Jurnal Pendidikan dan Teknologi Indonesia (JPTI)
Vol. 5, No. 6, Juni 2025, Hal. 1769-1785
DOI: https://doi.org/10.52436/1.jpti.824
p-ISSN: 2775-4227
e-ISSN: 2775-4219

Optimasi Performa Query Subsidi Debitur dengan Index and Table Partition, Subquery and Indexing, dan Parallel Query Execution

Eko Firmansyah[*1], Hanafi Firdaus[2], Samidi[3]

[1,2,3]Program Studi Magister Ilmu Komputer, Fakultas Teknologi Informasi, Universitas Budi Luhur, Indonesia
[*]Email: [1]2311601609@student.budiluhur.ac.id, [2]2311601583@student.budiluhur.ac.id, [3]samidi@budiluhur.ac.id

Abstrak

Pertumbuhan data yang pesat menyebabkan turunnya performa perhitungan subsidi debitur pada Sistem Informasi Kredit Program (SIKP). Waktu proses selama 8 jam membuat layanan SIKP terganggu karena pemangku kebijakan tidak dapat mencetak laporan. Penelitian ini bertujuan untuk mengoptimalkan perhitungan subsidi debitur dengan metode optimasi *query*. Metode yang akan digunakan mencakup *Index and Table Partition Subquery and Indexing*, dan *Parallel Query Execution.* Pengujian dilakukan secara eksperimen menggunakan 300 juta sampel data subsidi debitur Bank Rakyat Indonesia (BRI) tahun 2022. Hasil penelitian menunjukan bahwa sebelum menggunakan metode optimasi, sistem membutuhkan waktu rata-rata 7.992,34 detik untuk menyelesaikan 1 siklus perhitungan data sampel. *Index and Table Partition* menghasilkan waktu rata-rata 7.097,01 detik. *Subquery and Indexing* menghasilkan waktu rata-rata 3.270,33 detik. *Parallel Query Execution* menghasilkan waktu rata-rata 5.923,67 detik. Hasil optimal diperoleh ketika ketiga metode dikombinasikan, menghasilkan waktu rata-rata 3.038 detik, yaitu 61,98% lebih efisien dibandingkan metode yang ada. Penerapan metode optimasi *query* ini secara langsung dapat meningkatkan efisiensi SIKP dan operasional pemangku kebijakan tidak terganggu. Hasil penelitian ini memberikan solusi optimasi *query* kombinasi untuk pengolahan data besar.

Kata kunci: *Index and Table Partition, Optimasi Query, Oracle Database, Parallel Query Execution, Pengolahan Data Besar, Subquery and Indexing.*

Query Performance Optimization of Debtor Subsidy with Index and Table Partition, Subquery and Indexing, and Parallel Query Execution

Abstract

The rapid growth of data has led to a decline in the performance of subsidy debtor calculations in the Sistem Informasi Kredit Program (SIKP). The 8-hour processing time disrupts SIKP services as policymakers are unable to generate reports. This study aims to optimize subsidy debtor calculations using query optimization methods. The methods applied include Index and Table Partition, Subquery and Indexing, and Parallel Query Execution. The experiment was conducted using 300 million subsidy debtor data samples from Bank Rakyat Indonesia (BRI) in 2022. The results indicate that before optimization, the system required an average processing time of 7,992.34 s to complete one cycle of data sample calculation. The Index and Table Partition method reduced the processing time to 7,097.01 s, while Subquery and Indexing achieved an average time of 3,270.33 s. The Parallel Query Execution method resulted in an average processing time of 5,923.67 s. The optimal result was achieved when the three methods were combined, resulting in an average processing time of 3,038 s, which is 61.98% more efficient than the existing method. The implementation of these query optimization methods directly enhances SIKP efficiency, ensuring that policymakers' operations remain unaffected. This study provides a combined query optimization solution for large-scale data processing.

Keywords: *Index and Table Partition, Large-Scale Data Processing, Oracle Database, Parallel Query Execution, Query Optimization, Subquery and Indexing.*

1. PENDAHULUAN

Di era digital saat ini, peran database terhadap kelangsungan bisnis sangatlah penting. Database menjadi muara data yang berguna untuk memenuhi seluruh kebutuhan para pelaku bisnis. Sistem Informasi Kredit Program (SIKP) adalah sistem yang digunakan untuk menatausahakan dan menyediakan informasi penyaluran kredit program dalam hal ini adalah data Kredit Usaha Rakyat (KUR) [1]. Data tersebut diolah secara periodik setiap bulan untuk menghasilkan informasi yang akurat dan komprehensif terkait subsidi debitur. Proses ini sudah berjalan sejak tahun 2016 hingga saat ini, sehingga jumlah data yang diolah meningkat secara signifikan. Berikut adalah rekap pertumbuhan data subsidi debitur yang diambil dari database SIKP.

Tabel 1. Rekap Pertumbuhan Data Subsidi SIKP

Tahun	Jumlah Data	Persentase Peningkatan
2016	59.036.471	99,99
2017	418.878.190	85,91
2018	723.871.532	50,29
2019	967.981.125	68,49
2020	1.136.718.290	78,53
2021	1.367.254.604	87,66
2022	1.535.952.766	84,99
2023	1.763.598.275	90,43

Dari data pada Tabel 1 dapat dilihat bahwa rata-rata pertumbuhan data per tahun adalah 80,79%. Pertumbuhan data yang cepat ini mengakibatkan peningkatan waktu latensi dan penurunan performa sistem secara keseluruhan [2]. Hal ini dapat dirasakan oleh para pengguna sistem ketika melakukan proses perhitungan subsidi debitur yang semakin lambat performanya. Saat ini waktu yang dibutuhkan sistem untuk menyelesaikan 1 siklus perhitungan subsidi debitur adalah 8 jam. Selama waktu proses tersebut, layanan SIKP terganggu sehingga pengguna sistem khususnya pemangku kebijakan tidak dapat mencetak laporan yang dibutuhkan. Permasalahan ini membuat penyedia sistem menghadapi tantangan dalam menangani jumlah data yang sangat besar sambil tetap menjaga kinerja tinggi, skalabilitas, dan ketersediaannya [3]. Penanganan data dengan skala besar pada database memerlukan teknik optimasi yang dapat mengatasi keterbatasan arsitektur yang ada dalam hal performa dan skalabilitas [4]. Dalam upaya pencarian teknik optimasi tersebut, peneliti mencoba untuk membandingkan berbagai metode yang telah ditawarkan oleh peneliti sebelumnya.

Database yang tidak menerapkan metode *index* akan mengalami kendala latensi pencarian data dan memakan banyak memory pada server [5]. *Index* dapat menjadi salah satu opsi metode optimasi database untuk meningkatkan performa *query* [6]. Penggunaan *Index and Table Partition* menghasilkan *response time* yang lebih baik ketika menyeleksi jutaan baris data [7]. *B-Tree Index* dirancang untuk menyeimbangkan waktu simpan dan proses pencarian sehingga performa *query* meningkat secara signifikan [8]. *Bitmap Index* sangat efektif untuk kolom dengan jumlah nilai unik yang rendah, tetapi kurang optimal dalam kondisi dengan kardinalitas tinggi maupun rendah [9]. Metode *Subquery and Indexing* menghasilkan peningkatan performa secara signifikan khususnya pada penggunaan *In* dan *Exist* [10]. Penggunaan metode *QuerySplit* dengan pendekatan *Subquery* mencegah kompleksitas tinggi pada sistem karena sudah membagi beban eksekusi secara terpisah [11]. Pemecahan beban ini memunculkan potensi optimasi tambahan yang dapat diterapkan semisal proses *Indexing*. Solusi optimasi *query* ini perlu memperhatikan jumlah data yang diolahnya. Jika data yang diolah sangat banyak, disarankan untuk tidak menggunakan query dengan tipe *correlated* karena membutuhkan waktu yang lebih banyak lagi [12]. Bentuk *query* dengan *join* lebih cepat hasil prosesnya jika dibandingkan dengan bentuk *query subselect* [13]. Metode *Parallel Query Execution* mempercepat waktu proses eksekusi *query* secara signifikan dengan memanfaatkan prosesor *multi-core* dan lingkungan terkluster [14]. Dalam eksekusi query hubungan spasial yang melibatkan enam *thread*, *Parallel Algorithm* menunjukkan peningkatan kinerja hingga 3,6 kali lipat pada SQLite/SpatiaLite dan 5,1 kali lipat pada PostgreSQL/PostGIS dibandingkan dengan pemrosesan *single-thread* [15].

Jika melihat kondisi dan ruang lingkupnya, terdapat perbedaan yang diperkirakan berdampak pada hasil penelitian yang akan dilakukan. Data yang digunakan untuk penelitian ini jauh lebih besar dengan tingkat kerumitan *query* yang jauh lebih kompleks. Adapun database yang digunakan pada penelitian ini adalah Oracle. Pada penelitian sebelumnya juga tidak ditemukan penggunaan kombinasi 3 metode optimasi *query*. Hal ini menunjukan adanya celah atau gap pengetahuan yang dapat dimanfaatkan peneliti untuk mengeksplorasi metode optimasi database dengan cara yang berbeda.

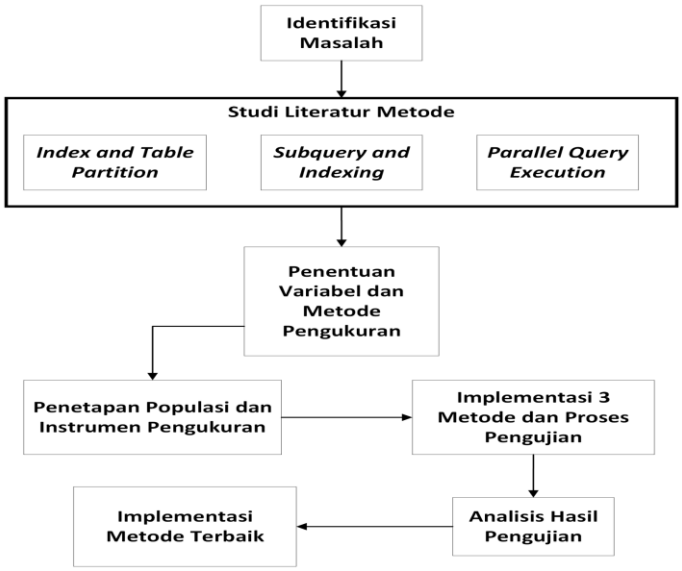
Penelitian ini bertujuan untuk mengoptimalkan performa perhitungan subsidi debitur dengan mengunakan metode optimasi *query* database. Metode optimasi ini akan diuji secara eksperimen sehingga peneliti dapat

p-ISSN: 2775-4227
e-ISSN: 2775-4219

mengidentifikasi metode terbaik dalam yang dapat diterapkan pada SIKP. Hasil penelitian ini memberikan impikasi praktis peningkatan kinerja sistem yang secara langsung dapat mencegah gangguan layanan dan mengurangi tingkat risiko *overhead* [16]. Implikasi lainnya adalah kontribusi pada ilmu pengetahuan berupa pembuktian, penyanggahan, modifikasi atau kombinasi metode dengan kondisi yang berbeda yang dapat dijadikan sumber literatur penelitian berikutnya.

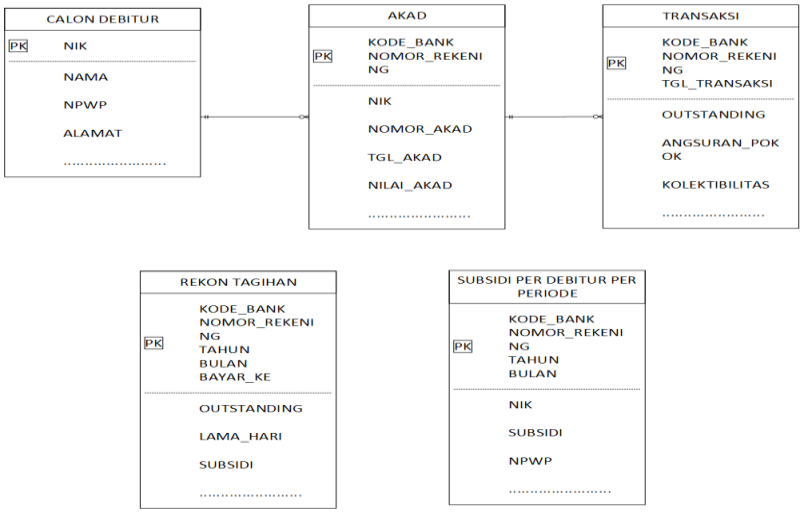
2. METODE PENELITIAN

Peneliti menggunakan metode eksperimen untuk menguji berbagai metode optimasi database pada penelitian sebelumnya. Pertama, perhitungan subsidi debitur diidentifikasi terlebih dahulu akar permasalahannya dengan menjalankan simulasi prosedur *eksisting* menggunakan data sampel. Hasil dari simulasi akan dituangkan kedalam tabel perbandingan. Setelah itu, peneliti akan melakukan studi literatur terkait metode yang akan diuji yaitu *Index and Table Partition*, *Subquery and Indexing,* dan *Parallel Query Execution.* Setelah dilakukan studi literatur, peneliti selanjutnya menentukan variabel, metode pengukuran, populasi, serta instrumen yang akan digunakan. Ketiga metode tersebut akan dilakukan proses pengujian dan dicatat kedalam tabel untuk dianalisis dan dinilai kinerjanya. Metode dengan peningkatan kinerja terbaik akan diterapkan kedalam sistem. Berikut adalah desain penelitian yang telah dirancang menggunakan metode eksperimen yang dijabarkan dalam Gambar 1.

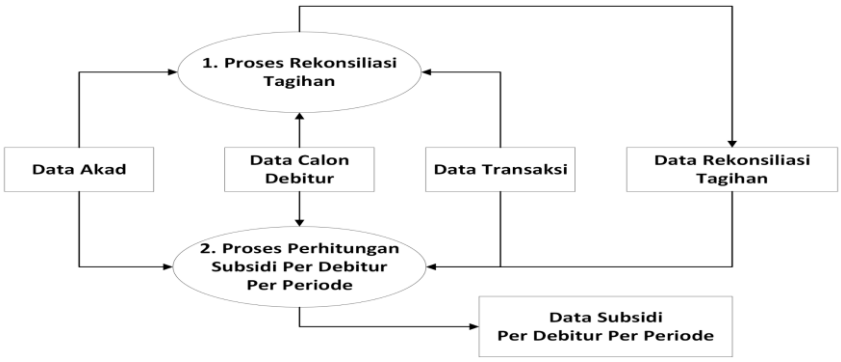


Gambar 1. Desain Penelitian

2.1. Identifikasi Masalah



Gambar 2. ERD SIKP



Gambar 3. Proses Perhitungan Subsidi Per Debitur Per Periode

Salah satu fungsi SIKP adalah melakukan proses pengolahan data calon debitur, akad, transaksi, dan rekonsiliasi tagihan menjadi data subsidi per debitur per periode. Subsidi yang dimaksud adalah subsidi bunga yang diterima oleh debitur KUR [17]. Tabel Calon Debitur memiliki hubungan *one to many* dengan Tabel Akad melalui kunci tamu NIK. Tabel Akad memiliki hubungan *one to many* dengan Tabel Transaksi melalui kunci tamu kombinasi KODE_BANK dan NOMOR_REKENING. Tabel Rekon Tagihan dan Tabel Subsidi Per Debitur Per Periode adalah *output* atau hasil proses sistem. *Entity Relationship Diagram* (ERD) pada database SIKP dijelaskan melalui Gambar 2, sedangkan proses yang dilakukan sistem untuk mengolah data subsidi dijelaskan melalui Gambar 3.

Untuk membatasi ruang lingkup penelitian, peneliti hanya akan menguji performa *query* pada proses nomor 2. Proses tersebut membutuhkan waktu eksekusi 8 hingga 10 jam lebih. Dari perhitungan ini peneliti mengidentifikasi bahwa terdapat masalah dalam performa yang disebabkan oleh jumlah data dan frekuensi pengolahan data. Solusi dari metode penelitian sebelumnya tidak dapat secara langsung diterapkan karena berbeda kondisi dan ruang lingkupnya.

2.2. Studi Literatur

Index dalam database adalah struktur data yang digunakan untuk membuat salinan terorganisir dari kolom tertentu dalam tabel, yang memungkinkan database menemukan baris data dengan lebih cepat. *Index* meningkatkan performa *query* karena dapat mengurangi jumlah halaman data yang harus dibaca dari disk. Akses langsung ke baris data yang relevan ini secara drastis menyebabkan penurunan kebutuhan akan pemindaian penuh seluruh baris data pada suatu tabel [18]. Melalui analisis matematis dan hasil eksperimen, struktur *B-tree Index* yang diusulkan selalu memberikan kinerja yang lebih baik dibandingkan dengan teknik yang ada [19]. Proses pencarian duplikasi data transaksi pada teknologi *blockchain* menjadi lebih cepat jika menggunakan *B-tree Index* [20]. Adapun keterbatasan pada metode ini adalah waktu penulisan data dan proses penghapusan data yang relatif lambat [21]. Keterbatasan lainnya adalah metode *B-tree Index* tidak berjalan dengan optimal jika diterapakan pada data multidimensi [22]. Penggunaan metode *Index and Table Partition* pada pemrosesan data sampel sejumlah 2.248.590 baris, meningkatkan performa *query* pencarian dibandingkan dengan proses pencarian tanpa metode optimasi [7]. Jika dibandingkan dengan penelitian yang akan dilakukan, metode ini memiliki kesamaan yaitu besarnya jumlah data diolah. Adapun perbedaan yang dapat diidentifikasi adalah jenis databasenya dimana peneliti sebelumnya menggunakan PostgreSQL, sedangkan penelitian ini menggunakan Oracle. Kompleksitas *query* yang akan diuji juga sangat jauh berbeda yang menyebabkan interpretsasi hasil yang berbeda.

$$I_T^K = IDX_i \quad (1)$$

Pada persamaan (1), *I* adalah proses *indexing* pada tabel *T* yang memiliki kolom atau atribut *K*. Proses ini menghasilkan *index* dengan nama *IDXi*. Kolom yang diberi indeks adalah kolom yang memiliki banyak ikatan *Foreign Key* dengan tabel lain ketika dilakukan proses *Join*.

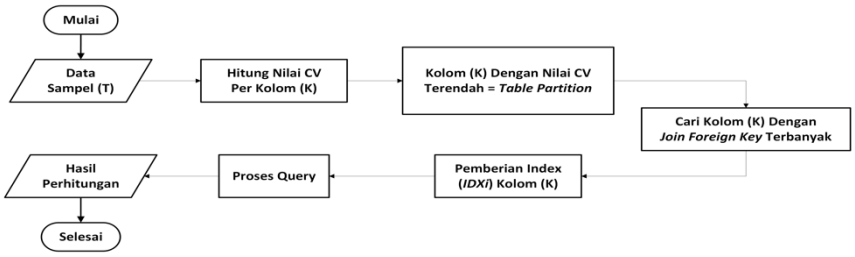
Table Partition meningkatkan performa dengan memecah tabel besar menjadi bagian-bagian yang lebih kecil sehingga sistem basis data hanya membaca dan memproses bagian yang relevan [16]. Kolom yang dilakukan partisi adalah kolom yang memiliki nilai sebaran merata atau berimbang dengan proses *select grouping*. Proses perhitungan sebaran data dilakukan dengan mengacu pada teori *Coefficient of Variation*. Teori ini dapat digunakan untuk menilai stabilitas suatu proses dengan mempertimbangkan hubungan antara rata-rata dan standar deviasi [23].

p-ISSN: 2775-4227
e-ISSN: 2775-4219

$$\text{CV} = \frac{\sigma}{\mu} \times 100\% \qquad (2)$$

Pada persamaan (2), *Coefficient of Variation* (CV) didapatkan dari persentase pembagian standar deviasi σ dengan rata-rata nilai percobaan per proses atau kolom μ. Kolom yang memiliki nilai CV terendah memiliki sebaran data yang paling berimbang.

Gambar 4 menunjukan langkah-langkah penerapan *Index and Table Partition.*



Gambar 4. *Flowchart* Penerapan Metode *Index and Table Partition*

Pemberian *table partition P* pada kolom *K* di tabel *T* dilakukan dengan cara sebagai berikut.

```
EXECUTE IMMEDIATE 'CREATE TABLE tabel_T (kolom_K VARCHAR2(100))
    PARTITION BY RANGE (kolom_K)
    (PARTITION p1 VALUES LESS THAN (100),
    PARTITION p2 VALUES LESS THAN (200))';
```

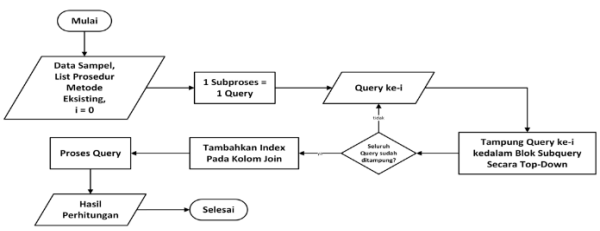
Pemberian *index I* pada kolom *K* di tabel *T* dilakukan dengan cara sebagai berikut.

```
EXECUTE IMMEDIATE 'CREATE INDEX IDX_kolom_K ON tabel_T(kolom_K)';
```

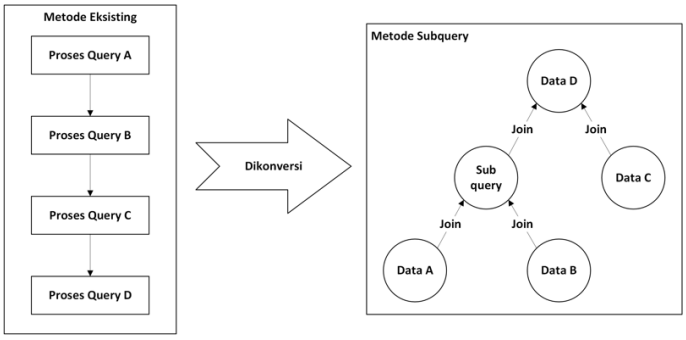
Subquery adalah *query* yang disisipkan di dalam pernyataan SQL lain, seperti dalam klausa SELECT, FROM, atau WHERE. *Subquery* digunakan untuk mengembalikan nilai tunggal atau sekumpulan hasil yang dapat digunakan sebagai kondisi dalam *query* utama [24]. *Subquery* dapat meningkatkan performa dengan mengisolasi logika *query* yang kompleks menjadi operasi yang lebih kecil dan lebih efisien yang dapat dioptimalkan secara mandiri [25]. Metode optimasi *subquery* yang tepat juga dapat meningkatkan potensi kinerja dan mengurangi tingkat kompleksitas database [26]. Penggunaan metode *Subquery* terbukti meningkatkan performa database dengan mengurangi waktu proses sebesar 2,55993 detik dibandingkan dengan proses tanpa metode optimasi [10]. Jika dibandingkan dengan penelitian yang akan dilakukan, terdapat kesamaan dalam hal optimasi yang dilakukan. Adapun perbedaannya terletak pada jenis database yang digunakan yaitu MySQL dan jumlah data sampel yang diuji yang dapat menyebabkan perbedaan interpretasi hasil. Peneliti juga menemukan gap pengetahuan bahwa metode *Subquery* dengan pendekatan *QuerySplit* berpeluang untuk dikombinasikan dengan metode lainnya untuk mencapai hasil yang maksimal [11]. Hal ini disebabkan *QuerySplit* memecah *query* yang rumit menjadi *subquery* yang lebih sederhana.

$$A \bowtie_\theta B = \{(a,b) | a \epsilon A, (b \epsilon B)\} \qquad (3)$$

Persamaan (3) merepresentasikan operasi *Left Outer Join* dalam basis data relasional. Operasi ini memastikan bahwa setiap baris dalam himpunan A akan tetap muncul dalam hasil akhir, dengan nilai dari himpunan B yang sesuai berdasarkan kondisi θ, jika tidak ditemukan pasangan yang cocok di B, maka nilai *NULL* akan digunakan. *Flowchart* penggunaan metode *Subquery and Indexing* dijelaskan melalui Gambar 5, sedangkan proses konversi metode *eksisting* menjadi *subquery* pada data sampel dijelaskan melalui Gambar 6.



Gambar 5. *Flowchart* Penerapan Metode *Subquery and Indexing*

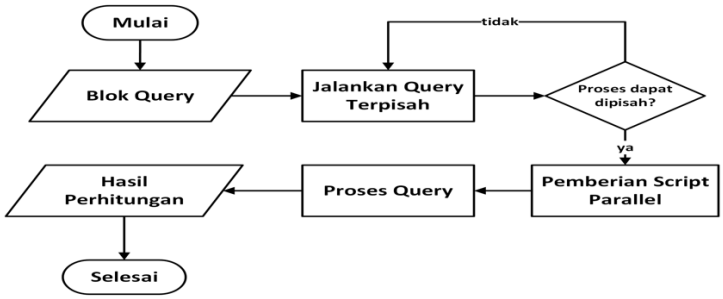


Gambar 6. Proses Konversi Subproses Menjadi *Subquery*

Karena SIKP menggunakan database Oracle, maka peneliti tertarik untuk menerapkan salah satu metode optimasi perhitungan pada Oracle. *Oracle Parallel Execution* adalah metode yang digunakan untuk mempercepat operasi dengan membagi satu tugas menjadi beberapa sub-tugas yang lebih kecil sehingga ratusan terabyte data dapat diproses dalam waktu hitungan menit [27]. Pemrosesan paralel di SQL Server sangat penting untuk menangani data dalam jumlah besar sehingga memungkinkan server lebih cepat ketika mengeksekusi karena dapat memanfaatkan semua sumber daya CPU dan Memory [14]. Dari penelitian itu belum didapatkan hasil pengukuran secara rinci peningkatan performa yang terjadi.

$$R \bowtie S = \bigcup_{i=1}^{p}(R_i \bowtie S_i) \qquad (4)$$

Persamaan (4) menunjukkan menunjukkan bahwa hasil akhir dari *join* antara tabel *R* dan *S* merupakan gabungan dari hasil *join* dari setiap subset data *Ri* dan *Si* yang telah dibagi berdasarkan jumlah prosesor *p*. Untuk meningkatkan efisiensi, teknik hash partitioning sering digunakan, di mana data dibagi ke dalam partisi sehingga setiap pasangan data dengan kunci yang sama akan diproses oleh prosesor yang sama, mengurangi *overhead* komunikasi antar prosesor dan mempercepat eksekusi *join*. Gambar 7 berikut menjelaskan *flowchart* penggunaan metode *Parallel Query Execution* pada data sampel.



Gambar 7. *Flowchart* Metode *Parallel Query Execution*

Pemberian *Parallel Query Execution* dilakukan dengan cara berikut.

```
EXECUTE IMMEDIATE 'SELECT /*+ PARALLEL(4) */ kolom1, kolom2
      FROM blok_query
      WHERE kondisi';
```

2.3. Variabel dan Metode Pengukuran

Variabel yang akan digunakan sebagai dasar pengukuran adalah waktu eksekusi dalam satuan detik. Untuk membatasi penelitian, peneliti tidak menggunakan pemakaian sumber daya CPU dan memori sebagai bagian dari pengukuran. Sedangkan metode pengukuran yang akan digunakan adalah sebagai berikut :

1. Performa q*uery* : Mengukur waktu eksekusi *query* dalam satuan detik,
2. Beban q*uery :* Mengukur jumlah baris yang diproses.

Karena peneliti tidak memiliki kewenangan *Database Administrator*, maka peneliti hanya dapat mengukur performa *query* melalui PC lokal yang terhubung pada jaringan database. Pencatatan hasil pengujian dilakukan secara manual dari *console* Oracle menggunakan aplikasi Toad kemudian dipindahkan ke dalam tabel hasil uji.

p-ISSN: 2775-4227
e-ISSN: 2775-4219

Data hasil uji ini akan dimasukan kedalam Jupiter Notebook untuk dianalisis sehingga menghasilkan grafik pengukuran. Agar hasil pengukuran menjadi lebih komprehensif, maka peneliti akan menguji sebanyak 3 kali untuk setiap proses dan mengambil nilai rata-ratanya untuk dianalisis.

2.4. Populasi, Sampel, dan Instrumen Pengukuran

Penelitian ini akan menggunakan populasi yang bersumber dari database SIKP. Karena jumlah data terlalu banyak, maka peneliti menggunakan sampel dari data subsidi Bank Rakyat Indonesia (BRI) tahun 2022 dengan jumlah 328.239.380 baris data dan tahun 2023 dengan jumlah 429.086.031 baris data. Data ini mencakup data sampel utama yaitu data rekonsiliasi tagihan bulanan, serta data tambahan berupa populasi yaitu data *header* tagihan, data debitur, data akad, dan data referensi. Pada Tabel 2 berikut dijelaskan struktur data tabel utama dan representasinya pada database SIKP.

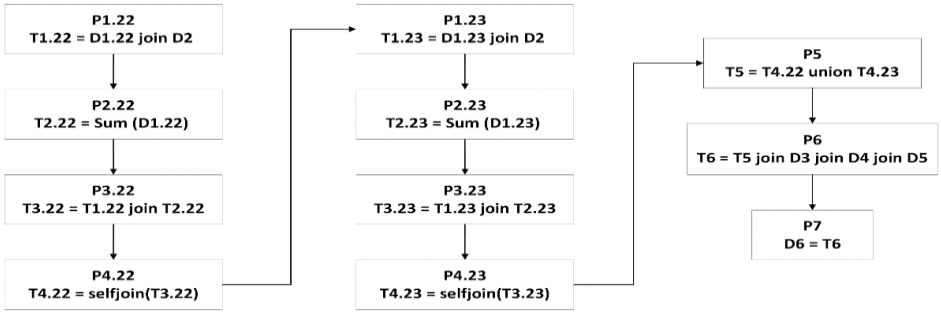
Tabel 2. Struktur Data Rekonsiliasi Tagihan Bulanan BRI

Nama Kolom	Tipe Kolom	Keterangan
ID_TAGIHAN_DETAIL	NUMBER	Kode Unik Baris/ Primary Key
KODE_BANK	VARCHAR2	3 digit Kode Penyalur KUR
TAHUN	VARCHAR2	Tahun subsidi
BULAN	VARCHAR2	Bulan subsidi
SKEMA	VARCHAR2	1=KUR Mikro, 2=KUR TMI, 3=KUR Kecil
SEKTOR_NEGARA_TUJUAN	VARCHAR2	Sektor Usaha/ Negara Tujuan
NIK	VARCHAR2	16 digit NIK debitur
REKENING_BARU	VARCHAR2	50 digit nomor rekening pinjaman debitur
OUTSTANDING	NUMBER	Nilai sisa outstanding/ pinjaman yang belum dibayar
LAMA_HARI	NUMBER	Jumlah hari terhitung subsidi
SUBSIDI_SIKP	NUMBER	Nilai subsidi yang berhak diterima debitur versi SIKP
SUBSIDI_BANK	NUMBER	Nilai subsidi yang berhak diterima debitur versi Penyalur
REKON	VARCHAR	Kode hasil rekonsiliasi (1=SAMA, 0=TIDAK SAMA)
BAYAR_KE	NUMBER	Pembayaran subsidi keberapa dalam periode yang sama

Instrumen pengukuran yang akan digunakan dalam penelitian ini adalah Oracle versi 19c, PC Windows 10 Pro 64-bit, 4 Core 2.7 Hz, 8 GB RAM, Toad for Oracle Xpert versi 11.5.1.2, Navicat Premium (64 bit) versi 12.1.18, Anaconda Toolbox versi 4.1.0, Jupiter Notebook versi 7.2.2.

3. HASIL DAN PEMBAHASAN

Sebelum melakukan proses implementasi dan menguji ketiga teori yang telah dijabarkan pada bab metode penelitian, peneliti menjelaskan kondisi detil alur proses *eksisting* terlebih dahulu. Perhitungan subsidi dengan metode *eksisting* dilakukan dengan cara menjalankan suatu prosedur Oracle yang terdiri dari 10 sub proses *query* terpisah. Prosedur ini dijelaskan melalui rangkaian serial proses pada Gambar 8.



Gambar 8. Prosedur Perhitungan *Eksisting*

Pada Gambar 8, proses perhitungan subsidi dimulai dengan menjalankan *query* P1.22 dimana 22 adalah representasi data sampel tahun 2022 yang berisi proses *Join* tabel D1.22 dengan tabel D2.22 sehingga menghasilkan tabel temporari T1.22. Selanjutnya sistem menjalankan *query* P2.22 yang berisi *Summary* tabel D1.22 sehingga menghasilkan tabel temporari T2.22. Kemudian sistem menjalankan *query* P3.22 yang berisi proses *Join* tabel T1.22 dengan tabel T2.22 sehingga menghasilkan tabel temporari T3.22. Berikutnya sistem

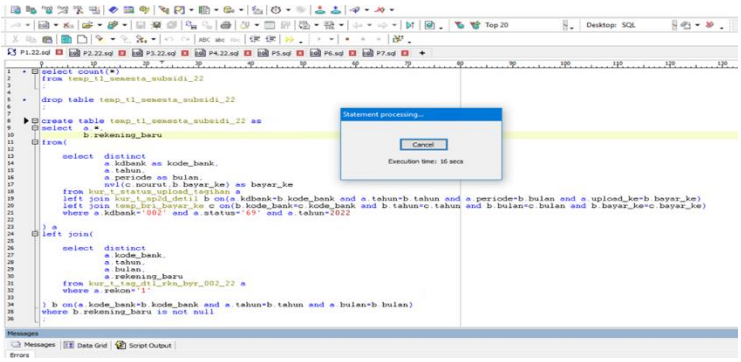
menjalankan *query* P4.22 yang berisi proses *SelfJoin* tabel T3.22 untuk menyandingkan baris dengan baris sebelumnya. Hal ini dilakukan juga untuk data sampel tahun 2023 sehingga menghasilkan tabel T4.23. Tabel T4.22 digabungkan dengan tabel T4.23 melalui proses *Union* untuk menghasilkan tabel temporari T5. Terakhir sistem menjalankan *query* P6 yang berisi proses *Join* tabel T5 dengan tabel D3, D4, dan D5 sehingga menghasilkan tabel temporari T6. Berikut adalah penjelasan nama tabel yang berasal dari database SIKP termasuk sampel yang ditetapkan beserta jumlah baris datanya.

Tabel 3. Nama Tabel Sampel dan Jumlah Barisnya

Tabel	Keterangan	Tipe	Jumlah Baris 2022	Jumlah Baris 2023
D1	Data Rekonsiliasi Tagihan	Sampel	328.239.380	429.086.031
D2	Data *Header* Tagihan	Populasi	15	21
D3	Data Akad (Rekening)	Populasi	35.047.496	38.588.699
D4	Data Calon Debitur (NIK)	Populasi	19.670.763	21.540.556
D5	Data Referensi	Populasi	1.012	1.012
D6	Data Subsidi Per Debitur	Temporari	147.234.256	179.945.454
T1	Temporari D1 *join* D2	Temporari	147.234.256	179.945.454
T2	Temporari *summary* D2	Temporari	136.905.359	168.581.165
T3	Temporari T1 *join* T2	Temporari	147.234.256	179.945.454
T4	Temporari *selfjoin* T3	Temporari	147.234.256	179.945.454
T5	Temporari T4 2022 *union* T4 2023	Temporari	327.179.710	327.179.710
T6	Temporari T5 *join D3, D4, D5*	Temporari	327.179.710	327.179.710

3.1. Pengujian Kondisi *Eksisting* Tanpa Metode Optimasi

Pada Gambar 9 berikut dilakukan proses eksekusi perhitungan subsidi debitur dengan metode *eksisting* pada data sampel yang sudah ditetapkan.



Gambar 9. Proses Eksekusi Prosedur Perhitungan *Eksisting*

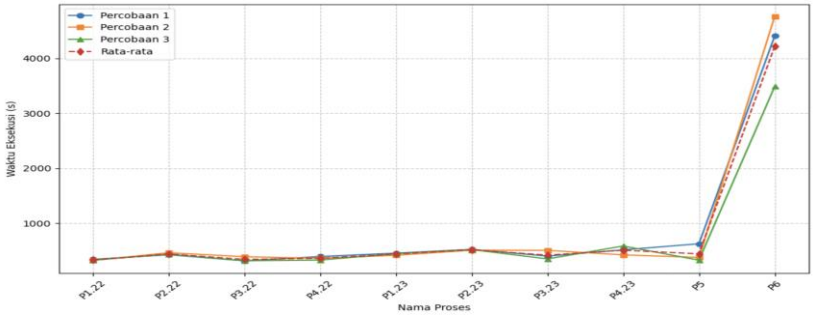
Berdasarkan hasil tiga kali percobaan proses *eksisting*, peneliti mendapatkan rata-rata nilai pengukuran total waktu yang dibutuhkan adalah 7992,34 detik dengan rincian yang tertuang pada Tabel 4.

Tabel 4. Pengukuran Waktu Proses *Eksisting*

Proses	Percobaan ke-1	Percobaan ke-2	Percobaan ke-3	Rata-rata
P1.22	339	319	325	327,67
P2.22	423	460	432	438,33
P3.22	312	387	314	337,67
P4.22	391	359	326	358,67
P1.23	452	413	443	436
P2.23	521	506	517	514,67
P3.23	399	503	347	416,33
P4.23	512	421	582	505
P5	623	374	324	440,33
P6	4406	4761	3486	4217,67

p-ISSN: 2775-4227
e-ISSN: 2775-4219

Gambar 10 berikut menunjukan hasil pengukuran pengujian metode *eksisting* menggunakan grafik.



Gambar 10. Grafik Pengukuran Waktu Proses Metode *Eksisting*

Dari pengukuran hasil uji pada Tabel 4 dan Gambar 10 tersebut dapat dilihat bahwa lonjakan waktu proses terjadi ketika sistem memroses prosedur P6 dengan waktu rata-rata 4217,67 detik. Hal ini terjadi karena *query* pada prosedur itu mengolah data besar dengan cara *Left Join* kebanyak tabel yaitu D3, D4, dan D5.

3.2. Pengujian Metode *Index* dan *Table Partition*

Berdasarkan *flowchart* penerapan metode *Index and Table Partition* pada Bab Metode Penelitian, langkah pertama yang dilakukan peneliti adalah menetapkan kolom yang akan diberi *Table Partition* karena partisi tabel hanya bisa dilakukan sebelum tabel terbentuk. Peneliti menjalankan *query* penentuan kolom yang akan dipartisi dengan menggunakan teori *Coefficient of Variation*.

```
WITH Counts AS (
    SELECT 'SKEMA' AS Kolom, skema AS Kategori, COUNT(*) AS Jumlah FROM temp_d1_rekon_tagihan_22 GROUP BY skema
    UNION ALL
    SELECT 'SEKTOR_NEGARA_TUJUAN', sektor_negara_tujuan, COUNT(*) FROM temp_d1_rekon_tagihan_22 GROUP BY sektor_negara_tujuan
    UNION ALL
    SELECT 'BULAN', bulan, COUNT(*) FROM temp_d1_rekon_tagihan_22 GROUP BY bulan
),
Stats AS (
    SELECT Kolom,
           AVG(Jumlah) AS Mean_Count,
           STDDEV(Jumlah) AS Std_Count,
           (STDDEV(Jumlah) / NULLIF(AVG(Jumlah), 0)) AS Coef_Variation,
           MAX(Jumlah) - MIN(Jumlah) AS Range_Count
    FROM Counts
    GROUP BY Kolom
)
SELECT * FROM Stats
ORDER BY Coef_Variation ASC;  -- Urutkan dari yang paling seimbang
```

Data Grid

Messages | Data Grid | Script Output

KOLOM	MEAN_COUNT	STD_COUNT	COEF_VARIATION	RANGE_COUNT
BULAN	27353281,6666667	9605897,83095053	0,351178990075494	27509607
SKEMA	82059845	130851589,969203	1,59458734011992	275972367
SEKTOR_NEGARA_TUJUAN	25249183,0769231	45090000,368327	1,78580036553887	155663067

Gambar 11. Penentuan Kolom Partisi Menggunakan Teori CV

Langkah selanjutnya berdasarkan Gambar 11, hasil perhitungan menunjukan bahwa kolom BULAN memiliki nilai CV terendah yang artinya kolom tersebut memiliki sebaran data yang paling berimbang atau optimal. Atas dasar hasil itu peneliti membuat ulang tabel D1 menggunakan *Table Partition* pada kolom BULAN seperti yang tertuang pada Gambar 12.

```
CREATE TABLE TEMP_D1_REKON_TAGIHAN_22 (
    KODE_BANK VARCHAR2(10),
    TAHUN VARCHAR2(4),
    BULAN VARCHAR2(2),
    BAYAR_KE NUMBER(3),
    ...
)
PARTITION BY RANGE (BULAN)
    PARTITION p01 VALUES LESS THAN (2),
    PARTITION p02 VALUES LESS THAN (3),
    PARTITION p03 VALUES LESS THAN (4),
    ...
```

Gambar 12. Proses Pembuatan Tabel Menggunakan *Table Partition*

Langkah berikutnya adalah peneliti menemukan kolom yang paling banyak dijadikan kunci untuk dilakukan proses *Join*. Berdasarkan pengamatan pada seluruh *query* yang terlibat, kolom KODE_BANK, TAHUN, BULAN, dan BAYAR_KE merupakan kolom yang paling banyak dijadikan kunci untuk proses *Join*. Tipe *index* yang digunakan adalah *B-tree Index*. Oleh karena itu, peneliti menerapkan penggunaan *Index* pada kolom tersebut seperti yang terlihat pada Gambar 12.

```
CREATE INDEX IDX_TEMP_D1_REKON_TAGIHAN
ON TEMP_D1_REKON_TAGIHAN_22 (KODE_BANK, TAHUN, BULAN, BAYAR_KE);
```

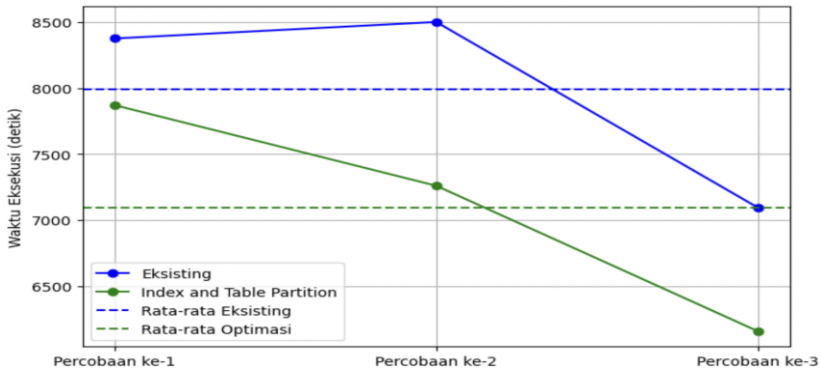
Gambar 13. Pemberian *Index* pada Tabel D1

Langkah terakhir adalah peneliti menjalankan ulang prosedur perhitungan subsidi debitur dengan kondisi menggunakan metode optimasi *Index and Table Partition* dan mencatat hasilnya. Berdasarkan hasil tiga kali percobaan pengujian peneliti mendapatkan nilai rata-rata pengukuran terhadap total waktu yang dibutuhkan adalah 7097,01 detik dengan rincian yang tertuang pada Tabel 5.

Tabel 5. Pengukuran Waktu Metode *Index* dan *Table Partition*

Proses	Percobaan ke-1	Percobaan ke-2	Percobaan ke-3	Rata-rata
P1.22	130	138	124	130,67
P2.22	107	108	111	108,67
P3.22	275	272	296	281
P4.22	394	303	394	363,67
P1.23	155	141	135	143,67
P2.23	158	155	180	164,33
P3.23	529	395	552	492
P4.23	545	466	505	505,33
P5	302	522	375	399,67
P6	5277	4761	3486	4508

Gambar 14 berikut menunjukan hasil pengukuran pengujian metode *eksisting* menggunakan grafik.



Gambar 14. Grafik Pengukuran Waktu Proses Metode *Index and Table Partition*

Hasil pengujian metode *Index* dan *Table Partition* pada Tabel 5 dan Gambar 14 menunjukan sedikit peningkatan performa jika dilihat dari waktu proses total rata-rata yaitu sebesar 11,19%, dengan waktu proses berkurang dari 7992,34 detik menjadi 7097,01 detik. Peningkatan ini sesuai dengan hasil yang ditunjukan oleh penelitian sebelumnya walaupun persentase peningkatannya cukup jauh berbeda. Dalam penelitian sebelumnya, penerapan metode optimasi menghasilkan peningkatan performa sebesar 84,66%, dengan waktu proses yang berkurang dari 763 milidetik menjadi 117 milidetik [7]. Perbedaan hasil uji ini dapat kita analisis lebih lanjut dengan melihat rincian prosesnya. Peningkatan performa yang signifikan ditunjukan pada proses P1 dan P2. Performa P1 meningkat sebesar 60% hingga 67%, sedangkan performa P2 meningkat sebesar 68% hingga 75%. Namun terdapat penurunan performa pada proses P6 sebesar 6% yang menyebabkan peningkatan secara keseluruhan tidak signifikan. Tabel yang diolah pada proses P6 merupakan tabel dengan kondisi data multidimensi atau memiliki lebih dari 1 atribut kunci yaitu KODE_BANK, TAHUN, BULAN, dan BAYAR_KE. Kondisi ini

p-ISSN: 2775-4227
e-ISSN: 2775-4219

sesuai dengan keterbatasan metode *Index* pada teori yang telah dijelaskan sebelumnya [22]. Hal inilah yang menyebabkan perbedaan hasil uji dengan penelitian sebelumnya sangat signifikan.

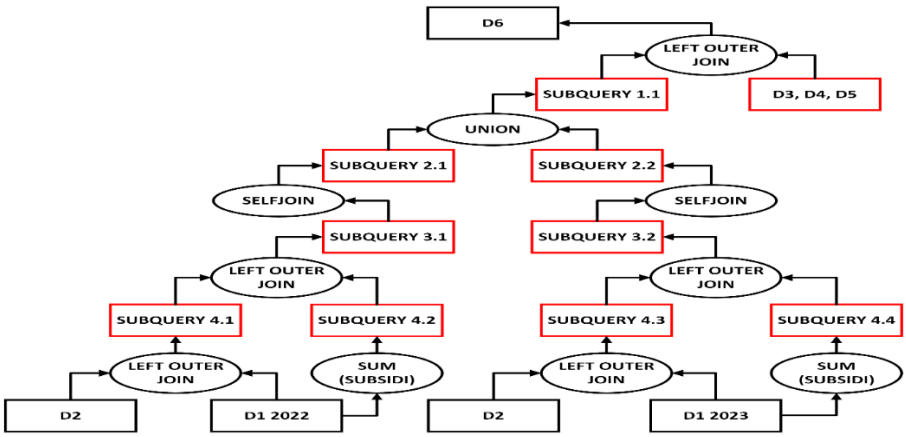
3.3. Pengujian Metode *Subquery and Indexing*

Berdasarkan *flowchart* penerapan metode *Subquery and Indexing* pada Bab Metode Penelitian, langkah pertama yang dilakukan peneliti adalah mengubah subproses pada metode eksisting menjadi *query* tunggal. Langkah kedua adalah memetakan *query* dengan *subquery*-nya serta diberikan nama label *subquery* seperti yang dijelaskan pada Tabel 6 berikut ini.

Tabel 6. Pembentukan Struktur *Query Top Down*

Nama Proses	Nama *Subquery*	Keterangan
P5	1.1	subquery 2.1 union subquery 2.2
P4.22	2.1	Selfjoin dari subquery 3.1
P4.23	2.2	Selfjoin dari subquery 3.2
P3.22	3.1	Subquery 4.1 left join Subquery 4.2
P3.23	3.2	Subquery 4.3 left join Subquery 4.4
P2.22	4.1	Tabel D1 2022 left outer join Tabel D2
P1.22	4.2	Tabel D1 2022 sum(subsidi)
P2.23	4.3	Tabel D1 2023 left outer join Tabel D2
P1.23	4.4	Tabel D1 2023 sum(subsidi)

Langkah ketiga adalah memasukan *subquery* kedalam 1 blok besar *query* utama secara *top down* satu per satu sampai seluruh *subquery* habis. Proses pemindahan ini dapat dilihat pada Gambar 15.



Gambar 15. Diagram Struktur *Subquery*

Pada Gambar 16 berikut adalah penerapan metode *Subquery* dari proses *eksisting* menjadi 1 *query* tunggal yang berisi 9 s*ubquery* dengan kedalaman 4 tingkat.

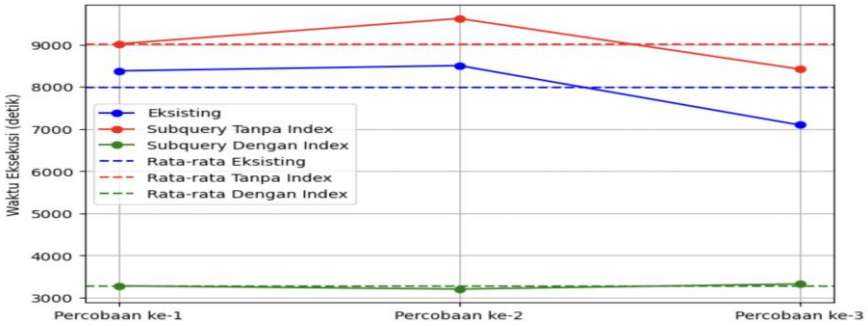
```
create table temp_t6_subsidi_all_byr_lgkp as
select  A.*,C.NIK,C.TGL_LAHIR,D.KODE_WILAYAH,D.NAMA_WILAYAH,SUBSTR(C.KODE_KABKOTA,1,2) AS KDPROV,E.NAMA_WILAYAH AS NMPROV,B.S
from(

    -- subquery 1.1
    select  a.*
    from(

        -- subquery 2.1
        select  a.kode_bank,a.rekening_baru,a.tahun,a.bulan,a.bayar_ke,a.subsidi as subsidi_ini,nvl(b.subsidi,0) as subsidi_s
        from(

            -- subquery 3.1
            select  a.*,nvl(b.subsidi,0) as subsidi
            from(

                -- subquery 4.1
                select  a.*,b.rekening_baru
                from
                left join(

                    select  distinct a.kode_bank,a.tahun,a.bulan,a.rekening_baru
                    from temp_d1_rekon_tagihan_22 a
                    where a.rekon='1'

                ) b on(a.kode_bank=b.kode_bank and a.tahun=b.tahun and a.bulan=b.bulan)
                where b.rekening_baru is not null

            ) a
            left join(

                -- subquery 4.2
                select  a.kode_bank,a.rekening_baru,a.tahun,a.bulan,nvl(b.nourut,a.bayar_ke) as bayar_ke,sum(a.subsidi_sikp)
                from temp_d1_rekon_tagihan_22 a
                left join temp_bri_bayar_ke b on(a.kode_bank=b.kode_bank and a.tahun=b.tahun and a.bulan=b.bulan and a.bayar_
                where a.kode_bank='002' and a.rekon='1' and a.tahun=2022
                group by a.kode_bank,
                         a.rekening_baru
```
Gambar 16. Implementasi Metode *Subquery*

Untuk mendapatkan hasil penelitian yang lebih mendalam, pengujian dilakukan dengan 2 cara yaitu tanpa *Index* dan dengan *Index.* Berdasarkan hasil tiga kali percobaan pengujian metode *Subquery*, peneliti mendapatkan nilai pengukuran terhadap waktu yang dibutuhkan dengan rincian pada Tabel 7 sebagai berikut.

Tabel 7. Pengukuran Waktu Metode *Subquery*

Proses	Percobaan ke-1	Percobaan ke-2	Percobaan ke-3	Rata-rata
Tanpa Index	9019	9620	8420	9019,67
Dengan Index	3281	3205	3325	3270,33

Gambar 17 berikut menujukan hasil pengukuran pengujian metode *Subquery and Indexing* menggunakan grafik.

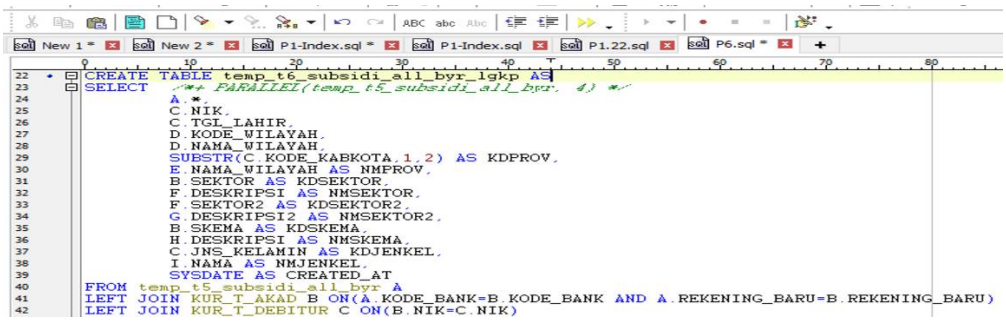


Gambar 17. Grafik Perbandingan Pengukuran Metode *Subquery and Indexing*

Hasil pengujian metode *Subquery and Indexing* pada Tabel 7 dan Gambar 17 menunjukan peningkatan performa yang signfikan. Peningkatan tersebut hanya muncul pada *subquery* yang menggunakan *index* yaitu sebesar 59,89%, dengan rata-rata waktu proses berkurang dari 7992,34 detik menjadi 3270,33 detik. Dalam penelitian sebelumnya, penerapan metode optimasi menghasilkan peningkatan performa sebesar 83,11%, dengan waktu proses yang berkurang dari 3,08008 detik menjadi 0.52015 detik [10]. Perbedaan hasil uji ini dapat kita analisis lebih lanjut dengan melihat tingkat kompleksitas susunan *query* dan jumlah data yang diolah. Penelitian ini menggunakan data sampel ratusan juta baris dengan kedalaman *subquery* 4 tingkat menggunakan mekanisme *Join*. Sedangkan penelitian terdahulu hanya menggunakan *subquery In* dan *Exists* saja dengan jumlah data ratusan ribu [10]. Perbedaan inilah yang menyebabkan hasil pengujian tidak sama peningkatan performanya. Hasil pengujian metode *Subquery* tanpa *Index* menunjukan penurunan performa secara signifkan sebesar 12,85%, dengan rata-rata waktu proses bertambah dari 7992,34 detik menjadi 9019,67 detik. Peneliti menganalisis bahwa penurunan tersebut terjadi karena *subquery* digunakan bersamaan dengan proses *Join* yang menyebabkan peningkatan beban pada sistem untuk melakukan relasi operasional antara kolom.

3.4. Pengujian Metode *Parallel Query Execution*

Berdasarkan *flowchart* penerapan metode *Parallel Query Execution* pada Bab Metode Penelitian, langkah pertama yang dilakukan peneliti adalah mengecek ketergantungan proses atau blok *query.* Peneliti menilai bahwa proses yang dapat dijalankan secara paralel adalah proses P6. Langkah kedua adalah peneliti menerapkan *Oracle Parallel Execution* pada *query* yang digunakan di proses P6 seperti pada Gambar 18 berikut.



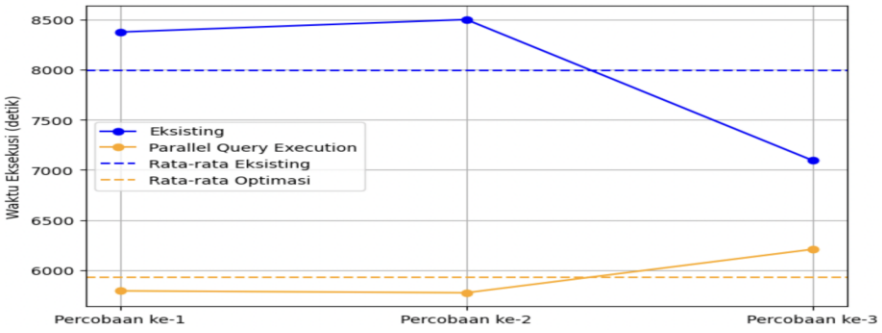
Gambar 18. Implementasi Metode *Parallel Query Execution*

p-ISSN: 2775-4227
e-ISSN: 2775-4219

Berdasarkan hasil tiga kali percobaan pengujian metode *Parallel Query Execution*, peneliti mendapatkan nilai pengukuran terhadap waktu yang dibutuhkan dengan rincian pada Tabel 8 sebagai berikut.

Tabel 8. Pengukuran Waktu Metode Parallel Query Execution

Percobaan ke-1	Percobaan ke-2	Percobaan ke-3	Rata-rata
5791,67	5772,67	6206,67	5923,67



Gambar 19. Grafik Pengukuran Waktu Proses Metode *Parallel Query Execution*

Hasil pengujian metode *Parallel Query Execution* pada Tabel 8 dan Gambar 19 menunjukan sedikit peningkatan performa yaitu sebesar 25,88%, dengan rata-rata waktu proses berkurang dari 7992,34 detik menjadi 5923,67 detik. Namun jika kita lihat pada percobaan ketiga, metode optimasi ini mengalami penurunan efesiensi dengan meningkatnya waktu proses eksekusi *query*. Peneliti menduga hal ini disebabkan metode ini menggunakan banyak sumber daya pada server melalui banyak *thread.* Dalam penelitian sebelumnya, penerapan metode optimasi yang dilakukan belum disertakan bukti peningkatan performa secara rinci [14]. Hasil penelitian ini memberikan bukti ilmiah berupa persentase peningkatan performa yang terjadi.

3.5. Pengujian Kombinasi Metode *Index*, *Table Partition*, Subquery, dan *Parallel Query Execution*

Karena ketiga metode pada pengujian sebelumnya memberikan hasil yang positif, maka peneliti tertarik untuk melakukan eksperimen dengan mencoba kombinasi ketiga metode optimasi ini dan menguji hasilnya dengan data sampel yang sama. Pertama, peneliti menggunakan metode *Table Partition* untuk membuat tabel sampel D1 dengan partisi kolom BULAN. Kedua, peneliti menambahkan metode *Index* pada kolom KODE_BANK, TAHUN, BULAN, dan BAYAR_KE. Ketiga, peneliti mengubah susunan *query* menggunakan metode *Subquery* dan membentuknya menjadi 1 *query* tunggal. Setelah semua selesai, peneliti melakukan eksekusi *query* ini menggunakan metode *Parallel Query Execution.* Penerapan kombinasi metode ini dapat dilihat pada Gambar 20.

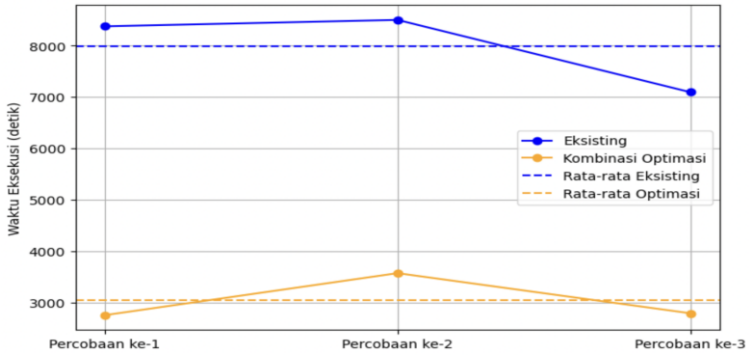
```
create table temp_t6_subsidi_all_byr_lgkp as
select  A.*,C.NIK,C.TGL_LAHIR,D.KODE_WILAYAH,D.NAMA_WILAYAH,SUBSTR(C.KODE_KABKOTA,1,2) AS KDPROV,E.NAMA_WILAYAH A
from(

    -- subquery 1.1
    select  a.*
    from(

        -- subquery 2.1
        select  a.kode_bank,a.rekening_baru,a.tahun,a.bulan,a.bayar_ke,a.subsidi as subsidi_ini,nvl(b.subsidi,0)
        from(

            -- subquery 3.1
            select  a.*,nvl(b.subsidi,0) as subsidi
            from(

                -- subquery 4.1
                select  a.*,b.rekening_baru
                from
                left join(

                    select  /*+ PARALLEL(temp_d1_rekon_tagihan_22, 4) */
                            distinct a.kode_bank,a.tahun,a.bulan,a.rekening_baru
                    from temp_d1_rekon_tagihan_22 a
                    where a.rekon='1'

                ) b on(a.kode_bank=b.kode_bank and a.tahun=b.tahun and a.bulan=b.bulan)
                where b.rekening_baru is not null

            ) a
            left join(

                -- subquery 4.2
                select  /*+ PARALLEL(temp_d1_rekon_tagihan_22, 4) */
                        a.kode_bank,a.rekening_baru,a.tahun,a.bulan,nvl(b.nourut,a.bayar_ke) as bayar_ke,sum(a.su
                from temp_d1_rekon_tagihan_22 a
                left join temp_bri_bayar_ke b on(a.kode_bank=b.kode_bank and a.tahun=b.tahun and a.bulan=b.bulan
                where a.kode_bank='002' and a.rekon='1' and a.tahun=2022
                group by a.kode_bank,
                        a.rekening_baru,
```

Gambar 20. Implementasi Kombinasi Metode Optimasi

Berdasarkan hasil tiga kali percobaan pengujian kombinasi metode optimasi, peneliti mendapatkan nilai pengukuran terhadap waktu yang dibutuhkan dengan rincian pada Tabel 9 sebagai berikut.

Tabel 9. Pengukuran Waktu Metode Kombinasi

Percobaan ke-1	Percobaan ke-2	Percobaan ke-3	Rata-rata
2753	3571	2790	3038

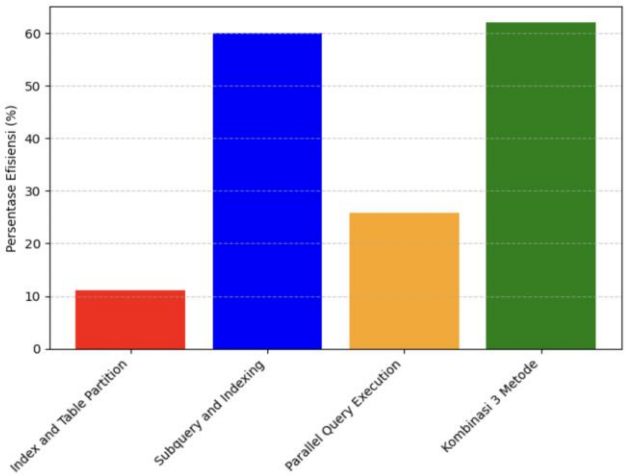


Gambar 21. Grafik Pengukuran Waktu Metode Kombinasi

Hasil pengujian kombinasi metode optimasi pada Tabel 9 dan Gambar 21 menunjukan peningkatan performa yang siginifikan yaitu sebesar 61,98%, dengan rata-rata waktu proses berkurang dari 7992,34 detik menjadi 3038 detik. Peneliti sudah melakukan pencarian jurnal ilmiah yang mengkaji topi kombinasi ketiga metode ini namun belum ditemukan sehingga hasil pengujian kombinasi metode ini tidak dapat dibandingkan secara langsung.

4. DISKUSI

Berdasarkan pengamatan hasil pengujian, peneliti menemukan fakta bahwa semua metode yang diuji menunjukan peningkatan performa pada proses perhitungan subsidi debitur. Peneliti menganalisis bahwa peningkatan tersebut dapat terjadi dengan kondisi atau syarat tertentu. Metode *Index and Table Partition* terbukti meningkatkan kinerja proses dengan rata-rata sebesar 11,19% secara keseluruhan. Metode *Subquery and Indexing* terbukti meningkatkan kinerja proses sebesar 59,89% secara keseluruhan. Sedangkan metode *Parallel Query Execution* terbukti meningkatkan kinerja proses sebesar 25,88% secara keseluruhan. Hasil yang paling optimal didapatkan ketika peneliti menggunakan ketiga metode optimasi ini secara bersama-sama yaitu sebesar 61,98%. Pada Gambar 22 dapat dilihat bagaimana perbandingan persentase peningkatan kinerja atau efesiensi untuk setiap metode.



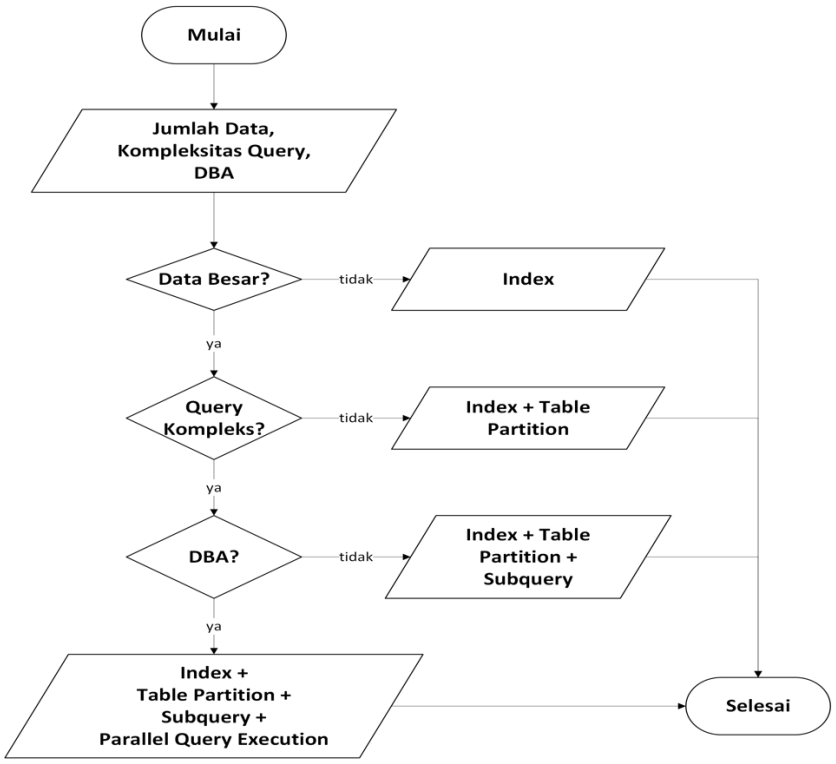
Gambar 22. Grafik Perbandingan Persentase Efesiensi Metode Optimasi

Hasil perbandingan persentase efesiensi metode optimasi pada Gambar 22 menunjukan peningkatan performa yang paling optimal adalah metode kombinasi. Selisih waktu antara metode kombinasi dengan metode

p-ISSN: 2775-4227
e-ISSN: 2775-4219

Subquery and Indexing sangat ketat yang metode *Subquery and Indexing* memberikan dampak peningkatan yang paling besar. Namun metode *Subquery* itu sendiri bergantung pada metode *Index and Table Partition* karena jika hanya menggunakan metode *Subquery*, hasilnya justru menurunkan performa *query*. Penggunaan metode *Parallel Query Execution* sedikit meningkatkan performa pada metode kombinasi dengan risiko peningkatan penggunaan sumber daya yang sulit untuk dimitigasi yang dapat menyebabkan *deadlock system* [28].

Dengan kondisi jumlah data yang sangat besar atau dikenal dengan *Large-Scale Data Processing*, dan tingkat kompleksitas susunan *query* yang sangat tinggi, peneliti membutuhkan kombinasi dari berbagai macam metode optimasi pada penelitian sebelumnya untuk mendapatkan hasil yang maksimal. Berdasarkan hal tersebut peneliti menemukan framework atau langkah yang paling efektif dalam menentukan penggunaan metode optimasi sesuai dengan kondisi dan ruang lingkup yang ada.



Gambar 23. Framework Penentuan Metode Optimasi

Berdasarkan Gambar 23, langkah pertama yang harus dilakukan adalah menentukan skala pengolahan data, yang dapat dikategorikan sebagai skala normal (kurang dari 1 juta baris) atau skala besar (lebih dari 1 juta baris data). Langkah kedua adalah menentukan skala kompleksitas *query* yang dapat dikategorikan sebagai skala normal (tidak ada atau sedikit fungsi *Join*) atau skala kompleks (banyak fungsi *Join*). Langkah ketiga adalah apakah peneliti memiliki kewenangan Database Administrator. Jika skala pengolahan data termasuk normal, maka metode optimasi yang digunakan cukup *Index*. Jika skala pengolahan data termasuk besar namun skala kompleksitas query normal, maka metode optimasi yang digunakan kombinasi *Index* dan *Table Partition.* Jika skala pengolahan data besar dan kompleksitas *query* kompleks atau rumit, maka metode optimasi yang digunakan kombinasi *Index*, *Table Partition*, dan *Subquery*. Untuk metode terakhir bersifat opsional, jika peneliti memiliki akses Database Administrator (DBA), metode *Parallel Query Execution* dapat ditambahkan dengan konsekuensi peningkatan penggunaan sumber daya.

Penelitian ini masih jauh dari kata sempurna yang artinya masih banyak keterbatasan dalam pengujian dan pendalaman metode optimasi *query*. Peneliti tidak memiliki akses sebagai SYSDBA Oracle sehingga hanya dapat menguji performa *query* dari perangkat PC yang terhubung dengan jaringan. Kondisi ini tentu saja dapat mempengaruhi tingkat akurasi hasil pengujian yang disebabkan oleh latensi kecepatan jaringan. Peneliti juga tidak memiliki akses menggunakan perangkat *Oracle Optimization* untuk mempercepat proses optimasi bentuk *query* yang ideal. Penelitian di masa yang akan datang dapat mengembangkan metode optimasi database dengan kecerdasan buatan berbasis AI sehingga proses optimasi dilakukan secara *real- time*.

5. KESIMPULAN

Penelitian ini menegaskan bahwa efektivitas metode optimasi dalam pengolahan *query* ditentukan oleh skala data, tingkat kompleksitas *query*, serta kewenangan dalam pengelolaan sumber daya sistem. Hasil penelitian menunjukkan bahwa metode kombinasi menghasilkan peningkatan efisiensi yang paling optimal, dengan *Subquery and Indexing* sebagai metode yang memiliki kinerja kompetitif, meskipun tetap bergantung pada *Index and Table Partition*. Selain itu, penggunaan *Parallel Query Execution* terbukti mampu meningkatkan performa lebih lanjut, meskipun disertai dengan peningkatan konsumsi sumber daya yang signifikan. Dari perspektif praktis, penelitian ini berhasil meningkatkan performa *query* perhitungan subsidi debitur pada database SIKP. Peningkatan performa ini mengurangi risiko gangguan layanan secara signifikan sehingga pemangku kebijakan dapat mencetak laporan secara *real-time*. Secara teoritis, temuan ini memperkaya pemahaman mengenai hubungan antar metode optimasi dalam konteks kompleksitas *query* dan *Large-Scale Data Processing*. Kontribusi penelitian ini terhadap ilmu pengetahuan terletak pada pengembangan framework pengambilan keputusan dalam optimasi query, yang berpotensi menjadi acuan bagi studi lanjutan serta pengembangan sistem basis data yang lebih efisien dan adaptif.

DAFTAR PUSTAKA

[1] Kementerian Keuangan, "Peraturan Menteri Keuangan Nomor 12 Tahun 2024 tentang Pedoman Penggunaan Sistem Informasi Kredit Program," Jakarta, Feb. 2024. Accessed: Mar. 19, 2025. [Online]. Available: https://peraturan.bpk.go.id/Download/338733/2024pmkeuangan012.pdf.pdf

[2] Mb. Hartanto, T. Muhammad Fawa, and D. P. Eko Hendro, "ANALISA KINERJA DATABASE DAN IMPLEMENTASI CACHE REDIS PADA WEB SERVICE LUMEN," Bandar Lampung, Oct. 2023. Accessed: Mar. 19, 2025. [Online]. Available: https://jurnal.umitra.ac.id/index.php/altek/article/view/1429/pdf

[3] M. Sughaturu Krishnappa, B. Mohan Harve, V. Jayaram, A. Nagpal, K. Kumar Ganeeb, and B. Shesharao Ingole, "ORACLE 19C SHARDING: A COMPREHENSIVE GUIDE TO MODERN DATA DISTRIBUTION," *International Journal of Computer Engineering and Technology (IJCET)*, vol. 15, no. 5, pp. 15–20, Sep. 2024, doi: 10.5281/zenodo.13880818.

[4] M. M. Rahman, S. Islam, M. Kamruzzaman, and Z. H. Joy, "ADVANCED QUERY OPTIMIZATION IN SQL DATABASES FOR REAL-TIME BIG DATA ANALYTICS," *ACADEMIC JOURNAL ON BUSINESS ADMINISTRATION, INNOVATION & SUSTAINABILITY*, vol. 4, no. 3, pp. 1-1–14, Jun. 2024, doi: 10.69593/ajbais.v4i3.77.

[5] S. Maesaroh, H. Gunawan, A. Lestari, M. S. A. Tsaurie, and M. Fauji, "Query Optimization In MySQL Database Using Index," *International Journal of Cyber and IT Service Management*, vol. 2, no. 2, pp. 104–110, Mar. 2022, doi: 10.34306/ijcitsm.v2i2.84.

[6] E. Witono, "Perbandingan Response Time Penggunaan Index, Views, dan Materialized Views Database Mysql," Jakarta, Mar. 2022. doi: http://dx.doi.org/10.30645/j-sakti.v6i1.463.

[7] Samidi, Fadly, Y. Virmansyah, R. Yulyanto Suladi, and A. Bambang Lesmana, "Optimasi Database dengan Metode Index dan Partisi Tabel Database Postgresql pada Aplikasi E-Commerce. Studi pada Aplikasi Tokopintar," Jakarta, Apr. 2022. doi: https://doi.org/10.31004/jptam.v6i1.3257.

[8] M. Samsul Anwar and N. F. Rozi, "Optimisasi Performa Akses Data dalam Grafana Menggunakan Indeks B-Tree MySQL," *Prosiding Seminar Implementasi Teknologi Informasi dan Komunikasi*, vol. 3, no. 2, pp. 286–292, Jul. 2024, doi: 10.31284/p.semtik.2024-2.6211.

[9] M. Kumar, T. K. Gupta, and D. U. Sarwe, "An Optimization of Bitmap Index Compression Technique in Bulk Data Movement Infrastructure," *IOP Conf Ser Mater Sci Eng*, vol. 1099, no. 1, p. 012074, Mar. 2021, doi: 10.1088/1757-899X/1099/1/012074.

[10] W. Grivin Mokodaser and M. Dwijayanti, "Implementasi Metode Indexing dan Penggunaan Subquery untuk Optimalisasi Database Rawat Jalan Rumah Sakit Menggunakan Mysql," *Cogito Smart Journal*, vol. 8, no. 2, pp. 335–345, Dec. 2022, doi: 10.31154/cogito.v8i2.415.335-345.

[11] J. Zhao, H. Zhang, and Y. Gao, "Efficient Query Re-optimization with Judicious Subquery Selections," *Proceedings of the ACM on Management of Data*, vol. 1, no. 2, pp. 1–26, Jun. 2023, doi: 10.1145/3589330.

[12] S. A. Rismawati, N. Zahira Ramadhani, E. T. Aswuri, and S. Mukaromah, "Prosiding Seminar Nasional Teknologi dan Sistem Informasi (SITASI) 2023 Surabaya," Surabaya, Sep. 2023. doi: https://doi.org/10.33005/sitasi.v3i1.395.

p-ISSN: 2775-4227
e-ISSN: 2775-4219

[13] R. Bagus, B. Sumantri, G. Subari, F. Mahardika, and H. Jayusman, "PERBANDINGAN EFISIENSI WAKTU PROSES PENGAKSESAN DATA ANTARA QUERY BERBENTUK JOIN DENGAN SUBSELECT," *Jurnal Manajemen Informatika & Komputerisasi Akuntansi*, vol. 7, no. 1, pp. 25–33, Apr. 2023, doi: 10.46880/jmika.Vol7No1.pp25-33.

[14] M. Nuriev, R. Zaripova, A. Potapov, and M. Kuznetsov, "Achieving new SQL query performance levels through parallel execution in SQL Server," in *E3S Web of Conferences*, Kazan: EDP Sciences, Dec. 2023, pp. 7–14. doi: 10.1051/e3sconf/202346004005.

[15] M. Ilba, "Parallel algorithm for improving the performance of spatial queries in SQL: The use cases of SQLite/SpatiaLite and PostgreSQL/PostGIS databases," *Science Direct*, vol. 155, Oct. 2021, doi: https://doi.org/10.1016/j.cageo.2021.104840.

[16] H. Zhang, D. G. Andersen, A. Pavlo, M. Kaminsky, L. Ma, and R. Shen, "Reducing the storage overhead of main-memory OLTP databases with hybrid indexes," in *Proceedings of the ACM SIGMOD International Conference on Management of Data*, San Francisco: Association for Computing Machinery, Jun. 2016, pp. 1567–1581. doi: 10.1145/2882903.2915222.

[17] Kementerian Koordinator Bidang Perekonomian, "Peraturan Menteri Koordinator Bidang Perekonomian Nomor 1 Tahun 2022 Tentang Pedoman Pelaksanaan Kredit Usaha Rakyat," 2023.

[18] A. Silberschatz, H. F. Korth, and S. Sudarshan, *Database System Concepts*, 6th ed. New York: McGraw-Hill Education, 2010.

[19] Q. Zhang, X. Gong, H. A. Khan, J. Wei, and Y. Ren, "A Novel Index-Organized Storage Model for Hybrid DRAM-PM Main Memory Database Systems," Nov. 18, 2024, *Shanghai*. doi: 10.21203/rs.3.rs-5286510/v1.

[20] A. Murugan and J. Vijayalakshmi, "Detecting Multi-Block Double Spent Transaction Based On B-Tree Indexing," *INTERNATIONAL JOURNAL OF SCIENTIFIC & TECHNOLOGY RESEARCH*, vol. 9, p. 2, Feb. 2020, [Online]. Available: www.ijstr.org

[21] H. Roh, W.-C. Kim, S. Kim, and S. Park, "A B-Tree index extension to enhance response time and the life cycle of flash memory," *sciencedirect*, vol. 179, no. 18, pp. 3136–3161, Aug. 2009.

[22] R. Bayer, "The universal B-tree for multidimensional indexing: General concepts," *Springer*, vol. 1274, Jul. 2005.

[23] Z. Jalilibal, A. Amiri, P. Castagliola, and M. B.C. Khoo, "Monitoring the coefficient of variation: A literature review," *Science Direct*, vol. 161, Nov. 2021.

[24] Microsoft Learn, "Subqueries (SQL Server)," 2024. Accessed: Mar. 21, 2025. [Online]. Available: https://learn.microsoft.com/id-id/sql/relational-databases/performance/subqueries?view=sql-server-ver16

[25] G. Fritchey, "SQL Query Performance Tuning," *Research Gate*, pp. 1–15, Aug. 2014.

[26] T. Taipalus, "The effects of database complexity on SQL query formulation," *Science Direct*, vol. 165, Jul. 2020.

[27] Oracle Corporation, "Parallel Execution Fundamentals," 2009, *Oracle Corporation*.

[28] Y. Wang, M. Li, H. Dai, K. B.Kent, K. Ye, and C. Xu, "Deadlock Avoidance Algorithms for Recursion-Tree Modeled Requests in Parallel Executions," *EEE Transactions on Computers*, vol. 71, no. 9, pp. 2073–2087, Sep. 2022.