15.757551 2 1 122.153 -3.91901 11.04 2.03397
2004-02-28 00:59:16.02785 3 1 19.9884 37.0933 45.08 2.69964
2004-02-28 01:03:16.33393 11 1 19.3024 38.4629 45.08 2.68742
2004-02-28 01:06:16.013453 17 1 19.1652 38.8039 45.08 2.68742
2004-02-28 01:06:46.778088 18 1 19.175 38.8379 45.08 2.69964
```

raw data pada Listing 4.1 masih berupa format teks (flat file) yang tidak memiliki batasan kolom maupun struktur tipe data relasional. Data tersebut akan melalui tahap transformasi untuk pembersihan data anomali, scaling data serta menyesuaikan tipe datanya. Setelah proses transformasi selesai, data tersebut dimuat (load) ke dalam tabel fakta.

4.1.2 Proses Ekstraksi Data

Ekstraksi dilakukan dengan mengambil informasi dari setiap baris *raw data*, lalu memisahkan setiap kolomnya menjadi *temporary table*. Setiap kolom disesuaikan dengan

tipe data.

hasil ekstraksi lima baris awal ditunjukkan pada Tabel 4.1. Tabel ini dapat dibaca sebagai *output* dari proses ekstraksi.

Tabel 4.1: hasil ekstraksi lima baris awal

Date	Time	Epoch	Moteid	Temp	Humidity	Light	Voltage
2004-03-31	03:38:15.757551	2	1	122.1530	-3.91901	11.04	2.03397
2004-02-28	00:59:16.027850	3	1	19.9884	37.09330	45.08	2.69964
2004-02-28	01:03:16.333930	11	1	19.3024	38.46290	45.08	2.68742
2004-02-28	01:06:16.013453	17	1	19.1652	38.80390	45.08	2.68742
2004-02-28	01:06:46.778088	18	1	19.1750	38.83790	45.08	2.69964

Ringkasan data hasil ekstraksi sebelum pembersihan lanjutan ditunjukkan pada Tabel 4.2.

Tabel 4.2: Ringkasan data hasil ekstraksi

Komponen	**Nilai**
Jumlah baris	2.313.682 baris
Waktu awal data	2004-02-28 00:58:46.002832
Waktu akhir data	2004-04-05 11:02:32.715337

Hasil ekstraksi memuat 2.313.682 baris data dengan waktu pengamatan mulai 2004-02-28 sampai 2004-04-05. Data tersebut kemudian dibersihkan dan disesuaikan formatnya sebelum dimuat ke dalam tabel fakta.

4.1.3 Proses Transformasi Data

Tahap transformasi merupakan proses untuk pembersihan dan standardisasi data sebelum dimasukkan ke dalam tabel fakta pada data warehouse. Pada tahap ini, data yang masih berada dalam *temporary table* diproses melalui empat tahap, yaitu pembersihan (*cleaning*), penyesuaian atribut (*attribut*), penghapusan duplikasi (*dedup*), dan data (*scaling*).

Proses *cleaning* dan *dedup* bertujuan untuk menyaring baris data yang memiliki nilai anomali, kosong (*null*), atau duplikat akibat *error* yang terjadi pada sensor *wireless*. *Raw Data* dipindai, lalu baris yang terduplikasi pada waktu dan *moteid* yang sama persis dihapus. Ringkasan jumlah baris akibat proses *cleaning* dan *dedup* ditunjukkan pada Tabel 4.3.

Tabel 4.3: Perbandingan jumlah data sebelum dan sesudah pembersihan

Komponen	**Jumlah Baris Data**
Sebelum pembersihan (*raw data*)	2.313.682 baris
Sesudah *cleaning* dan *dedup*	2.219.803 baris
Total data dihapus	**93.879 baris**

Pada saat data *raw data* diekstraksi ke dalam *temporary table*, struktur tabel sementara ditunjukkan pada Tabel 4.4.

Tabel 4.4: Struktur kolom *temporary table*

Nama Kolom Mentah	**Tipe Data Awal**
time	varchar
date	varchar
epoch	integer
moteid	integer
temp	numeric
humidity	numeric
light	numeric
voltage	numeric

Berdasarkan struktur awal pada Tabel 4.4, informasi waktu terpisah ke dalam dua kolom berbeda (`tanggal` dan `waktu`). Bertujuan untuk kompatibel dengan arsitektur (*time-series*) pada TimescaleDB, kedua atribut tersebut dilebur (*concatenation*) menjadi satu kolom bernama `time` dan dikonversi menjadi tipe data `timestamp`. Selain itu, dilakukan standardisasi penamaan atribut `temp` menjadi `temperature`. Perbandingan struktur sebelum dan sesudah proses penyesuaian atribut ditunjukkan pada Tabel 4.5.

Tabel 4.5: Contoh data sebelum dan sesudah transformasi atribut

Sebelum Transformasi	**Sesudah Transformasi**
tanggal=2004-02-28 waktu=00:59:16.027850 epoch=3 moteid=1 temp=19.9884 humidity=37.0933 light=45.08 voltage=2.69964	time=2004-02-28 00:59:16.027850 epoch=3 moteid=1 temperature=19.9884 humidity=37.0933 light=45.08 voltage=2.69964

Transformasi pada tahap ini dieksekusi menggunakan sintaks kueri SQL yang diringkas pada Listing 4.2.

Listing 4.2: Contoh transformasi atribut waktu dan nilai pengamatan

```
SELECT
    (tanggal || '␣' || waktu)::timestamp AS time,
    epoch::integer AS epoch,
    moteid::integer AS moteid,
    temp::double precision AS temperature,
    humidity::double precision AS humidity,
    light::double precision AS light,
    voltage::double precision AS voltage
FROM data_hasil_ekstraksi_temporary;
```

Kueri pada Listing 4.2 mengeksekusi penggabungan dan melakukan konversi (*casting*) tipe data. Setelah struktur data siap, langkah terakhir pada tahap transformasi adalah menerapkan *data scaling*.

Pada penelitian ini, data warehouse berfokus pada mekanisme *memory-aware* di bawah *workload* yang berat, volume data sekitar 2,4 juta baris dianggap belum cukup memicu lonjakan I/O dan membebani kapasitas RAM pada server pengujian. Oleh karena itu, diterapkan pembesaran volume atau disebut dengan *scaling* data. *Scaling* data merupakan duplikasi baris data secara sintetis menggunakan teknik (*timestamp-shifting*). Data asli duplikat secara berulang dan digeser ke periode tahun berikutnya sebagai contoh data pengukuran sensor dengan *time* 2004-02-28 00:59:16.027850 dengan kolom-kolom yang ada diduplikasi dengan *time* yang digeser ke 2005-02-28 00:59:16.027850 sehingga membentuk data historis baru yang lebih banyak dan dapat digunakan untuk pengujian.

Contoh pergeseran waktu pada satu baris data sebelum dan sesudah proses *scaling* ditunjukkan pada Tabel 4.6.

Tabel 4.6: Contoh bentuk pergeseran waktu pada proses *data scaling*

Tahap	**Contoh isi data**
Sebelum *data scaling*	`time=2004-02-28 00:59:16.027850, epoch=3, moteid=1, temperature=19.9884, humidity=37.0933, light=45.08, voltage=2.69964`
Sesudah *data scaling*	`time=2005-02-28 00:59:16.027850, epoch=3, moteid=1, temperature=19.9884, humidity=37.0933, light=45.08, voltage=2.69964`

Secara keseluruhan, proses *timestamp-shifting* pada Tabel 4.6 ini akan berdampak pada ukuran data tabel fakta yang akan diuji. Rincian lonjakan volume data dan rentang waktu akibat *scaling* dirangkum pada Tabel 4.7.

Tabel 4.7: Perubahan ukuran dan rentang waktu data sesudah *data scaling*

Komponen	**Sebelum *Scaling***	**Sesudah *Scaling***
Jumlah baris data	2.219.803 baris	44.396.060 baris
Waktu awal data	2004-02-28 00:58:46.002832	2004-02-28 00:58:46.002832
Waktu akhir data	2004-04-05 11:02:32.715337	2005-09-19 11:02:32.715337

Proses *data scaling* menghasilkan 44.396.060 baris dengan rentang waktu sampai 2005-09-19. Setelah tahap ini selesai, data dipindahkan dari *temporary table* ke tabel fakta pada *data warehouse*.

4.1.4 Proses Load Data

Setelah proses transformasi, data dimuat atau disebut (*load*) ke dalam tabel fakta bernama `sensor_readings`. Berbeda dengan *temporary table* yang bersifat sementara, tabel fakta diimplementasikan sebagai *hypertable* pada lingkungan data warehouse dengan TimescaleDB. Pembuatan struktur *hypertable* ini akan memastikan bahwa jutaan baris

data yang masuk dapat langsung dipartisi secara horizontal menjadi *chunk* berdasarkan atribut *time*.

Mengacu pada model partisi pada Persamaan 2.2, relasi R pada implementasi ini adalah tabel fakta `sensor_readings`, sedangkan setiap P_i direalisasikan sebagai satu *chunk*. Batas τ_i dan τ_{i+1} ditentukan oleh nilai *chunk time interval* pada konfigurasi yang digunakan.

Rincian struktur fisik beserta tipe data dari tabel fakta `sensor_readings` yang siap menampung *production data* tersebut ditunjukkan pada Tabel 4.8.

Tabel 4.8: Struktur fisik tabel fakta `sensor_readings`

Nama Kolom	**Tipe Data**
time	timestamp
epoch	integer
moteid	integer
temperature	double precision
humidity	double precision
light	double precision
voltage	double precision

Upaya memindahkan (*load*) baris data dari *temporary table* hasil transformasi ke dalam tabel fakta tersebut, sistem mengeksekusi sintaks *bulk insert* menggunakan kueri SQL pada Listing 4.3.

Listing 4.3: Contoh proses *load* data ke tabel fakta `sensor_readings`

```
INSERT INTO sensor_readings (
    time, epoch, moteid, temperature, humidity, light, voltage
)
SELECT
    time, epoch, moteid, temperature, humidity, light, voltage
FROM data_hasil_transformasi;
```

Sintaks pada Listing 4.3 memastikan bahwa data hasil transformasi dimuat ke dalam tabel fakta. Data Production menjadi bahan untuk membangun data warehouse pada tahap selanjutnya.

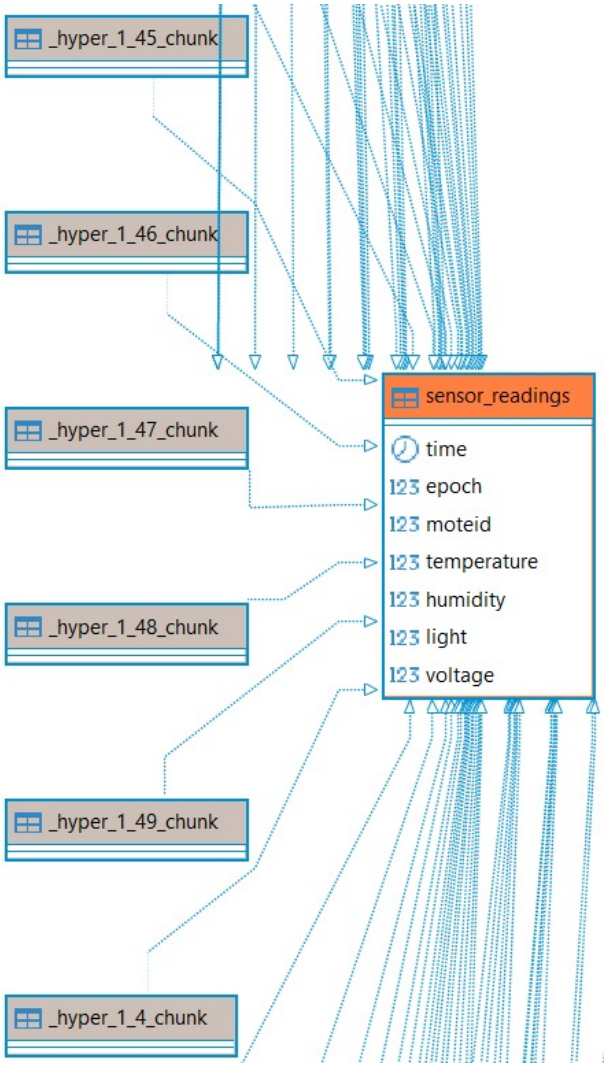
4.2 Pembangunan Data Warehouse

Data hasil *load* yang berada dalam tabel `sensor_readings` menjadi fondasi dalam pembangunan arsitektur *data warehouse*. Sebagaimana pada yang diuraikan pada Bab 2, akumulasi data historis berskala besar harus disimpan secara terpusat pada sebuah struktur yang dinamakan tabel fakta (*fact table*). Pada penelitian ini, tabel `sensor_readings` berperan sebagai tabel fakta, bertindak sebagai penyimpanan rekaman pengamatan dan menjadi objek utama bagi seluruh kueri analitik.

Dalam mengatasi tingginya biaya komputasi akibat besarnya volume data historis, tabel fakta tersebut dirancangkan arsitektur TimescaleDB dengan mengubah tabel standar `sensor_readings` menjadi *hypertable*.

Pada arsitektur ini, *hypertable* bertindak sebagai tabel logis (*logical table*) yang merepresentasikan tabel fakta secara utuh namun secara realitasnya tabel dipartisi secara otomatis menjadi beberapa fragmen data yang lebih kecil biasa disebut dengan *chunk*. Melalui *hypertable*, kueri SQL dapat dijalankan seolah-olah berinteraksi dengan satu tabel tunggal. Namun secara di latar belakang, berinteraksi dengan chunk sebagai partisi dari tabel fakta.

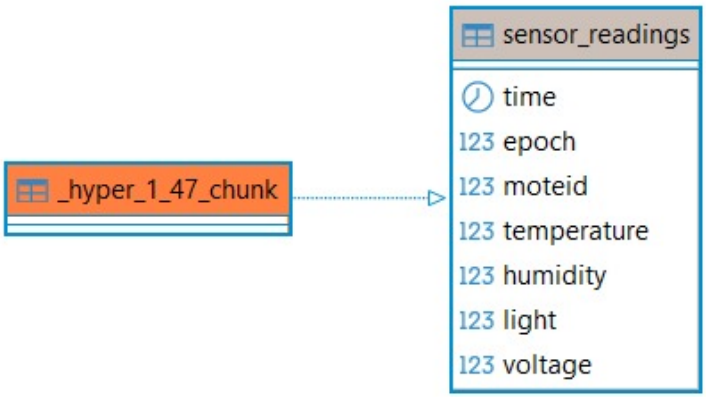
Wujud fisik dari partisi *chunk* yang menginduk pada *hypertable* `sensor_readings` ditunjukkan pada Gambar 4.1 dan Gambar 4.2.



Gambar 4.1: Pembentukan multi-*chunk* pada *hypertable* `sensor_readings`

Implementasi multi-*chunk* pada gambar 4.1 sebagai penyusun *hypertable* `sensor_readings`, sedangkan Gambar 4.2 menunjukkan isi partisi *chunk* terhadap yang merupakan tabel fakta. Setiap *chunk* menampung baris data pengukuran pada rentang

waktu tertentu.



Gambar 4.2: Pemetaan entitas partisi *chunk* terhadap tabel fakta

Proses mempartisi *hypertable* menjadi partisi-partisi *chunk* seperti yang terlihat pada Gambar 4.1 dikendalikan oleh parameter *chunk time interval*. Sebagai contoh jika *chunk time interval* adalah 7 hari, maka jumlah chunk yang ada pada *hypertable* yaitu jumlah hari pada data dibagi dengan tujuh. Setiap *chunk* dibuat secara otomatis untuk menampung baris data pengukuran pada satu rentang waktu yang spesifik.

Hypertable yang terpartisi berdasarkan atribut *time* ini menjadi fondasi berjalannya mekanisme *chunk pruning* dan kebijakan kompresi pada tahap *offline* selanjutnya. Ketika kueri analitik dijalankan dengan filter rentang waktu tertentu, susunan ini memastikan bahwa data hanya dibaca oada *chunk* yang relevan dengan rentang waktu kueri tersebut sehingga penggunaan *resource* memori dapat di minimalisir.

4.3 Pembentukan *Workload* Kueri

Pada penelitian ini, *workload* didefinisikan sebagai sekumpulan kueri yang dieksekusi secara bersamaan untuk merepresentasikan satu pola kebutuhan analitik dari pengguna. Pembentukan *workload* ini digunakan untuk mengevaluasi apakah sebuah kandidat konfigurasi partisi dan kompresi yang dipilih mampu mempertahankan kinerja terbaiknya ketika dihadapkan pada *workload* yang berubah-ubah.

4.3.1 Bentuk *Workload* Uji

Penelitian ini mengimplementasikan tiga variasi *workload* utama untuk menguji kinerja *data warehouse*. Terdapat tiga *Workload* yang diuji: *Workload A*, *Workload B*, dan *Workload C*. *Workload A* difokuskan untuk memindai data detail pada rentang waktu yang sangat singkat yaitu 1 hari (*point lookup*). *Workload B* digunakan untuk mengeksekusi agregasi dan rekapitulasi data mingguan pada berdasarkan `moteid` (*medium aggregation*). Sementara itu, *Workload C* digunakan untuk memindai riwayat bulanan (*historical scan*) pada rentang waktu yang jauh lebih panjang. Ringkasan karakteristik ketiga *workload* tersebut ditunjukkan pada Tabel 4.9.

Tabel 4.9: Bentuk *workload* uji

Label	**Bentuk kueri**
Workload A	*point lookup*
Workload B	*medium aggregation*
Workload C	*heavy historical scan*

Karakteristik pada Tabel 4.9 direpresentasikan ke dalam sintaks kueri analitik. Sebagai contoh, kueri untuk *Workload A* yang berfokus pada pencarian data spesifik (*point lookup*) diringkas pada Listing 4.4.

Listing 4.4: Sintaks kueri untuk *Workload A*

```
SELECT sr.time,
       sr.moteid,
       sr.temperature,
       sr.humidity,
       sr.light,
       sr.voltage
FROM sensor_readings sr
WHERE sr.moteid = 48
  AND sr.time >= timestamp '2004-07-01␣09:00:00'
  AND sr.time < timestamp '2004-07-01␣10:00:00'
ORDER BY sr.time;
```

Sintaks pada Listing 4.4 menunjukkan bahwa kueri hanya meminta data tanpa agregasi pada rentang waktu satu jam. Kueri pada *Workload B* digunakan untuk melakukan perhitungan statistika dasar seperti rata-rata yang dikelompokkan berdasarkan hari dan sensor. Bentuk kueri tersebut dipaparkan pada Listing 4.5.

Listing 4.5: Sintaks kueri agregasi untuk *Workload B*

```
SELECT date_trunc('day', sr.time) AS day_bucket,
       sr.moteid,
       avg(sr.temperature) AS avg_temperature,
       avg(sr.humidity) AS avg_humidity,
       avg(sr.light) AS avg_light,
       avg(sr.voltage) AS avg_voltage,
       count(*) AS row_count
FROM sensor_readings sr
WHERE sr.time >= timestamp '2004-07-01␣00:00:00'
  AND sr.time < timestamp '2004-07-08␣00:00:00'
GROUP BY 1, 2
ORDER BY 1, 2;
```

Kueri pada Listing 4.5 memberikan komputasi *cost* yang menengah karena sistem harus memindai data selama satu minggu. Selanjutnya, untuk memberikan tekanan

memori dan I/O disk, sistem mengeksekusi *Workload C* memaksa *data warehouse* membaca rentang historis selama satu bulan penuh. Sintaks kuerinya ditunjukkan pada Listing 4.6.

Listing 4.6: Sintaks kueri pemindaian historis untuk *Workload C*

```
SELECT date_trunc('day', sr.time) AS day_bucket,
       count(*) AS row_count,
       avg(sr.temperature) AS avg_temperature,
       avg(sr.humidity) AS avg_humidity,
       avg(sr.light) AS avg_light,
       avg(sr.voltage) AS avg_voltage
FROM sensor_readings sr
WHERE sr.time >= timestamp '2004-07-01␣00:00:00'
  AND sr.time < timestamp '2004-07-31␣00:00:00'
GROUP BY 1
ORDER BY 1;
```

Ketiga variasi kueri pada *listing* di atas digunakan sebagai *predefined query*. Penelitian ini juga menggunakan sembilan *ad-hoc query* dengan rentang waktu dan bentuk agregasi yang berbeda. Rinciannya ditunjukkan pada Tabel 4.10.

Tabel 4.10: Variasi *ad-hoc query* pada tahap *online*

Kode	**Bentuk kueri**	**Rentang waktu**
A1–A3	Pencarian data satu sensor	6 jam, 12 jam, dan 1 hari
B1–B3	Agregasi berdasarkan waktu dan sensor	3 hari, 14 hari, dan 7 hari
C1–C3	Agregasi data historis	45 hari, 60 hari, dan 30 hari

Ad-hoc query dipakai saat komposisi kueri pada tahap *online* berubah. Variasi rentang waktu dan bentuk agregasi menghasilkan akses data yang berbeda dari tiga *predefined query*.

4.3.2 *Workload Query Log*

Setiap kali *workload* uji dieksekusi, sistem secara otomatis menghasilkan catatan riwayat yang disebut *query log*. Di dalam *framework* PROADAPT, *query log* ini memiliki dua fungsi yaitu sebagai *input* dari tahap *offline* maupun tahap *online*.

Pada tahap *offline*, *query log* ini digunakan sebagai *input* untuk proses *workload clustering*. Sistem menganalisis pola akses kueri masa lalu untuk menentukan konfigurasi awal yang paling optimal. Setelah konfigurasi awal terbentuk dan sistem memasuki tahap *online*, *query log* berubah peran menjadi instrumen pemantauan *workloads* (*monitoring log*). Pada tahap ini, log secara aktif merekam hasil komputasi dari setiap *window* pengamatan, seperti latensi kueri, estimasi utilitas, aksi adaptasi yang dipilih, serta mengetahui jumlah *chunk* yang terkompresi.

Berikut merupakan gambaran perbedaan antara kedua fungsi tersebut, rincian struktur *query log* yang diambil dari database menggunakan ekstensi `pg_stat_statements` pada tahap *offline* ditunjukkan pada Tabel 4.11.

Tabel 4.11: Struktur kolom *query log* pada tahap *offline*

Kolom	**Keterangan**
`query`	Sintaks kueri SQL
`calls`	Jumlah kueri dieksekusi
`mean_exec_time`	Rata-rata waktu latensi kueri
`shared_blks_hit`	Jumlah blok yang ditemukan langsung di dalam buffer
`shared_blks_read`	Jumlah blok memori (RAM) yang dibaca dari *disk*
`temp_blks_written`	Jumlah blok memori (RAM) yang ditulis ke dalam *disk*
`blk_read_time`	Waktu CPU menunggu pembacaan *disk*

Satu blok PostgreSQL pada lingkungan penelitian berukuran 8 KB. Kolom `shared_blks_hit` dan `shared_blks_read` digunakan untuk menghitung BMR dan WSS. Kolom `temp_blks_written` mencatat blok sementara yang ditulis ketika operasi kueri memerlukan ruang kerja tambahan. Statistik tersebut menjadi input perhitungan *memory score* pada tahap *offline*.

Berikut contoh data *query log* tahap *offline* untuk (*Workload C*):

Tabel 4.12: Contoh rekaman *query log* tahap *offline* (*Workload C*)

Atribut	**Nilai**
`query`	`WITH params AS (SELECT $1::timestamp AS start_ts, ($2::timestamp + interval $3) AS end_ts) SELECT date_trunc($4, sr.time) AS day_bucket, count(*) AS row_count, avg(sr.temperature) AS avg_temp, avg(sr.humidity) AS avg_hum, avg(sr.light) AS avg_light, avg(sr.voltage) AS avg_volt FROM sensor_readings sr CROSS JOIN params p WHERE sr.time >= p.start_ts AND sr.time < p.end_ts GROUP BY 1 ORDER BY 1;`
`calls`	4
`mean_exec_time`	1858,47 ms
`shared_blks_hit`	12.640 blok
`shared_blks_read`	5705 blok
`temp_blks_written`	0 blok
`blk_read_time`	850,08 ms

Data pada Tabel 4.11 ini menjadi input pada tahap *offline*. Sistem melakukan iterasi (*looping*) perhitungan matematis terhadap data kueri masa lalu tersebut untuk menguji kelayakan berbagai kandidat konfigurasi partisi sebelum sistem menyala.

Berbeda dengan *query log* di atas, rincian atribut metrik adaptasi yang direkam di dalam *workload query log* pada saat tahap *online* berlangsung diringkas pada Tabel 4.13.

Tabel 4.13: Struktur kolom *workload query log* pada tahap *online*

Kolom	**Keterangan**
`window_index`	Indeks pengamatan pada tahap *online*
`window_mix`	Persentase *predefined query* pada *window*
`action`	Aksi adaptasi yang dilakukan
`estimated_window_utility`	Nilai estimasi utilitas pada pengamatan
`estimated_utility_drop`	Nilai penurunan utilitas terhadap konfigurasi acuan
`compress_after`	Kebijakan kompresi yang diterapkan
`buffer_miss_ratio`	Nilai BMR pada *window*
`working_set_mb`	Nilai WSS dalam MB

Nilai pada Tabel 4.13 dicatat setelah seluruh kueri pada satu *window* selesai. Contoh satu baris hasilnya ditunjukkan pada Tabel 4.14.

Tabel 4.14: Contoh rekapan adaptasi pada *workload query log*

Kolom	**Nilai contoh**
`window_index`	1
`window_mix`	100%
`action`	*rewrite*
`estimated_window_utility`	0,5660
`estimated_utility_drop`	0,1010
`compress_after`	7 hari
`buffer_miss_ratio`	0,000496
`working_set_mb`	38.809,22

Pada contoh tersebut, *window* pertama hanya memuat *predefined query*. Gap utilitas sebesar 0,1010 disertai tekanan I/O, sehingga aksi yang dipilih adalah *rewrite*. BMR dan WSS dari *window* yang sama digunakan untuk membaca perubahan akses blok setelah kueri dijalankan.

4.4 *Partitioning Schema Selection* (Tahap *Offline*)

Partitioning Schema Selection atau selanjutnya disebut Tahap *offline* digunakan untuk mendapatkan konfigurasi awal dengan nilai utilitas terbaik. Pada tahap ini, *workload* dijalankan pada beberapa kandidat konfigurasi, kemudian sistem menghitung nilai utilitas dari setiap kandidat konfigurasi. Nilai utilitas ini menjadi pertimbangan untuk menentukan kandidat konfigurasi yang akan digunakan pada data warehouse.

4.4.1 *Workload Clustering*

Workload clustering adalah proses pengelompokan query berdasarkan kemiripan pola akses datanya seperti rentang waktu kueri dan seberapa berat dilakukan operasi agregasi. Pada penelitian ini, *workload clustering* digunakan untuk mengelompokkan variasi workload menjadi beberapa kelompok atau yang disebut *cluster*, sehingga tahap offline dapat mengevaluasi kandidat konfigurasi *chunk_time_interval* dan kebijakan kompresi secara lebih efisien.

Input dan *output* dari proses *workload clustering* ditunjukkan pada Tabel 4.15. Pada tahap ini, sistem menerima ciri *workload* berupa rentang waktu kueri dan intensitas agregasi, lalu mengelompokkannya ke dalam *cluster* yang diwakili oleh *workload* terpilih.

Tabel 4.15: Hasil dari *workload clustering*

Rentang waktu kueri	**Intensitas agregasi**	**Diwakili sebagai**	**Cluster**
3 jam	Tanpa agregasi	*Workload A*	Cluster detail (`cluster_01`)
72 jam	Agregasi menengah	*Workload B*	Cluster menengah (`cluster_01`)
720 jam	Agregasi historis lebih berat	*Workload C*	Cluster historis (`cluster_03`)

Tabel 4.15 menunjukkan *input* dan *output* dari *workload clustering*. Hasil klasterisasi ini dipakai sebagai *input* pada tahap berikutnya yaitu *Partition Candidate Detector* untuk menentukan ruang kandidat konfigurasi pada *cluster workload* yang mewakili *Workload A*, *Workload B*, dan *Workload C*.

4.4.2 *Partition Candidate Detector*

Sesudah *cluster* diperoleh, tahap berikutnya adalah mencari nilai parameter kandidat yang akan digunakan untuk membentuk kandidat konfigurasi sebelum dievaluasi. Terdapat dua parameter kandidat yang diuji, yaitu *chunk interval* dan `compress_after`. *Chunk interval* adalah panjang rentang waktu pada setiap *chunk*. `compress_after` adalah batas waktu sebelum data didalam *chunk* mulai dikompresi. Berikut implementasi pembentukan nilai parameter kandidat yaitu *output* dari *Partition Candidate Detector*.

Tabel 4.16: Nilai parameter kandidat

Parameter	**Nilai yang diuji**
chunk interval	3 hari, 7 hari, 14 hari, 30 hari
`compress_after`	3 hari, 7 hari

Setelah nilai parameter kandidat konfigurasi ditemukan, tahap selanjutnya adalah membentuk kandidat konfigurasi yang akan dievaluasi pada tahap *utility estimator*.

4.4.3 Pembentukan Kandidat Konfigurasi

Nilai pada Tabel 4.16 kemudian dikombinasikan menjadi kandidat konfigurasi. *input* tahap ini adalah nilai parameter *chunk interval* dan `compress_after` yang sudah didapatkan pada tahap sebelumnya. *Output* dari tahap ini adalah daftar ruang kandidat konfigurasi yang siap dievaluasi oleh *utility estimator*.

Pada penelitian ini kandidat konfigurasi berupa satu parameter kandidat yaitu *chunk interval* dengan nilai dari `compress_after` dibiarkan tetap, maupun kombinasi dari dua parameter kandidat, yaitu *chunk interval* dan `compress_after`

Pembentukan kandidat satu parameter ditunjukkan pada Tabel 4.17.

Tabel 4.17: Kandidat konfigurasi yang diuji dengan satu parameter *chunk interval*

Kode kandidat	***chunk interval***
`baseline_7d`	7 hari
`candidate_14d`	14 hari
`candidate_30d`	30 hari
`candidate_3d`	3 hari

Pada kandidat konfigurasi satu parameter, nilai `compress_after` dibiarkan tetap. Sedangkan pada kandidat konfigurasi dua parameter, nilai `compress_after` dapat berupa kombinasi 3 hari dan 7 hari. Pembentukan kandidat dua parameter ditunjukkan pada Tabel 4.18.

Tabel 4.18: Kandidat konfigurasi yang diuji dengan dua parameter *chunk interval* dan `compress_after`

Kode kandidat	***chunk interval***	`compress_after`
`candidate_7d_ca_7d`	7 hari	7 hari
`candidate_14d_ca_7d`	14 hari	7 hari
`candidate_30d_ca_7d`	30 hari	7 hari
`candidate_7d_ca_3d`	7 hari	3 hari
`candidate_14d_ca_3d`	14 hari	3 hari
`candidate_30d_ca_3d`	30 hari	3 hari

Dengan kombinasi dua parameter kandidat, ditemukan enam kandidat konfigurasi yang siap dievaluasi. Setelah kandidat konfigurasi terbentuk, tahap selanjutnya adalah menghitung nilai utilitas dari setiap kandidat konfigurasi tersebut.

4.4.4 *Utility Estimator Memory-Aware*

Tahap ini menghitung nilai utilitas setiap kandidat konfigurasi berdasarkan hasil eksekusi *workload*. Misalkan C merupakan himpunan kandidat konfigurasi dan $c \in C$ merupakan satu kandidat yang sedang dinilai. Setiap kandidat memuat nilai *chunk*

interval dan, pada skenario tertentu, nilai `compress_after`. Selanjutnya, $k \in K$ menyatakan jenis *workload*, dengan $K = \{A, B, C\}$.

Fungsi utilitas pada penelitian ini di sesuaikan dari persamaan utilitas PROADAPT agar sesuai dengan *tools* TimescaleDB dalam lingkungan *single-VM*. Prinsip yang digunakan adalah penggunaan nilai utilitas sebagai dasar untuk membandingkan dan memilih konfigurasi terbaik. Komponen penilaiannya yaitu latensi, efektivitas *chunk pruning*, BMR, WSS, ukuran penyimpanan, dan *overhead*.

Setiap metrik memiliki satuan dan rentang nilai yang berbeda. Oleh karena itu, nilai pengukuran dinormalisasi ke rentang 0 sampai 1 di dalam ruang kandidat yang sama. Normalisasi metrik yang semakin kecil semakin baik, digunakan normalisasi invers yang ditulis sebagai berikut.

$$N^-(x) = \begin{cases} \dfrac{x_{\max} - x}{x_{\max} - x_{\min}}, & x_{\max} \neq x_{\min}, \\ 0, & x_{\max} = x_{\min}. \end{cases} \tag{4.1}$$

Metrik yang digunakan sebagai penalti, digunakan normalisasi sebagai berikut.

$$N^+(x) = \begin{cases} \dfrac{x - x_{\min}}{x_{\max} - x_{\min}}, & x_{\max} \neq x_{\min}, \\ 0, & x_{\max} = x_{\min}. \end{cases} \tag{4.2}$$

Pada kedua persamaan normalisasi tersebut, x menyatakan nilai metrik kandidat, sedangkan $x_{\min}$ dan $x_{\max}$ merupakan nilai minimum dan maksimum pada evaluasi yang sama. Nilai 0 digunakan ketika seluruh kandidat memiliki nilai yang sama karena komponen tersebut tidak membedakan urutan kandidat dan pembagian dengan nol harus dihindari.

Skor latensi dibentuk dari rata-rata hasil normalisasi invers p50 dan p95. *Memory score* dibentuk dari rata-rata hasil normalisasi invers BMR dan WSS. Kedua skor tersebut dirumuskan sebagai berikut.

$$R_k(c) = \frac{N^-(p50_k(c)) + N^-(p95_k(c))}{2}, \qquad M_k(c) = \frac{N^-(BMR_k(c)) + N^-(WSS_k(c))}{2}. \tag{4.3}$$

Efektivitas *chunk pruning* dinilai berdasarkan jumlah *chunk* yang dibaca oleh kandidat c pada *workload* k. Jika $C_k(c)$ menyatakan jumlah *chunk* yang dibaca, skor *pruning* dirumuskan sebagai berikut.

$$P_k(c) = N^-(C_k(c)) \,. \tag{4.4}$$

Nilai $P_k(c)$ yang lebih besar menunjukkan bahwa kandidat membaca lebih sedikit *chunk* dibandingkan kandidat lain di dalam evaluasi yang sama. Apabila seluruh kandidat

membaca jumlah *chunk* yang sama, komponen ini bernilai 0 untuk seluruh kandidat dan tidak memengaruhi urutan hasil. Penalti keduanya dirumuskan sebagai berikut.

$$D(c) = N^{+}(\text{Storage}(c)), \qquad O(c) = N^{+}(\text{Overhead}(c)). \tag{4.5}$$

Setelah seluruh komponen dihitung, skor kandidat c pada *workload* k dirumuskan sebagai berikut.

$$W_k(c) = 0{,}60R_k(c) + 0{,}20P_k(c) + 0{,}15M_k(c) - 0{,}05D(c). \tag{4.6}$$

Utilitas akhir tahap *offline* dibentuk dari rata-rata skor Workload A, Workload B, dan Workload C, kemudian dikurangi oleh penalti *overhead*.

$$U(c) = \frac{W_A(c) + W_B(c) + W_C(c)}{3} - 0{,}10O(c) \tag{4.7}$$

Bobot pada fungsi utilitas ditetapkan secara heuristik sebagai kebijakan penilaian dalam eksperimen ini. Nilai 0,60 membuat latensi menjadi komponen prioritas, nilai 0,20 dan 0,15 menempatkan efektivitas *pruning* dan *memory score* sebagai komponen pendamping, sedangkan ukuran penyimpanan dan *overhead* diperlakukan sebagai penalti.

Contoh perhitungan menggunakan hasil kandidat `candidate_3d` pada *Workload A*. Data pengukurannya ditunjukkan pada Tabel 4.19.

Tabel 4.19: Hasil pengukuran `candidate_3d` pada *Workload A*

Kandidat	***p50*** **(ms)**	***p95*** **(ms)**	***Chunk*** **terbaca**	`shared_hit`	`shared_read`	`temp_written`	**Storage (MB)**	***Overhead***
`candidate_3d`	2609,50	3155,72	1	257	7	0	962,09	302,20

Nilai BMR dan WSS dihitung dari blok yang tercatat pada *query log*.

$$\begin{aligned} BMR_A(c) &= \frac{7}{257+7} = 0{,}0265, \\ WSS_A(c) &= (257+7) \times 8 \text{ KB} = 2112 \text{ KB} = 2{,}0625 \text{ MB}. \end{aligned} \tag{4.8}$$

Hasil normalisasi kandidat tersebut diringkas pada Tabel 4.20.

Tabel 4.20: Hasil normalisasi `candidate_3d` pada *Workload A*

Komponen utilitas	**Nilai**
Latency score $R_A(c)$	0,3987
Skor *chunk pruning* $P_A(c)$	0,0000
Memory score $M_A(c)$	0,9639
Penalti penyimpanan $D(c)$	0,0971
Penalti *overhead* $O(c)$	1,0000

Dengan merujuk pada nilai di Tabel 4.20 dan persamaan utilitas workload pada Persamaan 4.6, nilai utilitas untuk *Workload A* dapat dihitung sebagai berikut.

$$\begin{aligned} W_A(\texttt{candidate\_3d}) &= 0{,}60(0{,}3987) + 0{,}20(0{,}0000) \\ &\quad + 0{,}15(0{,}9639) - 0{,}05(0{,}0971) \\ &= 0{,}3789 \end{aligned} \tag{4.9}$$

Perhitungan yang sama menghasilkan skor untuk *Workload B* dan *Workload C*. Rekapannya ditunjukkan pada Tabel 4.21.

Tabel 4.21: Ringkasan skor *workload* untuk `candidate_3d`

Workload	**Skor**
Workload A	0,3789
Workload B	0,4877
Workload C	0,6503

Penalti *overhead* kandidat bernilai 1 setelah normalisasi. Utilitas akhirnya dihitung sebagai berikut.

$$\begin{aligned} U(\texttt{candidate\_3d}) &= \frac{0{,}3789 + 0{,}4877 + 0{,}6503}{3} - 0{,}10(1) \\ &= 0{,}4056 \end{aligned} \tag{4.10}$$

Nilai utilitas akhir `candidate_3d` adalah 0,4056. Sistem menerapkan urutan perhitungan yang sama pada seluruh kandidat, kemudian memilih nilai tertinggi pada setiap skenario sebagai konfigurasi tahap *online*.

4.4.5 Konfigurasi Terbaik dan Metadata Partisi

Sesudah semua kandidat dinilai, tahap *offline* memilih satu konfigurasi dengan utilitas tertinggi. Bersamaan dengan itu, sistem juga membaca metadata partisi agar hasil pemilihan kandidat dapat langsung dikaitkan dengan keadaan *data warehouse*.

Contoh metadata yang dibaca sesudah konfigurasi dipilih ditunjukkan pada Tabel 4.22.

Tabel 4.22: Contoh metadata partisi sesudah konfigurasi dipilih

Metadata	**Nilai contoh**
Jumlah baris	44.396.060
Rentang waktu data	2004-02-28 00:58:46.002832 s.d. 2005-09-19 11:02:32.715337
Jumlah *chunk*	191
Chunk interval	3 hari
`compress_after`	7 hari
Chunk terkompresi	188 dari 191 *chunk*
Ukuran penyimpanan	962,09 MB

output tahap *offline* adalah konfigurasi acuan *chunk time interval* dan kebijakan kompresi beserta metadata partisinya. Konfigurasi ini kemudian dipakai sebagai dasar implementasi tahap *online*.

4.5 *Adaptive Partition (Online)*

Tahap *online* dimulai setelah konfigurasi terbaik dari tahap *offline* dipilih. Sistem menjalankan sekumpulan kueri pada setiap *window*, mencatat metrik eksekusi, menghitung estimasi utilitas, lalu memilih aksi adaptasi. Persentase *predefined query* diturunkan secara bertahap dari 100% menjadi 0%. Persentase sisanya merupakan *ad-hoc query*.

4.5.1 *Workload Monitor*

Workload Monitor membaca keadaan sistem pada setiap *window*. Satu *window* adalah satu periode pengamatan yang menjalankan sekumpulan kueri dengan komposisi tertentu. Seluruh kueri pada satu *window* diselesaikan terlebih dahulu, kemudian sistem membentuk satu ringkasan keadaan dan memilih satu aksi. Aksi tersebut diterapkan setelah pengukuran *window* selesai dan memengaruhi keadaan pada *window* berikutnya. Ringkasan yang dibentuk memuat komposisi kueri, latensi, statistik blok, WSS, kebijakan kompresi aktif, dan keadaan aksi sebelumnya. Ringkasan ini menjadi input *Utility Estimator Online*.

Contoh ringkasan satu *window* ditunjukkan pada Tabel 4.23.

Tabel 4.23: Contoh hasil pembacaan *Workload Monitor* pada satu *window*

Komponen	**Nilai contoh**
Window ke-	1
Persentase *predefined query*	100%
Utilitas konfigurasi terpilih *offline*	0,4700
BMR	0,000496
WSS	38.809,22 MB
`compress_after` aktif	7 hari
Latensi kueri	280,952 ms

Pada contoh tersebut, *window* pertama menjalankan seluruh *predefined query*. Hasil pengukuran menunjukkan BMR sebesar 0,000496, WSS sebesar 38.809,22 MB, dan latensi 280,952 ms ketika `compress_after` masih bernilai 7 hari.

4.5.2 *Utility Estimator Online*

Utility Estimator Online menilai apakah konfigurasi yang sedang digunakan masih cukup sesuai pada *window* ke-t. Nilai U_t^* merupakan nilai acuan yang sudah disesuaikan dengan komposisi kueri pada windoweksekusi kueri di dalam *window*. Nilai acuan tersebut kemudian dikurangi oleh penalti BMR, WSS, penulisan blok sementara, dan aksi pada *window* sebelumnya. Seluruh komponen penalti berada pada rentang 0 sampai 1.

$$\widehat{U}_t = U_t^* - 0{,}10B_t - 0{,}10M_t - 0{,}08S_t - 0{,}04A_t \tag{4.11}$$

Pada persamaan tersebut, B_t merupakan penalti BMR, M_t merupakan penalti WSS, S_t merupakan penalti penulisan blok sementara, dan A_t merupakan penalti aksi sebelumnya. Keempat hal tersebut membentuk estimasi utilitas untuk tahap *online* yang digunakan sebagai dasar pengambilan keputusan pada tahap *online*.

Selisih antara utilitas acuan dan estimasi utilitas dinyatakan sebagai *gap* utilitas.

$$G_t = \max\left(0, U_t^* - \widehat{U}_t\right). \tag{4.12}$$

Nilai G_t menunjukkan besarnya penurunan utilitas pada *window* ke-t. Ambang *gap* ditetapkan sebesar 0,08 pada implementasi penelitian ini. Nilai $G_t \leq 0{,}08$ menunjukkan bahwa keadaan masih cukup dekat dengan utilitas konfigurasi acuan, sedangkan nilai $G_t > 0{,}08$ diteruskan ke tahap *Adaptive Schema Selector* untuk memilih aksi yang sesuai.

Koefisien penalti 0,10; 0,10; 0,08; dan 0,04 serta ambang 0,08 ditetapkan sebagai parameter heuristik pada implementasi penelitian. Tujuannya sebagai aturan yang diterapkan secara konsisten pada seluruh *window* pengujian.

Contoh perhitungan satu *window* ditunjukkan pada Tabel 4.24.

Tabel 4.24: Contoh komponen perhitungan estimator *online*

Komponen	**Nilai**
Nilai acuan sesuai komposisi kueri U_t^*	0,666989
Penalti BMR B_t	0,009920
Penalti WSS M_t	1,000000
Penalti blok sementara S_t	0,000000
Penalti adaptasi A_t	0,000000
Estimasi utilitas *window* $\widehat{U}_t$	0,565997
Gap utilitas G_t	0,100992

Perhitungan estimasi nilai utilitas pada Tabel 4.24 dapat dihitung dengan Persamaan 4.11 dan dapat ditulis sebagai berikut.

$$\begin{aligned}\widehat{U}_t &= 0{,}666989 - 0{,}10(0{,}009920) - 0{,}10(1) \\ &\quad - 0{,}08(0) - 0{,}04(0) \\ &= 0{,}565997 \end{aligned} \tag{4.13}$$

Estimasi utilitas *window* pertama adalah 0,565997. Gap utilitasnya dihitung sebagai berikut.

$$\begin{aligned}G_t &= \max(0, 0{,}666989 - 0{,}565997) \\ &= 0{,}100992 \end{aligned} \tag{4.14}$$

output dari tahap ini adalah estimasi utilitas pada *window* yang sedang dibaca dan gap utilitas terhadap konfigurasi acuan. Nilai inilah yang dipakai pada tahap berikutnya untuk memilih aksi.

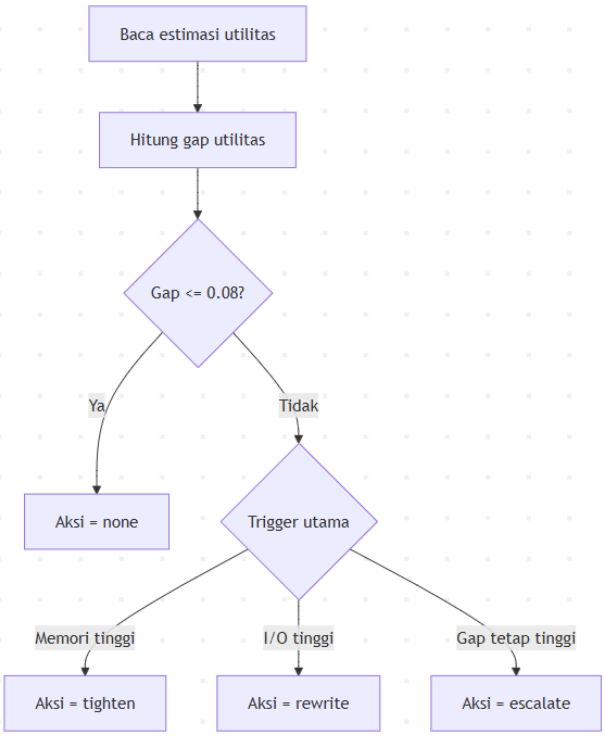
4.5.3 *Adaptive Schema Selector*

Sesudah estimasi utilitas dibaca, sistem memilih aksi adaptasi minor yang paling sesuai. Masukan tahap ini adalah estimasi utilitas, gap utilitas, dan kondisi sistem pada *window* aktif. *output* tahap ini adalah satu aksi akhir, yaitu *none*, *rewrite*, *tighten*, *relax*, atau *escalate*. Ringkasannya ditunjukkan pada Tabel 4.25.

Tabel 4.25: Aksi pada tahap *online*

Aksi	**Arti**
none	Kondisi sudah aman, *cooldown* aktif, atau manfaat adaptasi lebih kecil daripada biaya penerapannya
rewrite	Menjalankan *query rewriter* agar batas waktu kueri lebih jelas
tighten	Mempersingkat `compress_after`
relax	Memperpanjang waktu `compress_after`
escalate	Mengambil keputusan kembali ke tahap *offline*

Alur pemilihan aksi pada tahap *online* ditunjukkan pada Gambar 4.3. Gambar ini memperlihatkan urutan pembacaan estimasi utilitas, perhitungan gap utilitas, dan pemilihan aksi yang dijalankan pada tahap *online*.



Gambar 4.3: Alur pemilihan aksi pada tahap *online*

Satu contoh keputusan berurutan ditunjukkan pada Tabel 4.26.

Tabel 4.26: Contoh urutan keputusan pada tiga *window* awal

Window	**Komposisi**	**Utilitas**	**Gap**	**Detail**
1	100%	0,5660	0,1010	Tekanan I/O terdeteksi, sehingga aksi yang dipilih adalah *rewrite*
2	75%	0,5356	0,1400	*Cooldown* aktif, sehingga aksi yang dipilih adalah *none*
3	50%	0,4921	0,1800	Gap utilitas tetap di atas ambang, sehingga aksi yang dipilih adalah *escalate*

Pada *window* pertama, tekanan I/O memicu *query rewriting*. *Window* kedua tidak menerapkan perubahan karena *cooldown* masih aktif. Gap utilitas pada *window* ketiga tetap berada di atas ambang, sehingga aksi yang dipilih adalah *escalate*.

4.5.4 *Query Rewriter*

Query rewriter dipakai ketika *adaptive schema selector* memilih aksi *rewrite*. Tahap ini memperjelas batas waktu kueri agar sesuai dengan rentang *chunk* yang ada. Aturan ini berguna pada kueri yang masih memakai parameter waktu dinamis atau bentuk waktu yang belum langsung terbaca sebagai rentang eksplisit.

Contoh kueri sebelum dan sesudah *rewrite* ditunjukkan pada Listing 4.7 dan Listing 4.8.

Listing 4.7: Contoh kueri sebelum *rewrite*

```
WITH latest_data AS (
    SELECT max(time) AS end_ts
    FROM sensor_readings
),
params AS (
    SELECT f.end_ts - interval '3␣hours' AS start_ts,
           f.end_ts
    FROM latest_data f
)
SELECT sr.time,
       sr.moteid,
       sr.temperature
FROM sensor_readings sr
CROSS JOIN params p
WHERE sr.time >= p.start_ts
  AND sr.time < p.end_ts
ORDER BY sr.time;
```

Listing 4.8: Contoh kueri sesudah *rewrite*

```
WITH params AS (
    SELECT timestamp '2005-09-19␣08:02:32.715337' AS start_ts,
           timestamp '2005-09-19␣11:02:32.715337' AS end_ts
)
SELECT sr.time,
       sr.moteid,
       sr.temperature
FROM sensor_readings sr
CROSS JOIN params p
WHERE sr.time >= timestamp '2005-09-19␣08:02:32.715337'
  AND sr.time < timestamp '2005-09-19␣11:02:32.715337'
ORDER BY sr.time;
```

Perubahan pada Listing 4.8 membuat batas awal dan batas akhir waktu terlihat langsung. Bentuk ini sesuai dengan hasil *rewrite* yang benar-benar dijalankan pada implementasi tahap *online*.

4.5.5 Konfigurasi Update

Jika aksi yang dipilih adalah perubahan kebijakan kompresi, sistem memperbarui nilai `compress_after`. Tahap ini menerima hasil keputusan dari *adaptive schema selector*, lalu menerapkannya pada konfigurasi yang sedang aktif. Pada tahap *online*, perubahan hanya

dilakukan pada `compress_after`, sedangkan *chunk interval* tetap mengikuti konfigurasi acuan *offline*.

Satu contoh perubahan nilai `compress_after` ditunjukkan pada Tabel 4.27.

Tabel 4.27: Contoh perubahan `compress_after` pada tahap *online*

Komponen	**Nilai contoh**
Persentase *predefined query*	25%
Utilitas *window*	0,4765
Gap utilitas	0,1960
Trigger	kebijakan kompresi perlu dipercepat
`compress_after` sebelum	7 hari
`compress_after` sesudah	3 hari
Overhead adaptasi	0,4

Pada contoh tersebut, nilai `compress_after` berubah dari 7 hari menjadi 3 hari setelah pengukuran *window* selesai. Nilai baru digunakan pada *window* berikutnya, sedangkan *chunk interval* tetap mengikuti konfigurasi hasil tahap *offline*.

4.6 Analisis Kompleksitas Mekanisme

Analisis kompleksitas digunakan untuk membaca biaya komputasi dari mekanisme yang dirancang. Bagian ini menjelaskan bentuk biaya dari proses pemindaian data, seleksi konfigurasi tahap *offline*, dan pemilihan aksi tahap *online*. Notasi yang digunakan pada analisis ini diringkas pada Tabel 4.28.

Tabel 4.28: Notasi analisis kompleksitas mekanisme

Notasi	**Keterangan**
B	Jumlah blok data pada tabel yang dapat terbaca ketika pemindaian dilakukan tanpa pemangkasan *chunk*.
B_r	Jumlah blok data relevan yang benar-benar dibaca setelah *chunk pruning*.
C	Jumlah *chunk* yang diperiksa pada metadata partisi.
m	Jumlah kandidat konfigurasi pada tahap *offline*.
q	Jumlah kueri uji pada satu *workload*.
r	Jumlah pengulangan eksekusi kueri.
p	Jumlah komponen metrik yang dihitung pada fungsi utilitas.
q_w	Jumlah kueri di dalam satu *window* tahap *online*.
L	Panjang kueri yang dibaca oleh *query rewriter*.
a	Jumlah aksi adaptasi minor yang diperiksa oleh *adaptive schema selector*.
w	Jumlah *workload* yang dijalankan pada setiap kandidat.

Pada data warehouse, kueri analitik tanpa pemangkasan partisi dapat membaca blok data dalam jumlah besar. Kompleksitas pemindaian penuh ditulis pada Persamaan 4.15.

$$T_{scan} = O(B). \tag{4.15}$$

Ketika *chunk pruning* berjalan, sistem tetap memeriksa metadata *chunk*, tetapi blok data yang dibaca hanya blok yang sesuai dengan rentang waktu kueri. Kompleksitasnya ditulis pada Persamaan 4.16.

$$T_{prune} = O(C + B_r). \tag{4.16}$$

Persamaan 4.16 menunjukkan bahwa biaya baca data bergeser dari seluruh blok pada tabel menuju metadata *chunk* dan blok yang relevan. Pada konteks TimescaleDB, nilai B_r dipengaruhi oleh *chunk interval*, rentang waktu kueri, dan efektivitas *chunk pruning*.

Pada tahap *offline*, setiap kandidat konfigurasi menjalankan kumpulan kueri uji, kemudian hasilnya dipakai untuk menghitung nilai utilitas. Kompleksitas seleksi tahap *offline* ditulis pada Persamaan 4.17.

$$T_{offline} = O(m \cdot w \cdot q \cdot r \cdot T_{query}) + O(m \cdot w \cdot q \cdot p)\,. \tag{4.17}$$

Komponen pertama pada Persamaan 4.17 berasal dari eksekusi kueri pada seluruh kandidat. Komponen kedua berasal dari perhitungan metrik dan utilitas. Pada penelitian ini, bagian eksekusi kueri menjadi bagian yang paling besar karena setiap kandidat harus dijalankan pada *workload* yang sama.

Pada tahap *online*, satu *window* menjalankan sekumpulan kueri, membaca hasil pengukuran, menghitung estimasi utilitas, dan memilih satu aksi. Kompleksitas per *window* ditulis pada Persamaan 4.18.

$$T_{online} = O(q_w \cdot T_{query}) + O(L) + O(p + a). \tag{4.18}$$

Komponen $O(q_w \cdot T_{query})$ berasal dari eksekusi kueri dalam *window*. Komponen $O(L)$ muncul ketika *query rewriter* membaca dan menata ulang batas waktu kueri. Komponen $O(p + a)$ berasal dari perhitungan metrik dan pemilihan aksi adaptasi minor. Dengan susunan ini, tahap *online* tetap berpusat pada eksekusi kueri, sedangkan keputusan adaptasi dijalankan satu kali setelah seluruh kueri dalam satu *window* selesai.

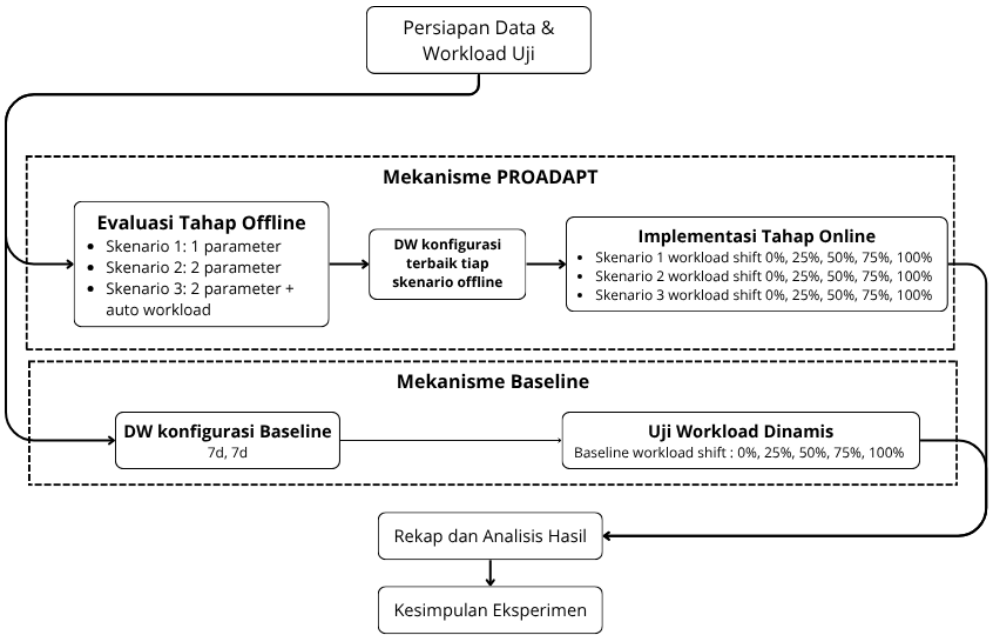
Pada pengelompokan *workload* menggunakan K-Means, biaya pengelompokan dapat ditulis seperti Persamaan 4.19.

$$T_{cluster} = O(n \cdot k \cdot i \cdot d), \tag{4.19}$$

Pada Persamaan 4.19, n menyatakan jumlah kueri pada *query log*, k menyatakan jumlah *cluster*, i menyatakan jumlah iterasi, dan d menyatakan jumlah fitur kueri. Fitur yang digunakan mengikuti ciri yang sudah dijelaskan pada *workload clustering*, yaitu rentang waktu kueri, bentuk agregasi, dan karakter pembacaan historis.

4.7 Pengujian Penelitian

Pengujian penelitian menghubungkan tahap *offline* dan tahap *online*. Pengujian dimulai dari persiapan data dan *workload*, dilanjutkan ke evaluasi tahap *offline*, lalu diteruskan ke implementasi tahap *online*. Rekapan hasil dari kedua tahap tersebut kemudian dipakai sebagai hasil dan dasar analisis pada Bab V.



Gambar 4.4: Alur pengujian penelitian

Gambar 4.4 menunjukkan rancangan alur pengujian yang akan dilakukan. Tahap awal menyiapkan data dan *workload* uji, kemudian mekanisme PROADAPT dijalankan melalui evaluasi tahap *offline* untuk memperoleh konfigurasi terbaik. Konfigurasi tersebut selanjutnya digunakan pada tahap *online* untuk membaca respons sistem terhadap komposisi kueri pada setiap *window*. Hasil dari kedua tahap tersebut akan direkap dan dianalisis pada Bab V.

4.7.1 Evaluasi Tahap *Offline*

Evaluasi tahap *offline* dilakukan untuk memilih konfigurasi awal dengan nilai utilitas terbaik. Tiga bentuk evaluasi yang dipakai diringkas pada Tabel 4.29.

Tabel 4.29: Ringkasan evaluasi tahap *offline*

Skenario	***Workload***	**Parameter yang diuji**	**Kandidat konfigurasi**

Skenario 1	A, B, dan C	*chunk interval*	`baseline_7d`
			`candidate_3d`
			`candidate_14d`
			`candidate_30d`

Skenario 2	A, B, dan C	*chunk interval* dan `compress_after`	`candidate_7d_ca_7d`
			`candidate_14d_ca_7d`
			`candidate_30d_ca_7d`
			`candidate_7d_ca_3d`
			`candidate_14d_ca_3d`
			`candidate_30d_ca_3d`

Skenario 3	*auto_workload_1*
	auto_workload_2
	auto_workload_3
		chunk interval dan `compress_after`	`auto_bound_7d_ca_7d`
			`auto_bound_14d_ca_7d`
			`auto_bound_30d_ca_7d`
			`auto_bound_7d_ca_3d`
			`auto_bound_14d_ca_3d`
			`auto_bound_30d_ca_3d`

Tiga skenario pada Tabel 4.29 memuat 16 kandidat konfigurasi, yaitu 4 kandidat pada Skenario 1 serta masing-masing 6 kandidat pada Skenario 2 dan Skenario 3. Setiap kandidat menjalankan tiga *workload*. Protokol eksekusi terdiri dari satu *warm-up* dan lima pengukuran utama untuk setiap *workload*.

4.7.2 Implementasi Tahap *Online*

Implementasi tahap *online* menggunakan konfigurasi terbaik dari setiap skenario *offline*. Setiap konfigurasi dijalankan selama lima *window*. Persentase *predefined query* diturunkan dari 100% menjadi 0% dengan perubahan 25% pada setiap *window*. Persentase sisanya merupakan *ad-hoc query*.

Ringkasan implementasinya ditunjukkan pada Tabel 4.30.

Tabel 4.30: Ringkasan implementasi tahap *online*

Dasar implementasi	**Komposisi lima *window***	**Metrik hasil**
Baseline 7 hari / 7 hari	100%, 75%, 50%, 25%, 0%	Latensi, BMR, WSS, dan *overhead*
Konfigurasi terbaik Skenario 1	100%, 75%, 50%, 25%, 0%	Latensi, BMR, WSS, aksi, dan *overhead*
Konfigurasi terbaik Skenario 2	100%, 75%, 50%, 25%, 0%	Latensi, BMR, WSS, aksi, dan *overhead*
Konfigurasi terbaik Skenario 3	100%, 75%, 50%, 25%, 0%	Latensi, BMR, WSS, aksi, dan *overhead*

Baseline dan ketiga konfigurasi terpilih menjalankan susunan kueri yang sama. Perbandingan dilakukan per *window* agar perubahan latensi, BMR, WSS, dan *overhead* dapat dibaca pada komposisi kueri yang setara.

BAB V
HASIL DAN PEMBAHASAN

Bab ini menyajikan hasil pengujian dari mekanisme partisi *memory-aware* berbasis PROADAPT yang telah dirancang pada Bab IV. Penyajian hasil dibagi ke dalam dua tahap. Tahap *offline* membaca proses seleksi konfigurasi awal berdasarkan fungsi utilitas, sedangkan tahap *online* membaca perilaku konfigurasi terpilih ketika komposisi kueri berubah. Evaluasi kinerja kemudian dilakukan melalui latensi, BMR, WSS, dan *overhead*.

5.1 Hasil tahap *Offline*

Pada bab ini, hasil tahap *offline* disajikan mulai dari data eksperimen yang digunakan, ruang uji kandidat, hasil seleksi konfigurasi pada tiap skenario, serta pembacaan metrik yang membentuk utilitas.

5.1.1 Data Eksperimen

Pengukuran dilakukan pada PostgreSQL/TimescaleDB *single-VM* menggunakan data Intel Berkeley Research Lab yang telah melalui proses ETL dan *data scaling*. Data acuan disimpan pada database bernama `dw_clean_source`, kemudian direalisasikan sebagai relasi utama `sensor_readings`. Seluruh kandidat diuji pada sumber data dan protokol eksekusi yang sama dengan kondisi sebagai berikut.

Tabel 5.1: Kondisi awal sumber data eksperimen

Aspek	**Kondisi**
Jumlah baris	44.396.060 baris
Objek pengukuran	tabel fakta `sensor_readings`
Waktu minimum data	`2004-02-28 00:58:46.002832`
Waktu maksimum data	`2005-09-19 11:02:32.715337`
Kondisi awal	7 hari, 82 *chunk*, seluruh *chunk* terkompresi

Tabel 5.1 menunjukkan bahwa data uji memiliki rentang waktu historis yang cukup panjang untuk membentuk banyak *chunk*. Kondisi awal 7 hari digunakan sebagai baseline karena konfigurasi tersebut mewakili pengaturan statis yang tidak berubah mengikuti pergeseran kueri. Pada lingkungan uji dengan RAM 4 GB, ukuran data tersebut membuat pembacaan blok, aktivitas buffer, dan kebijakan kompresi menjadi bagian penting dari evaluasi.

Pencatatan hasil dilakukan melalui tiga lapisan pengukuran, yaitu waktu eksekusi, `EXPLAIN (ANALYZE, BUFFERS, VERBOSE)`, dan `pg_stat_statements`. Protokol pengukuran dijaga sama untuk seluruh kandidat dalam setiap *batch*, yaitu satu kali *warm-up*, lima kali pencatatan utama, pengambilan data dari `pg_stat_statements`, serta penyimpanan catatan *runtime*, *plan*, dan metadata fisik.

5.1.2 Ruang Uji Kandidat

Ruang uji kandidat tahap *offline* mengikuti rancangan kandidat pada Tabel 4.16, Tabel 4.17, dan Tabel 4.18. Parameter yang diuji adalah *chunk interval* dan `compress_after`. *Chunk interval* menentukan panjang rentang waktu satu *chunk*, sedangkan `compress_after` menentukan kapan data historis mulai masuk ke kebijakan kompresi.

Perubahan *chunk interval* berpengaruh langsung terhadap jumlah *chunk* yang dikelola hypertable. Hubungan tersebut ditunjukkan pada Tabel 5.2.

Tabel 5.2: Jumlah *chunk* berdasarkan *chunk interval*

Chunk interval	**Jumlah *chunk***
3 hari	191
7 hari	82
14 hari	41
30 hari	20

Tabel 5.2 memperlihatkan bahwa konfigurasi partisi tidak hanya mengubah nama kandidat, tetapi juga mengubah bentuk data secara fisik. Interval 3 hari memberi peluang *pruning* yang lebih rapat, tetapi jumlah *chunk* yang dikelola menjadi lebih besar. Interval 14 hari dan 30 hari menurunkan jumlah *chunk*, tetapi satu *chunk* mencakup rentang waktu yang lebih luas. Perbedaan ini menjadi dasar pembacaan hasil pada latensi, BMR, dan WSS.

Tahap *offline* memiliki tiga skenario. Skenario 1 menguji satu parameter saja, yaitu *chunk interval*. Skenario 2 menguji dua parameter, yaitu *chunk interval* dan `compress_after`. Skenario 3 menggunakan ruang kandidat yang sama seperti Skenario 2, tetapi dijalankan pada *auto workload* terpilih dari proses *workload clustering* pada Tabel 4.15.

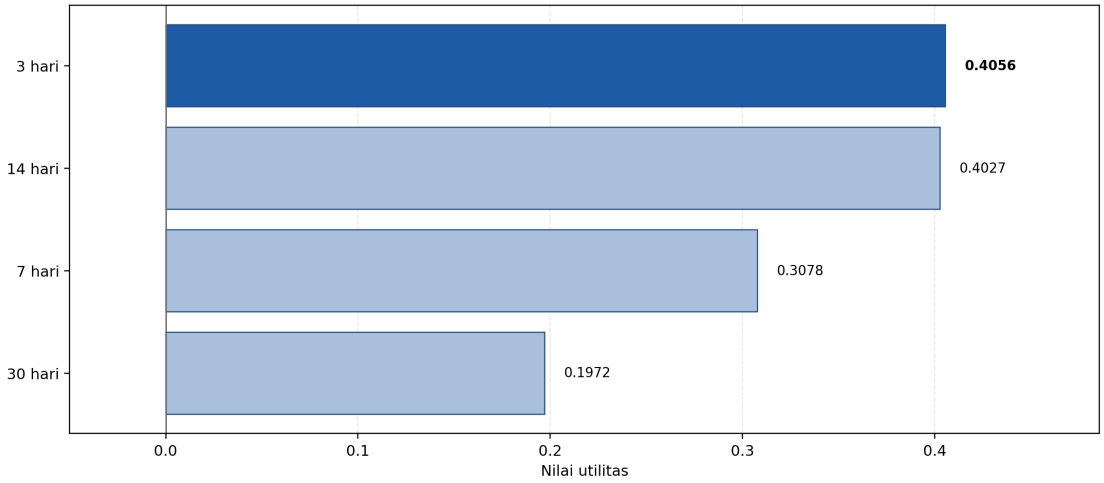
5.1.3 Skenario 1 *Offline* (1 Parameter)

Skenario 1 menggunakan ruang kandidat satu parameter. Nilai `compress_after` dijaga tetap, sedangkan *chunk interval* divariasikan menjadi 3 hari, 7 hari, 14 hari, dan 30 hari. Dengan ruang uji seperti ini, pengaruh panjang *chunk* terhadap utilitas dapat dibaca lebih langsung.

Tabel 5.3: Hasil mekanisme seleksi *offline* pada Skenario 1

Rank	Kandidat	Interval	Utilitas
1	candidate 3d	3 hari	**0,4056**
2	candidate 14d	14 hari	0,4027
3	baseline_7d	7 hari	0,3078
4	candidate 30d	30 hari	0,1972

Tabel 5.3 menunjukkan bahwa kandidat dengan utilitas tertinggi pada Skenario 1 adalah `candidate_3d` dengan nilai **0,4056**. Selisih utilitas antara `candidate_3d` dan `candidate_14d` cukup rapat, sedangkan `baseline_7d` dan `candidate_30d` berada pada urutan berikutnya.



Gambar 5.1: Grafik utilitas kandidat pada Skenario 1 (1 parameter)

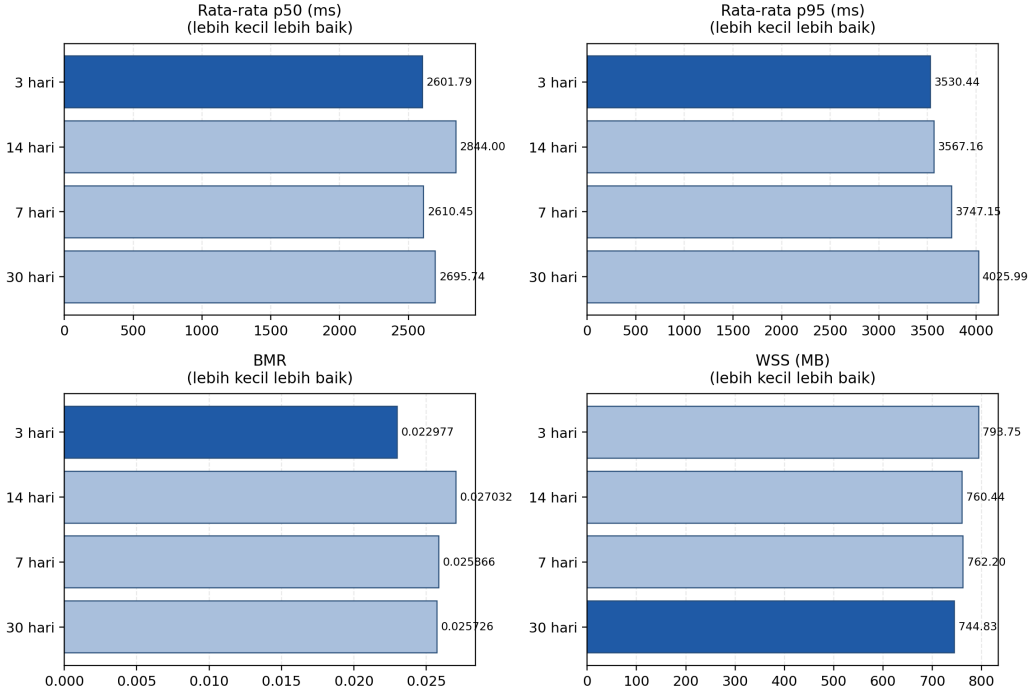
Gambar 5.1 memperlihatkan bahwa `candidate_3d` berada pada posisi tertinggi. Hasil ini memperlihatkan bahwa ruang satu parameter cenderung memberi nilai terbaik pada *chunk interval* yang lebih pendek. Dengan 191 *chunk*, konfigurasi 3 hari memberi pembagian waktu yang lebih rapat dibanding baseline 7 hari.

Tabel 5.4: Ringkasan metrik utama pada Skenario 1

Kandidat	Avg p50 (ms)	Avg p95 (ms)	Avg chunk terbaca	Avg BMR	Avg WSS (MB)	Storage (MB)	Overhead (detik)	Utilitas
candidate 3d	**2601,79**	**3530,44**	4,67	**0,022977**	793,75	962,09	302,20	**0,4056**
candidate 14d	2844,00	3567,16	1,67	0,027032	760,44	**955,59**	**267,58**	0,4027
baseline_7d	2610,45	3747,15	2,33	0,025866	762,20	964,20	297,50	0,3078
candidate 30d	2695,74	4025,99	**1,00**	0,025726	**744,83**	1022,52	299,81	0,1972

Tabel 5.4 menyajikan rata-rata *p50*, *p95*, BMR, WSS, jejak penyimpanan, dan *overhead*. Pada tahap *offline*, *overhead* dinyatakan dalam detik karena dihitung dari

waktu penerapan kandidat konfigurasi. Nilai tersebut menjadi dasar perhitungan utilitas yang telah dirumuskan pada Persamaan 4.6 dan Persamaan 4.7.



Gambar 5.2: Perbandingan p50, p95, BMR, dan WSS pada Skenario 1

Gambar 5.2 memperjelas bahwa utilitas tidak dibentuk oleh latensi saja. BMR dan WSS ikut memberi tekanan pada skor *memory-aware*, sedangkan ukuran penyimpanan dan *overhead* menjadi penalti. Oleh karena itu, `candidate_3d` dibaca sebagai konfigurasi terpilih pada ruang satu parameter, bukan sebagai konfigurasi yang otomatis unggul pada semua metrik.

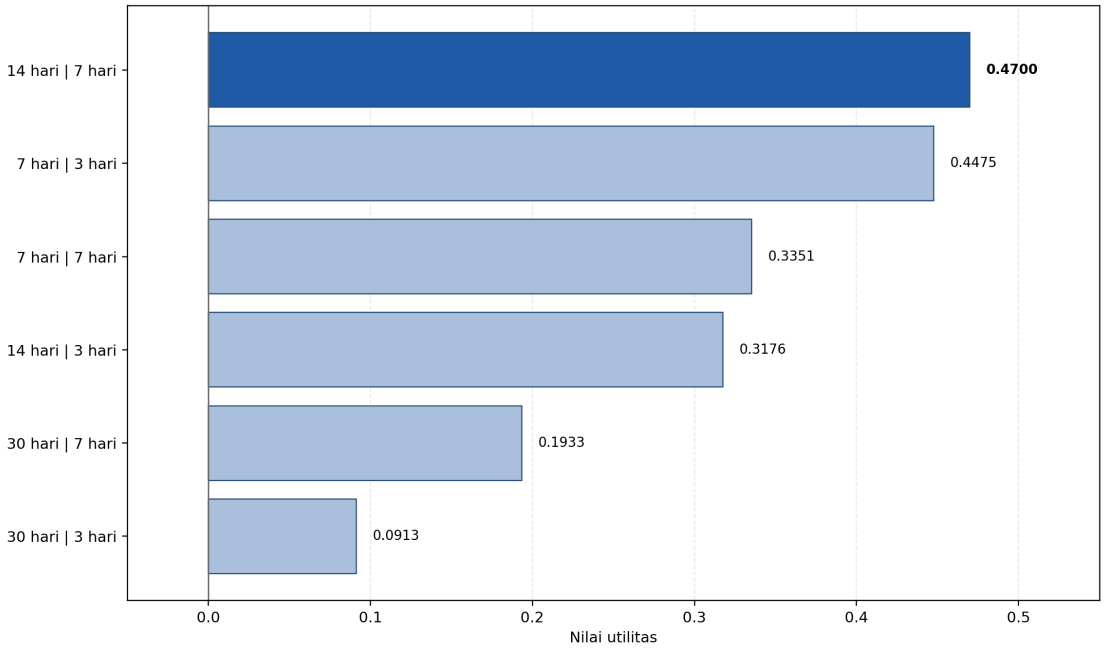
5.1.4 Skenario 2 *Offline* (2 Parameter)

Skenario 2 memperluas ruang uji dengan memasukkan `compress_after`. Enam kandidat diuji melalui kombinasi *chunk interval* 7 hari, 14 hari, dan 30 hari dengan `compress_after` 3 hari atau 7 hari. Dengan skenario ini, hasil seleksi tidak hanya membaca granularitas *chunk*, tetapi juga membaca kapan data historis mulai dikompresi.

Tabel 5.5: Hasil mekanisme seleksi *offline* pada Skenario 2

Rank	**Kandidat**	**Interval**	`compress_after`	**Utilitas**
1	candidate_14d ca_7d	14 hari	7 hari	**0,4700**
2	candidate_7d ca_3d	7 hari	3 hari	0,4475
3	candidate_7d ca_7d	7 hari	7 hari	0,3351
4	candidate_14d ca_3d	14 hari	3 hari	0,3176
5	candidate_30d ca_7d	30 hari	7 hari	0,1933
6	candidate_30d ca_3d	30 hari	3 hari	0,0913

Tabel 5.5 menunjukkan bahwa kombinasi dengan utilitas tertinggi pada Skenario 2 adalah `candidate_14d_ca_7d` dengan nilai **0,4700**. Kandidat `candidate_7d_ca_3d` berada pada urutan kedua, sedangkan empat kandidat lain berada di bawahnya.



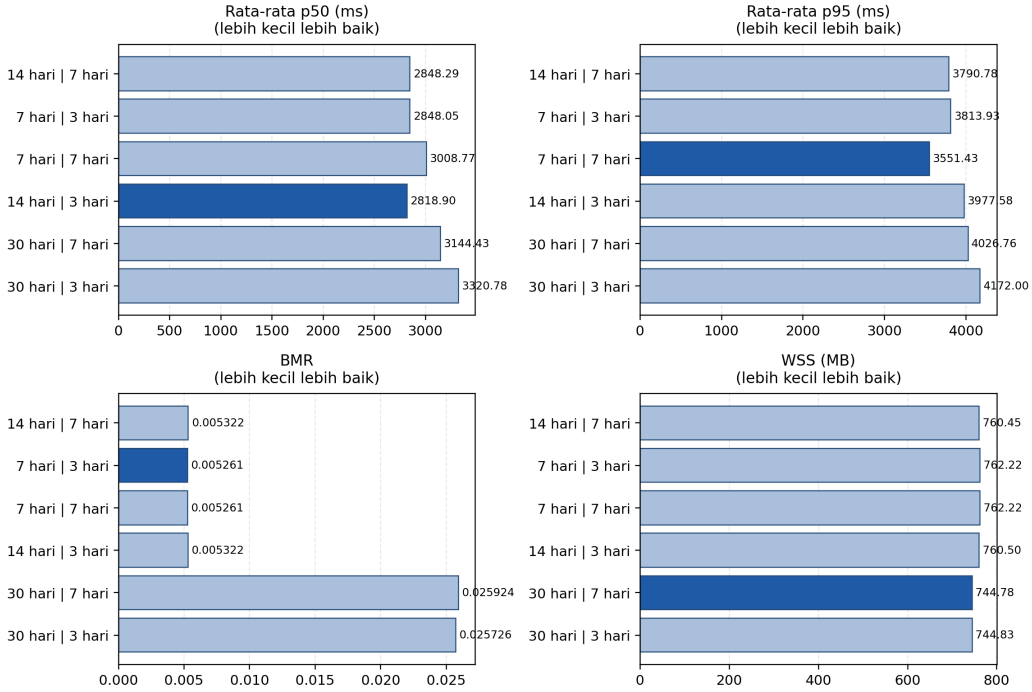
Gambar 5.3: Grafik utilitas kandidat pada Skenario 2 (2 parameter)

Gambar 5.3 menunjukkan bahwa penambahan parameter kompresi mengubah hasil seleksi. Pada Skenario 1, konfigurasi 3 hari terpilih. Pada Skenario 2, konfigurasi terpilih bergeser ke 14 hari dengan `compress_after` 7 hari. Pergeseran ini memperlihatkan bahwa panjang *chunk* dan kebijakan kompresi saling memengaruhi hasil akhir.

Tabel 5.6: Ringkasan metrik utama pada Skenario 2

Kandidat	Avg p50 (ms)	Avg p95 (ms)	Avg chunk terbaca	Avg BMR	Avg WSS (MB)	Storage (MB)	Overhead (detik)	Utilitas
candidate_14d ca_7d	2848,29	3790,78	1,67	0,005322	760,45	955,59	282,14	**0,4700**
candidate_7d ca_3d	2848,05	3813,93	2,33	**0,005261**	762,22	**942,98**	**262,93**	0,4475
candidate_7d ca_7d	3008,77	**3551,43**	2,33	**0,005261**	762,22	964,20	323,96	0,3351
candidate_14d ca_3d	**2818,90**	3977,58	1,67	0,005322	760,50	955,59	294,76	0,3176
candidate_30d ca_7d	3144,43	4026,76	**1,00**	0,025924	**744,78**	1023,81	278,74	0,1933
candidate_30d ca_3d	3320,78	4172,00	**1,00**	0,025726	744,83	1023,67	286,49	0,0913

Tabel 5.6 merangkum nilai utama untuk seluruh kandidat pada Skenario 2. Kolom *overhead* menunjukkan waktu penerapan kandidat konfigurasi dalam detik. Kandidat `candidate_14d_ca_7d` tidak dibaca sebagai pemenang karena satu metrik tunggal, tetapi karena gabungan latensi, BMR, WSS, *storage size*, dan *overhead* menghasilkan utilitas tertinggi pada ruang dua parameter.



Gambar 5.4: Perbandingan p50, p95, BMR, dan WSS pada Skenario 2

Gambar 5.4 menunjukkan perubahan pola metrik ketika kebijakan kompresi ikut diuji. Hasil ini penting karena data warehouse tidak hanya mengejar waktu respons, tetapi juga harus menjaga biaya baca blok dan penggunaan ruang kerja memori.

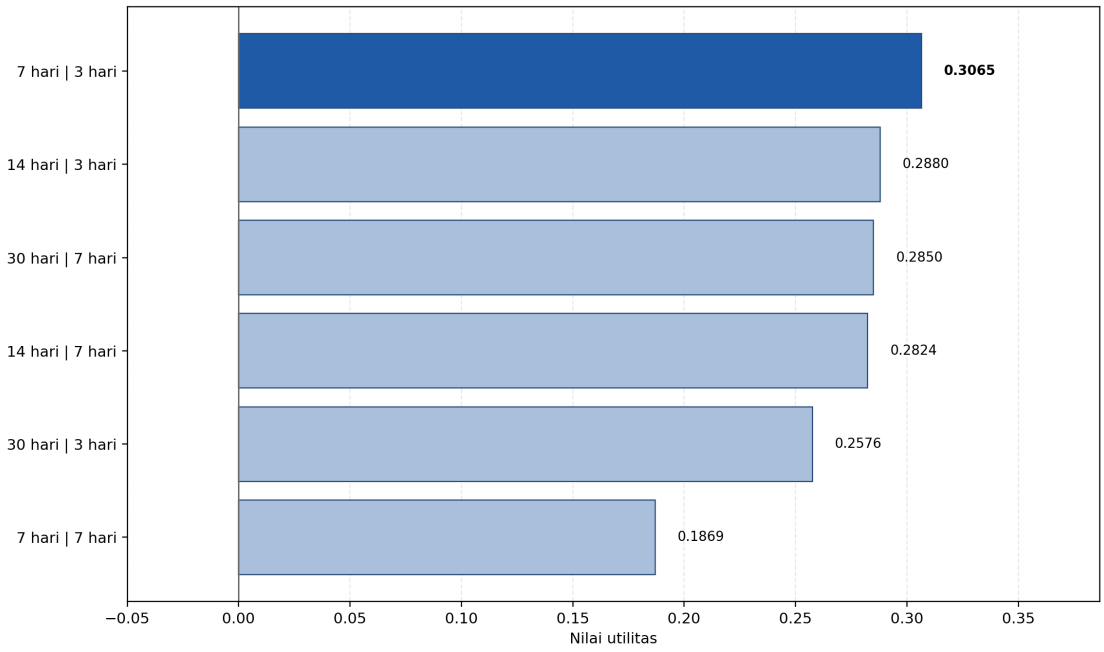
5.1.5 Skenario 3 *Offline* (2 Parameter + *Auto Workload*)

Skenario 3 menggunakan kandidat dua parameter seperti Skenario 2, tetapi kumpulan kuerinya berasal dari *auto workload* terpilih. Dasar pembentukan *auto workload* telah dijelaskan pada proses *workload clustering* di Bab IV. Dengan susunan ini, Skenario 3 membaca konfigurasi yang cocok terhadap hasil pengelompokan pola kueri.

Tabel 5.7: Hasil mekanisme seleksi *offline* pada Skenario 3

Rank	Kandidat	Interval	`compress_after`	Utilitas
1	auto_bound_7d ca_3d	7 hari	3 hari	**0,3065**
2	auto_bound_14d ca_3d	14 hari	3 hari	0,2880
3	auto_bound_30d ca_7d	30 hari	7 hari	0,2850
4	auto_bound_14d ca_7d	14 hari	7 hari	0,2824
5	auto_bound_30d ca_3d	30 hari	3 hari	0,2576
6	auto_bound_7d ca_7d	7 hari	7 hari	0,1869

Tabel 5.7 menunjukkan bahwa kandidat dengan utilitas tertinggi pada Skenario 3 adalah `auto_bound_7d_ca_3d` dengan nilai **0,3065**. Konfigurasi ini menggunakan *chunk interval* 7 hari dan `compress_after` 3 hari.



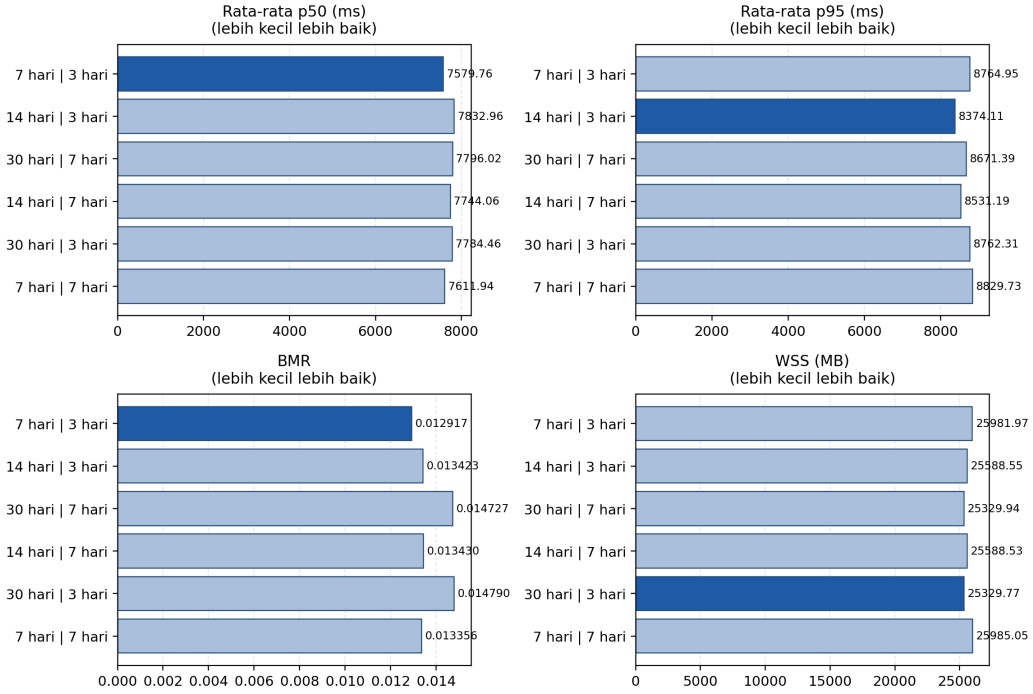
Gambar 5.5: Grafik utilitas kandidat pada Skenario 3 (2 parameter + auto workload)

Gambar 5.5 memperlihatkan bahwa konfigurasi 7 hari kembali menjadi pilihan kuat ketika dikombinasikan dengan kompresi 3 hari dan diuji pada *auto workload* terpilih. Artinya, interval 7 hari pada baseline tidak selalu menjadi masalah. Perbedaannya terletak pada cara konfigurasi tersebut dipilih dan pada kebijakan kompresi yang menyertainya.

Tabel 5.8: Ringkasan metrik utama pada Skenario 3

Kandidat	Avg p50 (ms)	Avg p95 (ms)	Avg chunk terbaca	Avg BMR	Avg WSS (MB)	Storage (MB)	Overhead (detik)	Utilitas
auto_bound_7d ca_3d	**7579,76**	8764,95	2,67	**0,012917**	25981,97	**942,98**	293,80	**0,3065**
auto_bound_14d ca_3d	7832,96	**8374,11**	1,67	0,013423	25588,55	955,59	300,60	0,2880
auto_bound_30d ca_7d	7796,02	8671,39	**1,00**	0,014727	25329,94	1022,52	286,31	0,2850
auto_bound_14d ca_7d	7744,06	8531,19	1,67	0,013430	25588,53	955,53	304,00	0,2824
auto_bound_30d ca_3d	7784,46	8762,31	**1,00**	0,014790	**25329,77**	1022,52	**274,85**	0,2576
auto_bound_7d ca_7d	7611,94	8829,73	2,67	0,013356	25985,05	964,20	332,17	0,1869

Tabel 5.8 menyajikan rata-rata *p50*, *p95*, BMR, WSS, *storage size*, dan *overhead* pada Skenario 3. Kolom *overhead* menggunakan satuan detik sesuai waktu penerapan kandidat konfigurasi pada tahap *offline*. Pada skenario ini, konfigurasi `auto_bound_7d_ca_3d` memiliki kombinasi skor terbaik di antara kandidat yang diuji.



Gambar 5.6: Perbandingan p50, p95, BMR, dan WSS pada Skenario 3

Gambar 5.6 memperlihatkan hubungan antara metrik latensi dan metrik *memory-aware*. Hasil ini menjadi dasar bahwa Skenario 3 tidak hanya membaca konfigurasi fisik, tetapi juga membaca pola kueri yang terbentuk dari *auto workload*.

5.2 Hasil Tahap *Online*

Pada bab ini, hasil tahap *online* disajikan mulai dari window kueri yang diuji, s=kenario yang dijalankan, urutan aksi yang diambil, serta pembacaan metrik latensi, BMR, WSS, dan *overhead*.

5.2.1 Window Uji

Tahap *online* dijalankan setelah konfigurasi terbaik dari tahap *offline* diperoleh. Pembanding yang digunakan adalah baseline dengan konfigurasi 7 hari / 7 hari, sedangkan konfigurasi tahap *online* berasal dari hasil eksperimen Skenario 1, Skenario 2, dan Skenario 3 tahap *offline*. Setiap konfigurasi menjalankan susunan *window* yang sama dijelaskan di bawah ini.

Tabel 5.9: Komposisi *window* pada implementasi tahap *online*

Window ke-	**Predefined query**	**Ad-hoc query**	**Interpretasi**
1	100%	0%	100% kueri berasal dari pola awal (*predefined query*).
2	75%	25%	75% kueri pola awal, 25 % Kueri baru masuk.
3	50%	50%	50% kueri pola awal, 50 % Kueri baru masuk.
4	25%	75%	25% kueri pola awal, 75 % Kueri baru masuk.
5	0%	100%	100% kueri merupakan kueri baru (*ad-hoc query*).

Tabel 5.9 menunjukkan bahwa tahap *online* dirancang sebagai pengujian *workload shift*. Persentase *predefined query* diturunkan bertahap dari 100% sampai 0%. Dengan pola ini, sistem diuji dari keadaan yang paling dekat dengan workload awal menuju keadaan yang seluruhnya berisi kueri baru.

Setiap *window* diselesaikan terlebih dahulu, kemudian sistem memilih satu aksi. Aksi yang muncul adalah *rewrite*, *none*, *tighten*, dan *escalate*. Aksi *rewrite* memperjelas predikat waktu kueri. Aksi *none* mempertahankan konfigurasi yang sedang berjalan. Aksi *tighten* mempercepat kebijakan kompresi. Aksi *escalate* menaikkan penanganan ke seleksi ulang ketika adaptasi ringan belum cukup.

5.2.2 Skenario 1

Tabel 5.10: Rekap implementasi tahap *online* pada Skenario 1

Window ke-	**Kueri awal (%)**	**Aksi**	**Latensi online (ms)**	**BMR**	**WSS (MB)**	**Penalti overhead**	**Utilitas online**
1	100%	rewrite	410,402	0,010251	4762,41	0,0000	0,4133
2	75%	none	436,052	0,009680	5021,46	1,0000	0,3810
3	50%	escalate	543,474	0,005354	6275,84	0,0000	0,4273
4	25%	escalate	605,182	0,000001	6381,75	0,4000	0,4195
5	0%	escalate	617,521	0,000000	6640,80	0,4000	0,4239

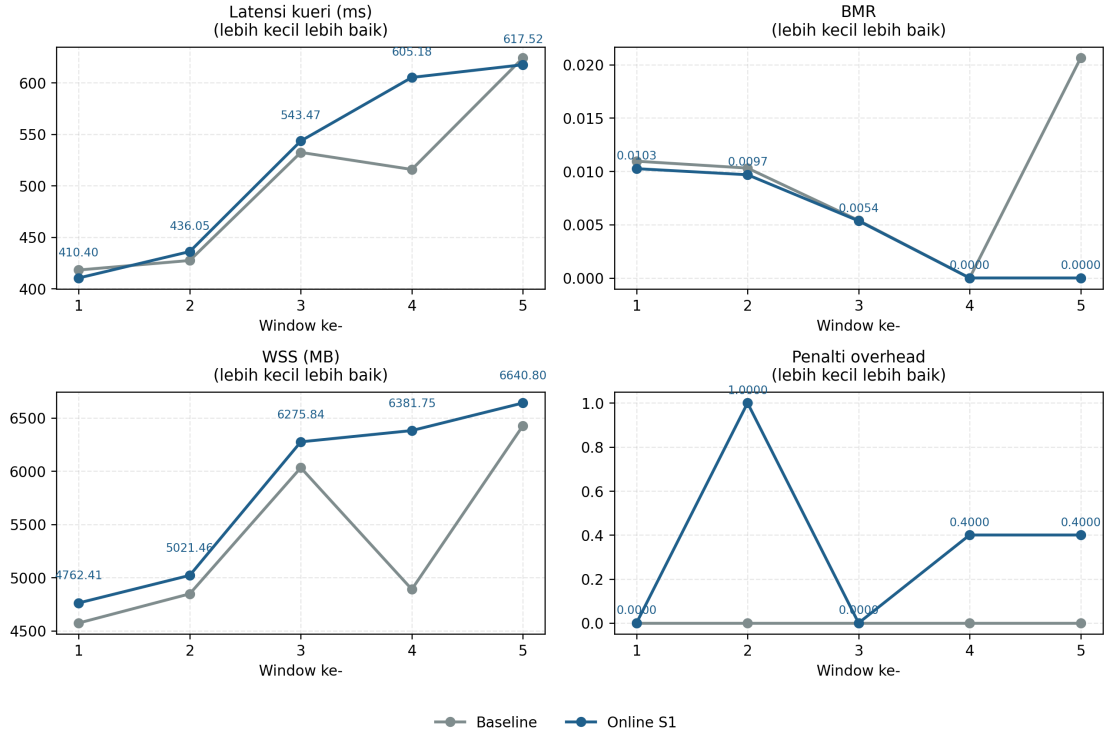
Tabel 5.10 menunjukkan lima keputusan pada Skenario 1. Aksi *rewrite* dipilih pada *window* pertama, *none* pada *window* kedua, dan *escalate* pada tiga *window* berikutnya.

Urutan ini menunjukkan bahwa konfigurasi 3 hari cukup terbantu oleh *rewrite* di awal, tetapi membutuhkan penanganan lebih besar ketika komposisi kueri mulai bergeser.

Tabel 5.11: Perbandingan empat metrik tahap *online* pada Skenario 1

Window	Kueri awal (%)	Latensi kueri (ms)		BMR		WSS (MB)		Penalti overhead	
		Baseline	Online	Baseline	Online	Baseline	Online	Baseline	Online
1	100%	418,138	410,402	0,010952	0,010251	4573,88	4762,41	0,0000	0,0000
2	75%	427,409	436,052	0,010304	0,009680	4847,93	5021,46	0,0000	1,0000
3	50%	532,372	543,474	0,005399	0,005354	6032,65	6275,84	0,0000	0,0000
4	25%	515,799	605,182	0,000000	0,000001	4890,26	6381,75	0,0000	0,4000
5	0%	624,093	617,521	0,020633	0,000000	6426,91	6640,80	0,0000	0,4000

Tabel 5.11 memperlihatkan nilai latensi, BMR, WSS, dan *overhead* pada setiap *window*. Nilai tersebut dibandingkan langsung dengan baseline pada komposisi kueri yang sama.



Gambar 5.7: Perbandingan empat metrik tahap *online* pada Skenario 1

Gambar 5.7 menunjukkan bahwa Skenario 1 menghasilkan BMR yang lebih rendah daripada baseline pada seluruh *window*. Namun, latensi hanya lebih rendah pada *window* pertama dan kelima. Tiga *window* lainnya masih menghasilkan latensi lebih tinggi daripada baseline. Hasil ini penting karena memperlihatkan bahwa *chunk* kecil dapat menekan rasio pembacaan dari *storage*, tetapi belum tentu menurunkan waktu eksekusi rata-rata pada semua komposisi kueri.

5.2.3 Skenario 2

Tabel 5.12: Rekap implementasi tahap *online* pada Skenario 2

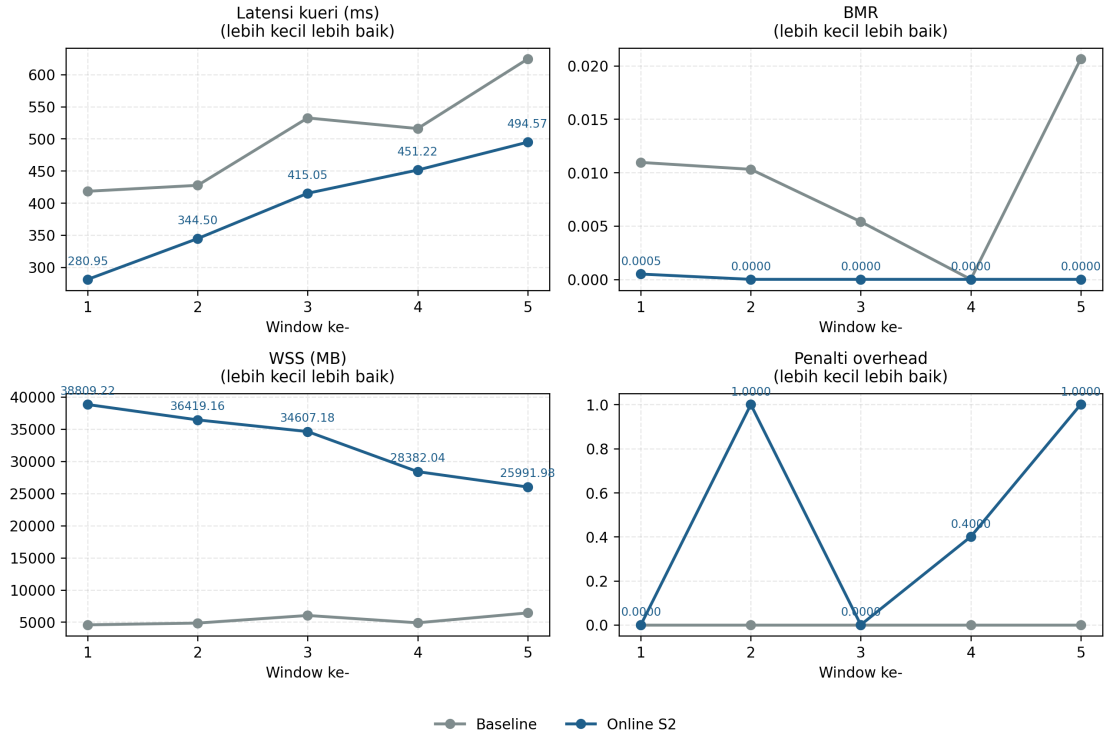
Window ke-	Kueri awal (%)	Aksi	Latensi online (ms)	BMR	WSS (MB)	Penalti overhead	Utilitas online
1	100%	rewrite	280,952	0,000496	38809,22	0,0000	0,5660
2	75%	none	344,495	0,000000	36419,16	1,0000	0,5356
3	50%	escalate	415,046	0,000000	34607,18	0,0000	0,4921
4	25%	tighten	451,216	0,000000	28382,04	0,4000	0,4765
5	0%	escalate	494,572	0,000000	25991,98	1,0000	0,4568

Tabel 5.12 menunjukkan urutan aksi *rewrite*, *none*, *escalate*, *tighten*, dan *escalate*. Perubahan `compress_after` diterapkan setelah aksi *tighten* pada *window* keempat. Urutan ini memperlihatkan bahwa konfigurasi 14 hari / 7 hari tidak hanya mengandalkan hasil offline, tetapi juga menyesuaikan kebijakan kompresi ketika tekanan kueri berubah.

Tabel 5.13: Perbandingan empat metrik tahap *online* pada Skenario 2

Window	Kueri awal (%)	Latensi kueri (ms)		BMR		WSS (MB)		Penalti overhead	
		Baseline	Online	Baseline	Online	Baseline	Online	Baseline	Online
1	100%	418,138	280,952	0,010952	0,000496	4573,88	38809,22	0,0000	0,0000
2	75%	427,409	344,495	0,010304	0,000000	4847,93	36419,16	0,0000	1,0000
3	50%	532,372	415,046	0,005399	0,000000	6032,65	34607,18	0,0000	0,0000
4	25%	515,799	451,216	0,000000	0,000000	4890,26	28382,04	0,0000	0,4000
5	0%	624,093	494,572	0,020633	0,000000	6426,91	25991,98	0,0000	1,0000

Tabel 5.13 menunjukkan bahwa latensi dan BMR Skenario 2 lebih rendah daripada baseline pada seluruh *window*, sedangkan WSS yang dihasilkan lebih besar.



Gambar 5.8: Perbandingan empat metrik tahap *online* pada Skenario 2

Gambar 5.8 memperlihatkan hasil yang lebih stabil dibanding Skenario 1. Kombinasi *chunk interval* 14 hari dan `compress_after` 7 hari menghasilkan latensi dan BMR yang lebih rendah daripada baseline pada lima *window*. WSS yang lebih besar menunjukkan bahwa penurunan waktu eksekusi disertai peningkatan volume blok yang bekerja selama kueri dijalankan.

5.2.4 Skenario 3

Tabel 5.14: Rekap implementasi tahap *online* pada Skenario 3

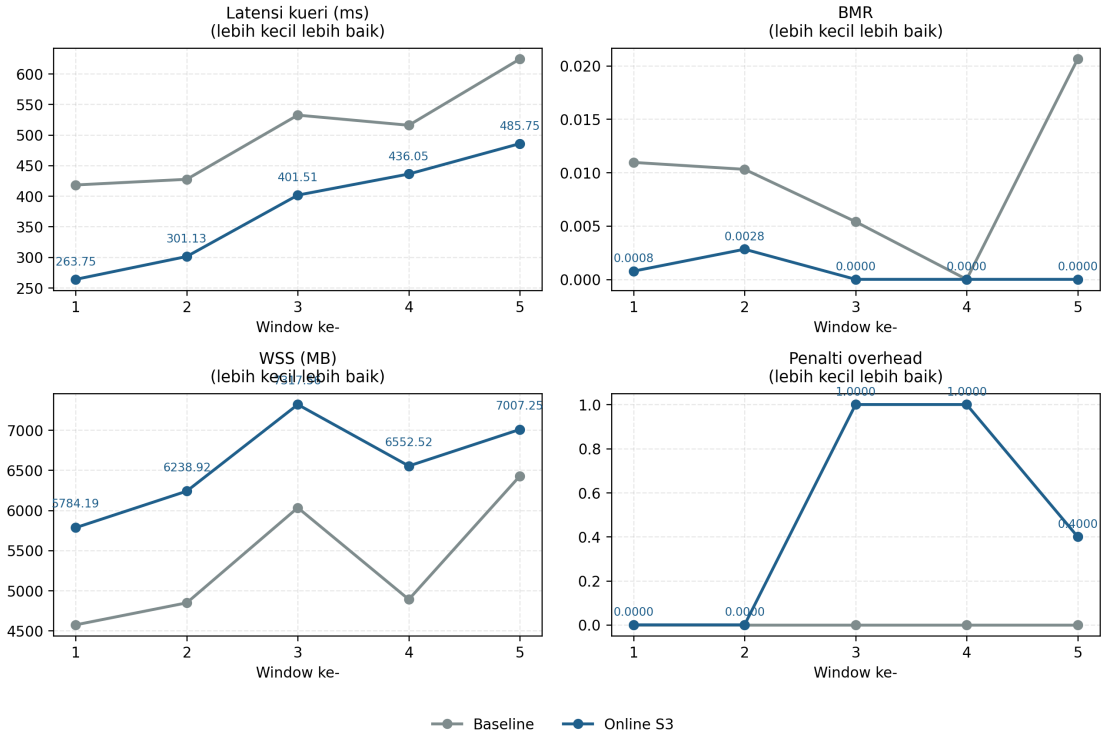
Window ke-	Kueri awal (%)	Aksi	Latensi online (ms)	BMR	WSS (MB)	Penalti overhead	Utilitas online
1	100%	none	263,752	0,000769	5784,19	0,0000	0,1771
2	75%	rewrite	301,134	0,002821	6238,92	0,0000	0,1673
3	50%	escalate	401,513	0,000000	7317,56	1,0000	0,0585
4	25%	escalate	436,054	0,000000	6552,52	1,0000	0,0577
5	0%	escalate	485,751	0,000000	7007,25	0,4000	0,0781

Tabel 5.14 menunjukkan aksi *none* pada *window* pertama, *rewrite* pada *window* kedua, dan *escalate* pada tiga *window* berikutnya. Pada konfigurasi ini, sistem mempertahankan pengaturan awal pada *window* pertama, kemudian mulai memperbaiki bentuk kueri dan menaikkan penanganan saat komposisi ad-hoc semakin besar.

Tabel 5.15: Perbandingan empat metrik tahap *online* pada Skenario 3

Window	Kueri awal (%)	Latensi kueri (ms)		BMR		WSS (MB)		Penalti overhead	
		Baseline	Online	Baseline	Online	Baseline	Online	Baseline	Online
1	100%	418,138	263,752	0,010952	0,000769	4573,88	5784,19	0,0000	0,0000
2	75%	427,409	301,134	0,010304	0,002821	4847,93	6238,92	0,0000	0,0000
3	50%	532,372	401,513	0,005399	0,000000	6032,65	7317,56	0,0000	1,0000
4	25%	515,799	436,054	0,000000	0,000000	4890,26	6552,52	0,0000	1,0000
5	0%	624,093	485,751	0,020633	0,000000	6426,91	7007,25	0,0000	0,4000

Tabel 5.15 menunjukkan bahwa latensi dan BMR Skenario 3 lebih rendah daripada baseline pada seluruh *window*. WSS Skenario 3 berada di atas baseline pada lima komposisi kueri yang diuji.



Gambar 5.9: Perbandingan empat metrik tahap *online* pada Skenario 3

Gambar 5.9 menunjukkan bahwa konfigurasi `auto_bound_7d_ca_3d` memberi hasil online paling kuat dari sisi latensi rata-rata. Konfigurasi ini memakai interval 7 hari seperti baseline, tetapi mempercepat kompresi menjadi 3 hari dan dipilih melalui *auto workload* terpilih. Hasil tersebut menjelaskan mengapa konfigurasi 7 hari dapat tetap kuat ketika tidak digunakan secara statis.

5.3 Pembahasan

Bagian ini membahas hasil seleksi konfigurasi tahap *offline*, kinerja konfigurasi terpilih pada tahap *online*, hubungan antara BMR dan WSS, serta analisis kompleksitas yang dijalankan oleh sistem.

5.3.1 Konfigurasi Terbaik

Bagian ini menyajikan analisis lebih lanjut terhadap hasil tahap *offline* guna menentukan konfigurasi skenario terpilih dengan nilai utilitas terbaik. Tahap *offline* menghasilkan satu konfigurasi terpilih pada setiap skenario. Hasil tersebut dirangkum pada Tabel 5.16.

Tabel 5.16: Ringkasan hasil terbaik tiap skenario pada tahap *offline* dan baseline

Skenario	**Kandidat terpilih**	**Parameter**	**Utilitas**
Skenario 1	`candidate_3d`	3 hari	**0,4056**
Skenario 2	`candidate_14d_ca_7d`	14 hari / 7 hari	**0,4700**
Skenario 3	`auto_bound_7d_ca_3d`	7 hari / 3 hari	**0,3065**
Baseline	`baseline`	7 hari / 7 hari	**0,3078**

Pada rangkuman dari Tabel 5.16, Skenario 1 memilih `candidate_3d` dengan utilitas 0,4056. Skenario 2 memilih `candidate_14d_ca_7d` dengan utilitas 0,4700, sedangkan Skenario 3 memilih `auto_bound_7d_ca_3d` dengan utilitas 0,3065.

Konfigurasi dengan utilitas tertinggi dipilih berdasarkan metrik gabungan dari latensi, efektivitas *chunk pruning*, BMR, WSS, ukuran penyimpanan, dan *overhead*. Nilai utilitas digunakan sebagai skor seleksi internal untuk menyusun peringkat kandidat pada masing-masing skenario. Kinerja akhir konfigurasi selanjutnya dibaca melalui hasil pengujian tahap *online*.

Skenario 1 memvariasikan *chunk interval* sebesar 3, 7, 14, dan 30 hari dengan kebijakan kompresi yang tetap. Hasil seleksi menemukan konfigurasi terbaik dengan interval 3 hari pada urutan pertama. Skenario 2 menggunakan kombinasi parameter *chunk interval* dan `compress_after`, sehingga konfigurasi terpilih menjadi interval 14 hari dengan `compress_after` 7 hari. Skenario 3 menggunakan ruang parameter dua dimensi pada kumpulan *workload* hasil pengelompokan. Konfigurasi terpilih menggunakan interval 7 hari dan `compress_after` 3 hari.

Baseline menggunakan interval 7 hari dan `compress_after` 7 hari, mendapatkan utilitas 0,3078. Hasil ini menunjukkan bahwa konfigurasi terpilih pada Skenario 2 memiliki utilitas lebih tinggi daripada baseline, sedangkan konfigurasi terpilih pada Skenario 1 dan Skenario 3 memiliki utilitas lebih rendah daripada baseline.

Hasil ini menerapkan prinsip tahap *offline* dan *online* pada PROADAPT. Tahap *offline* memilih konfigurasi awal dari ruang kandidat yang telah ditetapkan, kemudian konfigurasi terpilih digunakan pada tahap *online* untuk menghadapi perubahan komposisi kueri.

5.3.2 Kinerja *Offline*

Kinerja tahap *offline* dibaca dengan membandingkan konfigurasi terpilih pada setiap skenario terhadap konfigurasi 7 hari/7 hari. Konfigurasi tersebut digunakan sebagai baseline statis dengan 82 *chunk*. Perubahan setiap metrik dihitung terhadap hasil baseline 7 hari/7 hari pada ruang uji yang sama. Ringkasan perbandingannya disajikan pada Tabel 5.17.

Tabel 5.17: Perubahan kinerja konfigurasi terpilih terhadap baseline pada tahap *offline*

Skenario	***Chunk interval*** dan `compress_after`	**Δp50**	**Δp95**	**Δ*chunk* terbaca**	**ΔBMR**	**ΔWSS**	**Δ*storage***	**Δ*overhead***
Skenario 1	3 hari dan 7 hari	-0,33%	-5,78%	+100,43%	-11,17%	+4,14%	-0,22%	+1,58%
Skenario 2	14 hari dan 7 hari	-5,33%	+6,74%	-28,33%	+1,16%	-0,23%	-0,89%	-12,91%
Skenario 3	7 hari dan 3 hari	-0,42%	-0,73%	0,00%	-3,29%	-0,01%	-2,20%	-11,55%

Baseline menggunakan *chunk interval* 7 hari dan `compress_after` 7 hari. Konfigurasi ini membentuk 82 *chunk* dan menjadi acuan untuk membaca perubahan kinerja dari ketiga konfigurasi terpilih.

Pada Skenario 1, konfigurasi 3 hari/7 hari menghasilkan rata-rata *p50* sebesar 2.601,79 ms dan rata-rata *p95* sebesar 3.530,44 ms. Baseline pada pengujian yang sama menghasilkan *p50* sebesar 2.610,45 ms dan *p95* sebesar 3.747,15 ms. Konfigurasi Skenario 1 menurunkan *p50* sebesar 0,33% dan *p95* sebesar 5,78%.

BMR Skenario 1 turun dari 0,025866 menjadi 0,022977. Pada saat yang sama, rata-rata jumlah *chunk* yang terbaca meningkat dari 2,33 menjadi 4,67 dan WSS meningkat dari 762,20 MB menjadi 793,75 MB. Interval 3 hari membagi data ke dalam 191 *chunk*, sehingga batas partisinya lebih rapat daripada baseline. Ukuran penyimpanan turun tipis dari 964,20 MB menjadi 962,09 MB, sedangkan *overhead* adaptasi meningkat dari 297,50 detik menjadi 302,20 detik.

Hasil Skenario 1 menunjukkan penurunan *p50*, *p95*, dan BMR. Perbaikan tersebut disertai peningkatan jumlah *chunk* yang terbaca, WSS, dan *overhead* adaptasi. Gabungan seluruh komponen menghasilkan utilitas 0,4056 dan menempatkan `candidate_3d` sebagai konfigurasi terpilih pada ruang satu parameter.

Pada Skenario 2, konfigurasi 14 hari/7 hari menghasilkan rata-rata *p50* sebesar 2.848,29 ms. Baseline 7 hari/7 hari pada ruang uji dua parameter menghasilkan *p50* sebesar 3.008,77 ms. Nilai tersebut menunjukkan penurunan *p50* sebesar 5,33%. Rata-rata jumlah *chunk* yang terbaca juga turun dari 2,33 menjadi 1,67.

Rata-rata *p95* Skenario 2 mencapai 3.790,78 ms, sedangkan baseline menghasilkan 3.551,43 ms. BMR berubah dari 0,005261 menjadi 0,005322. Selisih BMR kedua konfigurasi sebesar 1,16%, sementara WSS turun dari 762,22 MB menjadi 760,45 MB.

Konfigurasi 14 hari/7 hari juga menurunkan ukuran penyimpanan dari 964,20 MB menjadi 955,59 MB. *Overhead* adaptasi turun dari 323,96 detik menjadi 282,14 detik atau

sebesar 12,91%. Penurunan *p50*, jumlah *chunk* terbaca, WSS, ukuran penyimpanan, dan *overhead* adaptasi menghasilkan utilitas 0,4700 bagi `candidate_14d_ca_7d`.

Pada Skenario 3, konfigurasi 7 hari/3 hari menggunakan *chunk interval* yang sama dengan baseline dan mempercepat `compress_after` dari 7 hari menjadi 3 hari. Skenario ini dijalankan pada *auto workload* hasil pengelompokan kueri.

Konfigurasi Skenario 3 menghasilkan rata-rata *p50* sebesar 7.579,76 ms dan *p95* sebesar 8.764,95 ms. Baseline pada *auto workload* yang sama menghasilkan *p50* sebesar 7.611,94 ms dan *p95* sebesar 8.829,73 ms. Nilai tersebut menunjukkan penurunan *p50* sebesar 0,42% dan *p95* sebesar 0,73%.

BMR turun dari 0,013356 menjadi 0,012917. Kedua konfigurasi membaca rata-rata 2,67 *chunk* dan menghasilkan WSS yang hampir sama, yaitu 25.985,05 MB pada baseline dan 25.981,97 MB pada Skenario 3. *Storage* turun dari 964,20 MB menjadi 942,98 MB, sedangkan *overhead* adaptasi turun dari 332,17 detik menjadi 293,80 detik.

Hasil Skenario 3 menunjukkan pengaruh perubahan kebijakan kompresi pada *chunk interval* yang sama. Konfigurasi 7 hari/3 hari menghasilkan latensi, BMR, *storage*, dan *overhead* adaptasi yang lebih rendah daripada baseline 7 hari/7 hari. Gabungan hasil tersebut menghasilkan utilitas 0,3065 bagi `auto_bound_7d_ca_3d`.

Secara keseluruhan, Skenario 1 menghasilkan penurunan *p95* dan BMR paling besar pada tahap *offline*. Skenario 2 menghasilkan penurunan *p50*, jumlah *chunk* terbaca, dan *overhead* adaptasi paling besar. Skenario 3 menghasilkan perbaikan pada *p50*, *p95*, BMR, *storage*, dan *overhead* adaptasi dengan jumlah *chunk* terbaca yang sama seperti baseline.

5.3.3 Kinerja *Online*

Tahap *online* membandingkan Skenario 1, Skenario 2, Skenario 3, dan baseline pada lima komposisi kueri yang sama. Persentase *predefined query* diturunkan dari 100% menjadi 0%, sedangkan persentase *ad-hoc query* dinaikkan dari 0% menjadi 100%. Rata-rata hasil seluruh *window* disajikan pada Tabel 5.18.

Tabel 5.18: Perbandingan kinerja Skenario 1, Skenario 2, Skenario 3, dan baseline pada tahap *online*

Skenario	***Chunk interval*** dan `compress_after`	**Latensi (ms)**	**Δlatensi**	**BMR**	**ΔBMR**	**WSS (MB)**	**ΔWSS**	**Penalti *overhead***
Skenario 1	3 hari dan 7 hari	522,526	+3,77%	0,005057	-46,53%	5.816,45	+8,63%	0,3600
Skenario 2	14 hari dan 7 hari	397,256	-21,11%	0,000099	-98,95%	32.841,91	+513,37%	0,4800
Skenario 3	7 hari dan 3 hari	377,641	-25,01%	0,000718	-92,41%	6.580,09	+22,89%	0,4800
Baseline	7 hari dan 7 hari	503,562	–	0,009458	–	5.354,32	–	0,0000

Baseline menghasilkan rata-rata latensi 503,562 ms, BMR 0,009458, dan WSS 5.354,32 MB. Konfigurasi baseline dipertahankan selama lima *window* dan tidak menjalankan pemilihan aksi adaptif. Nilai tersebut digunakan sebagai acuan untuk membaca kinerja ketiga skenario.

Skenario 1 menggunakan konfigurasi awal 3 hari/7 hari dan menghasilkan rata-rata latensi 522,526 ms. Nilai tersebut naik 3,77% dari baseline. Rata-rata BMR turun 46,53% menjadi 0,005057, sedangkan WSS meningkat 8,63% menjadi 5.816,45 MB. Penalti *overhead* rata-ratanya sebesar 0,3600.

Pada perbandingan setiap *window*, latensi Skenario 1 lebih rendah daripada baseline pada *window* pertama dan kelima. Latensi pada *window* kedua, ketiga, dan keempat berada di atas baseline. BMR lebih rendah pada *window* pertama, kedua, ketiga, dan kelima. Pada *window* keempat, baseline menghasilkan BMR 0,000000, sedangkan Skenario 1 menghasilkan 0,000001.

Aksi Skenario 1 terdiri atas *rewrite* pada *window* pertama, *none* pada *window* kedua, dan *escalate* pada tiga *window* berikutnya. Hasil ini menunjukkan bahwa konfigurasi 3 hari/7 hari menurunkan BMR, sementara rata-rata latensinya berada di atas baseline ketika komposisi kueri berubah.

Skenario 2 menggunakan konfigurasi awal 14 hari/7 hari dan menghasilkan rata-rata latensi 397,256 ms. Nilai tersebut turun 21,11% dari baseline. Latensi Skenario 2 lebih rendah pada seluruh *window*. Rata-rata BMR turun 98,95% menjadi 0,000099. WSS meningkat dari 5.354,32 MB menjadi 32.841,91 MB, sedangkan penalti *overhead* rata-ratanya sebesar 0,4800.

BMR Skenario 2 lebih rendah pada *window* pertama, kedua, ketiga, dan kelima. Pada *window* keempat, baseline dan Skenario 2 menghasilkan BMR 0,000000. WSS lebih besar daripada baseline pada seluruh *window*, dengan nilai tertinggi sebesar 38.809,22 MB pada komposisi 100% *predefined query*.

Urutan aksi Skenario 2 adalah *rewrite*, *none*, *escalate*, *tighten*, dan *escalate*. Aksi *tighten* dipilih setelah *window* keempat dan mengubah `compress_after` dari 7 hari menjadi 3 hari untuk *window* kelima. Kombinasi konfigurasi dan aksi tersebut menghasilkan penurunan latensi pada seluruh komposisi kueri.

Skenario 3 menggunakan konfigurasi 7 hari/3 hari dan menghasilkan rata-rata latensi 377,641 ms. Nilai tersebut turun 25,01% dari baseline dan menjadi latensi rata-rata terendah pada tahap *online*. Rata-rata BMR turun 92,41% menjadi 0,000718, sedangkan WSS meningkat 22,89% menjadi 6.580,09 MB. Penalti *overhead* rata-ratanya sebesar 0,4800.

Latensi Skenario 3 lebih rendah daripada baseline pada seluruh *window*. BMR lebih rendah pada *window* pertama, kedua, ketiga, dan kelima. Pada *window* keempat, kedua skenario menghasilkan BMR 0,000000. WSS Skenario 3 lebih besar pada seluruh *window*, tetapi kenaikannya lebih kecil dibandingkan Skenario 2.

Aksi Skenario 3 terdiri atas *none* pada *window* pertama, *rewrite* pada *window* kedua, dan *escalate* pada tiga *window* berikutnya. Konfigurasi ini menggunakan *chunk interval* yang sama dengan baseline dan `compress_after` yang lebih cepat. Hasilnya menunjukkan bahwa perubahan kebijakan kompresi dan pemilihan konfigurasi dari *auto*

workload menghasilkan latensi serta BMR yang lebih rendah daripada baseline.

Perbandingan keempat skenario menunjukkan pola yang berbeda. Skenario 1 menghasilkan BMR lebih rendah dengan latensi rata-rata yang lebih tinggi. Skenario 2 menghasilkan BMR terendah dan penurunan latensi sebesar 21,11%, disertai WSS paling besar. Skenario 3 menghasilkan latensi terendah dan penurunan BMR sebesar 92,41%, dengan kenaikan WSS yang lebih kecil daripada Skenario 2.

Rata-rata latensi ketiga skenario adalah 432,474 ms atau turun 14,12% dari baseline sebesar 503,562 ms. Rata-rata BMR ketiga skenario adalah 0,001958 atau turun 79,30% dari baseline sebesar 0,009458. Hasil tersebut menunjukkan bahwa Skenario 2 dan Skenario 3 memberikan penurunan latensi dan BMR, sedangkan Skenario 1 memberikan penurunan BMR.

5.3.4 *Trade-off Memory*

BMR dan WSS keduanya merupakan metrik yang digunakan untuk mengevaluasi sisi *memory-aware* yaitu memori (RAM) dan juga penyimpanan, namun keduanya memiliki interpretasi yang berbeda. BMR menunjukkan proporsi blok data yang dibaca oleh disk namun tidak tersedia pada*shared buffers*. Semakin banyak data yang dibaca oleh *disk*, maka biaya I/O akan semakin tinggi Nilainya dihitung dengan

$$\text{BMR} = \frac{\texttt{shared\_blks\_read}}{\texttt{shared\_blks\_hit} + \texttt{shared\_blks\_read}}.$$

WSS menunjukkan volume blok data yang dibaca selama kueri berjalan. Semakin besar WSS, maka semakin banyak blok data yang diakses selama kueri berjalan berpotensi terjadi nya *page faults*, yaitu ketika halaman memori yang dibutuhkan tidak tersedia di dalam *shared buffers* sehingga harus dibaca dari *disk* yang menyebabkan lonjakan I/O. Dengan ukuran blok PostgreSQL sebesar 8 KB, WSS dihitung dengan

$$\text{WSS} = (\texttt{shared\_blks\_hit} + \texttt{shared\_blks\_read}) \times 8 \text{ KB}.$$

Baseline menghasilkan rata-rata BMR 0,009458 dengan WSS 5.354,32 MB. Skenario 2 menghasilkan BMR 0,000099 dengan WSS 32.841,91 MB. Hasil tersebut menunjukkan bahwa Skenario 2 mengakses blok dalam volume kumulatif yang lebih besar, tetapi proporsi blok yang dibaca oleh *disk* jauh lebih kecil.

Nilai `shared_blks_read` menunjukkan blok yang perlu dibaca ke *shared buffers*. Pembacaan tersebut juga dapat dilakukan oleh *cache* sistem operasi. Oleh karena itu, pola BMR pada penelitian ini menggambarkan perilaku akses terhadap *shared buffers*.

WSS merupakan ukuran akses blok data secara kumulatif. Akses berulang terhadap blok yang sama ikut tercatat kembali dalam perhitungan. Kondisi tersebut menjelaskan nilai WSS Skenario 2 sebesar 32.841,91 MB yang melebihi kapasitas RAM lingkungan eksperimen sebesar 4 GB.

Skenario 3 menghasilkan pola yang lebih dekat dengan baseline dari sisi WSS. WSS meningkat dari 5.354,32 MB menjadi 6.580,09 MB, sedangkan BMR turun dari 0,009458 menjadi 0,000718. Pada saat yang sama, latensi turun menjadi 377,641 ms. Konfigurasi ini menghasilkan penurunan latensi dan BMR dengan kenaikan WSS yang lebih kecil dibanding Skenario 2.

Penalti *overhead* juga menjadi bagian dari *trade-off*. Baseline memiliki penalti 0,0000 karena tidak menjalankan pemilihan aksi adaptif. Skenario 1 menghasilkan penalti rata-rata 0,3600, sedangkan Skenario 2 dan Skenario 3 menghasilkan 0,4800. Pada Skenario 2 dan Skenario 3, biaya adaptasi berjalan bersama penurunan latensi rata-rata. Pada Skenario 1, biaya adaptasi berjalan bersama penurunan BMR, sementara latensi rata-ratanya berada di atas baseline.

Aspek *memory-aware* pada penelitian ini diperoleh dengan membaca BMR dan WSS secara bersamaan. BMR menggambarkan proporsi akses yang dilayani di luar *shared buffers* atau dilayani oleh *disk*, sedangkan WSS menggambarkan volume blok yang terlibat. Gabungan keduanya memperlihatkan efisiensi layanan buffer dan luas akses blok pada setiap konfigurasi.

5.3.5 Analisis Kompleksitas

Analisis kompleksitas pada Bab IV menjelaskan pertumbuhan biaya pemindaian data, pengelompokan *workload*, seleksi konfigurasi tahap *offline*, dan pengambilan keputusan tahap *online*. Pada bagian ini, bentuk kompleksitas tersebut dihubungkan dengan jumlah proses yang dijalankan selama eksperimen.

Pemindaian data. Pemindaian penuh terhadap tabel fakta membaca blok data tanpa memanfaatkan pemangkasan *chunk*. Berdasarkan Persamaan 4.15, kompleksitas proses tersebut ditulis kembali sebagai

$$T_{scan} = O(B),$$

dengan B menyatakan jumlah blok data yang dapat dibaca selama pemindaian. Ketika *chunk pruning* berjalan, sistem memeriksa metadata *chunk* dan membaca blok yang sesuai dengan rentang waktu kueri. Berdasarkan Persamaan 4.16, kompleksitasnya adalah

$$T_{prune} = O(C + B_r),$$

dengan C menyatakan jumlah *chunk* yang diperiksa untuk suatu kueri dan B_r menyatakan jumlah blok relevan yang dibaca.

Perubahan *chunk interval* menghasilkan jumlah *chunk* total yang berbeda. Perbandingan terhadap konfigurasi baseline 7 hari disajikan pada Tabel 5.19.

Tabel 5.19: Perbandingan jumlah *chunk* terhadap baseline

Chunk interval	**Jumlah *chunk***	**Rasio terhadap baseline**
3 hari	191	2,33
7 hari	82	1,00
14 hari	41	0,50
30 hari	20	0,24

Interval 3 hari membentuk jumlah *chunk* sekitar 2,33 kali baseline. Interval 14 hari dan 30 hari membentuk jumlah *chunk* sekitar 0,50 dan 0,24 kali baseline. Jumlah tersebut menunjukkan susunan fisik seluruh *hypertable*, sedangkan nilai C menunjukkan jumlah *chunk* yang diperiksa untuk satu kueri. Nilai C mengikuti rentang waktu kueri, susunan partisi, dan hasil *chunk pruning*.

Dampak susunan tersebut terlihat pada jumlah *chunk* terbaca, latensi, BMR, dan WSS. Konfigurasi terpilih Skenario 1 membaca rata-rata 4,67 *chunk*, Skenario 2 membaca 1,67 *chunk*, dan Skenario 3 membaca 2,67 *chunk*. Hasil tersebut menunjukkan hubungan antara bentuk partisi fisik dan jumlah bagian data yang terlibat dalam eksekusi kueri.

Pengelompokan *workload*. Sebelum evaluasi Skenario 3, kueri pada *query log* dikelompokkan berdasarkan karakteristik *workload*. Berdasarkan Persamaan 4.19, kompleksitas proses *workload clustering* ditulis sebagai

$$T_{cluster} = O(n \cdot k \cdot i \cdot d),$$

dengan n menyatakan jumlah kueri pada *query log*, k menyatakan jumlah *cluster*, i menyatakan jumlah iterasi, dan d menyatakan jumlah fitur kueri. Biaya pengelompokan bertambah mengikuti jumlah kueri dan jumlah iterasi K-Means. Hasil proses tersebut digunakan untuk membentuk *auto workload* pada Skenario 3.

Tahap *offline*. Setiap kandidat tahap *offline* dijalankan pada seluruh *workload*, kemudian hasil pengukurannya digunakan untuk menghitung metrik dan utilitas. Berdasarkan Persamaan 4.17, kompleksitas tahap tersebut adalah

$$T_{offline} = O(mwqrT_{query}) + O(mwqp)\,,$$

dengan m sebagai jumlah evaluasi kandidat, w sebagai jumlah *workload*, q sebagai jumlah kueri uji pada setiap *workload*, r sebagai jumlah eksekusi, dan p sebagai jumlah komponen yang dihitung pada fungsi utilitas.

Pada eksperimen ini, setiap *workload* diwakili oleh satu kueri uji sehingga $q = 1$. Setiap kueri menjalankan satu *warm-up* dan lima pengukuran utama sehingga jumlah eksekusinya adalah $r = 6$. Instansiasi tahap *offline* disajikan pada Tabel 5.20.

Tabel 5.20: Jumlah eksekusi tahap *offline*

Skenario	**Evaluasi kandidat**	**Jumlah *workload***	**Kueri per *workload***	***Warm-up* per *workload***	**Pengukuran per *workload***	**Total *warm-up***	**Total pengukuran**
Skenario 1	4	3	1	1	5	12	60
Skenario 2	6	3	1	1	5	18	90
Skenario 3	6	3	1	1	5	18	90
Total	**16**	–	–	–	–	**48**	**240**

Skenario 1 menjalankan empat evaluasi kandidat pada tiga *workload*. Jumlah eksekusinya adalah

$$4 \times 3 \times 1 \times (1 + 5) = 72. \tag{5.1}$$

Skenario 2 dan Skenario 3 masing-masing menjalankan enam evaluasi kandidat pada tiga *workload*. Jumlah eksekusi pada setiap skenario adalah

$$6 \times 3 \times 1 \times (1 + 5) = 108. \tag{5.2}$$

Secara keseluruhan, tahap *offline* menjalankan eksekusi kueri sebagai berikut.

$$72 + 108 + 108 = 288 \tag{5.3}$$

Jumlah tersebut terdiri atas 48 *warm-up* dan 240 pengukuran utama. Biaya terbesar berasal dari eksekusi kueri pada seluruh evaluasi kandidat. Perhitungan normalisasi, skor *workload*, dan utilitas dilakukan setelah data pengukuran tersedia.

Pertumbuhan biaya tahap *offline* bersifat linear terhadap jumlah evaluasi kandidat, jumlah *workload*, jumlah kueri uji, dan jumlah eksekusi. Penambahan kandidat atau pengulangan menambah jumlah proses pengukuran dalam proporsi yang sama.

Tahap *online*. Satu *window* tahap *online* menjalankan sekumpulan kueri, menghitung metrik, membaca kueri untuk proses *rewrite*, dan memilih satu aksi. Berdasarkan Persamaan 4.18, kompleksitas satu *window* ditulis sebagai berikut.

$$T_{online} = O(q_w T_{query}) + O(L) + O(p + a),$$

q_w sebagai jumlah kueri dalam satu *window*, L sebagai panjang kueri yang dibaca oleh *query rewriter*, p sebagai jumlah komponen metrik, dan a sebagai jumlah aksi yang diperiksa oleh *adaptive schema selector*.

Komponen $O(q_w T_{query})$ berasal dari eksekusi seluruh kueri dalam satu *window*. Komponen $O(L)$ berasal dari pembacaan dan penyesuaian predikat waktu oleh *query rewriter*. Komponen $O(p + a)$ berasal dari perhitungan metrik dan pemilihan aksi setelah eksekusi kueri selesai.

Jumlah proses tahap *online* disajikan pada Tabel 5.21.

Tabel 5.21: Jumlah proses tahap *online*

Konfigurasi	**Jumlah *window***	**Kueri per *window***	**Total eksekusi kueri**	**Pemilihan aksi**
Baseline	5	q_w	$5q_w$	0
Skenario 1	5	q_w	$5q_w$	5
Skenario 2	5	q_w	$5q_w$	5
Skenario 3	5	q_w	$5q_w$	5
Total	**20**	–	$\mathbf{20q_w}$	**15**

Baseline dan ketiga konfigurasi terpilih masing-masing menjalankan lima *window*. Jika satu *window* memuat q_w kueri, setiap konfigurasi menjalankan $5q_w$ eksekusi. Seluruh tahap *online* menjalankan eksekusi kueri sebagai berikut.

$$4 \times 5q_w = 20q_w$$

Pada Skenario 1, Skenario 2, dan Skenario 3, estimasi utilitas dan pemilihan aksi dijalankan satu kali setelah setiap *window* selesai. Lima *window* pada tiga skenario menghasilkan proses pemilihan aksi sebagai berikut.

$$3 \times 5 = 15$$

Baseline menjalankan lima *window* tanpa proses tersebut karena konfigurasinya dipertahankan selama pengujian.

Biaya tahap *online* terutama mengikuti jumlah kueri di dalam setiap *window*. Pembacaan kueri, perhitungan metrik, dan pemilihan aksi menjadi proses tambahan setelah eksekusi kueri. Pada eksperimen ini, proses tambahan tersebut tercatat melalui penalti *overhead* sebesar 0,3600 pada Skenario 1 serta 0,4800 pada Skenario 2 dan Skenario 3.

Analisis kompleksitas menunjukkan bahwa biaya tahap *offline* mengikuti jumlah evaluasi kandidat dan pengulangan, sedangkan biaya tahap *online* mengikuti jumlah kueri dan jumlah *window*. Latensi, BMR, WSS, dan *overhead* pada bagian sebelumnya menunjukkan hasil aktual dari proses tersebut pada lingkungan PostgreSQL/TimescaleDB *single-VM*.

BAB VI
KESIMPULAN DAN SARAN

6.1 Kesimpulan

Berdasarkan hasil penelitian dan pembahasan yang telah dilakukan, diperoleh kesimpulan sebagai berikut.

1. Penelitian ini telah berhasil mengadaptasi mekanisme berbasis PROADAPT agar dapat diterapkan pada *data warehouse* berbasis PostgreSQL/TimescaleDB dalam lingkungan *single-VM*. Tahap *offline* digunakan untuk memilih konfigurasi awal partisi, sedangkan tahap *online* digunakan untuk memantau perubahan *workload* dan menentukan aksi penyesuaian.

2. Penelitian ini telah berhasil menyusun fungsi *utility* yang *memory-aware* sebagai dasar pemilihan konfigurasi partisi dan kebijakan kompresi pada TimescaleDB. Fungsi tersebut menggabungkan *latency score*, *pruning score*, *memory score* berbasis BMR dan WSS, penalti ukuran penyimpanan, serta penalti *overhead* untuk menentukan peringkat kandidat pada setiap skenario *offline*.

3. Penelitian ini telah berhasil mengevaluasi dan membandingkan kinerja konfigurasi berbasis PROADAPT dengan konfigurasi *baseline* yang statis. Rata-rata latensi dari tiga skenario pengujian sebesar 432,474 ms atau turun 14,12% dibandingkan *baseline* sebesar 503,562 ms. Rata-rata BMR yang dihasilkan sebesar 0,001958 atau turun 79,30% dibandingkan *baseline* sebesar 0,009458.

6.2 Saran

Berdasarkan hasil penelitian yang telah dilakukan, saran untuk pengembangan penelitian selanjutnya adalah sebagai berikut.

1. Penelitian selanjutnya dapat mengembangkan mekanisme pembentukan kandidat konfigurasi secara adaptif berdasarkan karakteristik *workload*, serta mengintegrasikannya dengan *dashboard online* yang berjalan secara berkelanjutan untuk mendukung penyesuaian konfigurasi ketika pola kueri berubah.

2. Penelitian selanjutnya dapat memperluas evaluasi melalui penggunaan dataset, pola kueri, durasi pengujian, dan konfigurasi sumber daya yang lebih beragam.

DAFTAR PUSTAKA

Aly, A. M., Mahmood, A. R., Hassan, M. S., Aref, W. G., Ouzzani, M., Elmeleegy, H., dan Qadah, T. (2015). AQWA: Adaptive query-workload-aware partitioning of big spatial data. *Proceedings of the VLDB Endowment*, *8*(13), 2062–2073. doi: 10.14778/2831360.2831361

Badan Meteorologi, Klimatologi, dan Geofisika. (2026). *Bmkg – badan meteorologi, klimatologi, dan geofisika.* `https://www.bmkg.go.id/`. (Diakses 15 Juli 2026)

Benkrid, S., Bellatreche, L., Mestoui, Y., dan Ordonez, C. (2022). PROADAPT: Proactive framework for adaptive partitioning for big data warehouses. *Data & Knowledge Engineering*, *142*, 102102. doi: 10.1016/j.datak.2022.102102

Chou, H.-T., dan DeWitt, D. J. (1985). An evaluation of buffer management strategies for relational database systems. In *Proceedings of the 11th international conference on very large data bases* (pp. 127–141).

Direktorat Data dan Komputasi BMKG. (2024). *Data online bmkg.* `https://dataonline.bmkg.go.id/dataonline-home`. (Diakses 15 Juli 2026)

Inmon, W. H. (2005). *Building the data warehouse* (4th ed.). John Wiley & Sons.

Kimball, R., dan Ross, M. (2013). *The data warehouse toolkit: The definitive guide to dimensional modeling* (3rd ed.). John Wiley & Sons.

Kitchin, R., dan McArdle, G. (2016). What makes big data, big data? exploring the ontological characteristics of 26 datasets. *Big Data & Society*, *3*(1), 1–10. doi: 10.1177/2053951716631130

Lee, S., Son, Y., dan Kim, S. (2023). Analyzing I/O characteristics of time-series data using high performance storage devices. *IEEE Access*, *11*, 128998–129008. doi: 10.1109/ACCESS.2023.3329474

Liu, P.-J., Li, C.-P., dan Chen, H. (2024). Enhancing storage efficiency and performance: A survey of data partitioning techniques. *Journal of Computer Science and Technology*, *39*(2), 346–368. doi: 10.1007/s11390-024-3538-1

Lu, Y., Shanbhag, A., Jindal, A., dan Madden, S. (2017). AdaptDB: Adaptive partitioning for distributed joins. *Proceedings of the VLDB Endowment*, *10*(5), 589–600. doi: 10.14778/3055540.3055551

Martin, B., dan Davis, K. C. (2021). Multi-temperate logical data warehouse design for large-scale healthcare data. *Big Data Research*, *25*, 100255. doi: 10.1016/j.bdr.2021.100255

PostgreSQL Global Development Group. (2026a). *pg_stat_statements – track statistics of sql planning and execution.* `https://www.postgresql.org/docs/current/pgstatstatements.html`. (Diakses 15 Juli 2026)

PostgreSQL Global Development Group. (2026b). *Preset options: Block size.* `https://`

www.postgresql.org/docs/current/runtime-config-preset.html. (Diakses 15 Juli 2026)

Selinger, P. G., Astrahan, M. M., Chamberlin, D. D., Lorie, R. A., dan Price, T. G. (1979). Access path selection in a relational database management system. In *Proceedings of the 1979 acm sigmod international conference on management of data* (pp. 23–34). doi: 10.1145/582095.582099

Tiger Data. (2026a). *Compression in timescaledb.* https://docs.timescale.com/use-timescale/latest/compression/. (Dokumentasi resmi TimescaleDB, diakses 15 Juli 2026)

Tiger Data. (2026b). *Hypertables and chunks.* https://docs.timescale.com/use-timescale/latest/hypertables/. (Dokumentasi resmi TimescaleDB, diakses 15 Juli 2026)

UCAPAN TERIMA KASIH

Penyelesian penulisan tugas akhir ini tidak lepas dari orang-orang terdekat penulis. Penulis mengucapkan terima kasih yang sebesar-besarnya kepada:

1. Papa dan Mama yang telah membesarkan penulis dan senantiasa memberikan semangat tiada henti untuk terus berjuang menempuh pendidikan tinggi.
2. Diri sendiri dan Echa yang telah bersama-sama berusaha dan berjuang untuk menempuh pendidikan tinggi di ITS.
3. Teman-teman bimbingan anak Pak Budi yaitu, Candra, Nabila, Noni, Marsyanda, Aurel, Syifa, dan Aida yang telah memberikan banyak masukan, ide, serta teman untuk berkonsultasi seperjuangan dalam pengerjaan tugas akhir ini.
4. Teman-teman LIMITS Al Ittihad yang membersamai dari semester 2 hingga sekarang dan sampai waktu yang tidak ditentukan.
5. Teman-teman BPI Kabinet Baloko yang selalu membersamai perjuangan dalam mengurus angkatan AXXIIOM, Matematika ITS 2022, dan memberikan banyak masukan serta ide untuk mengembangkan diri.
6. Teman-teman URI terutama departemen Pengembangan Manajerial yang telah berkembang bersama untuk mengembangkan diri dan UKM Rebana ITS.
7. Teman-teman NechCode yaitu, Dios, Reina, dan Rizqi yang telah membersamai penulis memberikan pengalaman untuk mengembangkan agensi digital selama perkuliahan.
8. Teman-teman AXXIIOM, Matematika ITS 2022, dan seluruh teman-teman lainnya yang tidak bisa disebutkan satu persatu. Terimakasih telah memberi pengalaman tak terlupakan

Penulis juga mengharapkan kritik dan saran dari berbagai pihak sebagai bahan evaluasi penyempurnaan isi dalam laporan tugas akhir ini. Sekian dari penulis, semoga tugas akhir ini bermanfaat bagi semua pihak yang bersangkutan.

Surabaya, 22 Juli 2026

Hafidz Mulia

BIODATA PENULIS

Hafidz Mulia adalah mahasiswa Departemen Matematika, Fakultas Sains dan Analitika Data, Institut Teknologi Sepuluh Nopember (ITS) yang diterima pada tahun 2022. Sebelum menempuh pendidikan di ITS, penulis menyelesaikan pendidikan menengah di SMA IT Al Uswah Surabaya pada tahun 2019-2022. Selama menjalani studi di ITS, penulis aktif dalam kegiatan organisasi dan pengembangan mahasiswa. Penulis pernah menjadi Ketua Angkatan Matematika 2022 atau AXXIIOM STI 57, menjabat sebagai Wakil Ketua I HIMATIKA ITS pada tahun 2025, Kepala Departemen Kaderisasi LIMITS pada tahun 2024, dan Wakil Kepala Departemen Pengembangan Manajerial UKM REBANA-ITS pada tahun 2024. Dalam bidang akademik, penulis pernah menjadi asisten mata kuliah Algoritma dan Pemrograman I dan II serta Koordinator Asisten Laboratorium Pemrograman dan Komputasi Visual Departemen Matematika ITS. Penulis juga mengembangkan minat dalam bidang pemrograman dan pendidikan teknologi dengan bekerja sebagai *Programming Teacher* di Timedoor Academy serta membangun agensi teknologi digital sebagai Founder NechCode. Bidang yang diminati penulis meliputi pengembangan perangkat lunak, basis data, dan pendidikan pemrograman. Pihak yang ingin berdiskusi atau menghubungi penulis dapat melalui email hafidzmuliia@gmail.com. Terimakasih.