JITET (Jurnal Informatika dan Teknik Elektro Terapan)
Vol. 13 No. 3S1, pISSN: 2303-0577 eISSN: 2830-7062 http://dx.doi.org/10.23960/jitet.v13i3S1.8012

OPTIMASI PROSES DATA *WAREHOUSE* MENGGUNAKAN PARTISI DAN *INDEXING* PADA POSTGRESQL UNTUK MENINGKATKAN PERFORMA *QUERY*

Luthfi Sifa Khaerunnisa[1*], M. Ramadhan Syahrul Al Qodr[2], Juliana Widianti Dwi Putri[3], Joyce Rosita Firdaus[4], Chaerur Rozikin[5]

[1,2,3,4,5] Universitas Singaperbangsa Karawang; Jl. HS.Ronggo Waluyo, Puseurjaya, TelukjambeTimur, Karawang, Jawa Barat 41361;Telp. (0267) 641177

Keywords:
Data Warehouse, Optimization, Partition, Indexing, PostgreSQL.

Corespondent Email:
luthfisifa@gmail.com

Abstrak. Optimasi *query* pada data *warehouse* sangatlah krusial terutama dengan pertumbuhan volume dan kompleksitas analitik. Seiring dengan pertumbuhan volume data, performa *query* pada data *warehouse* seringkali mengalami degradasi yang menghambat proses analisis dan pengambilan keputusan. Penelitian ini bertujuan untuk mengoptimalkan performa data *warehouse* dengan menganalisis dampak dari teknik partisi dan *indexing t*erhadap kinerja *query* pada PostgreSQL 17.6 menggunakan dataset "Online Retail II" yang telah dibersihkan dan dibuat *star schema*. Eksperimen dilakukan secara komparatif yaitu tanpa optimasi, partisi saja, *indexing s*aja, dan partisi dengan *indexing.* Kinerjanya diukur menggunakan dua matriks utama yaitu waktu eksekusi dan *I/O*. Penelitian menunjukkan bahwa kombinasi partisi dan *indexing a*dalah strategi paling superior, yang mampu mengurangi waktu eksekusi *query* hingga 55.13% sedangkan partisi efektif untuk *query* berbasis rentang waktu dan *indexing u*ntuk akses selektif. Perancangan data *warehouse* berbasis PostgreSQL sebaiknya memadukan partisi yang selaras antara pola waktu dengan *indexing p*ada kolom filter/*join*.

Abstract. *Optimizing query performance in data warehouses is crucial as data volumes and analytical complexity continue to grow. As datasets scale, query performance often degrades, impeding timely analysis and decision-making. This study aims to enhance data warehouse performance by analyzing the impact of partitioning and indexing techniques on the query performance of* PostgreSQL *17.6, using the cleaned Online Retail II dataset modeled in a star schema. We conduct comparative experiments: no optimization, partitioning only, indexing only, and partitioning combined with indexing. Performance is evaluated using two primary metrics: execution time and I/O. The results demonstrate that the combined strategy is the most effective, reducing query execution time by up to 55.13%. Partitioning is particularly advantageous for time-range queries, while indexing benefits selective access. Accordingly,* PostgreSQL*-based data warehouse designs should integrate time-aligned partitioning with indexes on filter and join columns*.

1. PENDAHULUAN

Dengan dimulainya era digitalisasi, data yang dihasilkan oleh perusahaan, pemerintah, organisasi dan lainnya meningkat pesat secara eksponensial. Jika data ini diolah dengan benar maka akan menjadi aset yang berharga terutama

untuk pengambilan keputusan strategis. Dengan adanya data *warehouse* ini dapat menjadi solusi untuk mengkonsolidasi data dari berbagai sistem operasional dalam satu repositori pusat dengan akses cepat terhadap data historis dan agregat yang dirancang khusus untuk dianalisis dan dibuat laporannya supaya mendukung *Business Intelligence (BI)* [1].

Data *warehouse* menampung volume data historis yang sangat besar sehingga performa *query* menjadi faktor krusial. Tentu saja pengelolaan data *warehouse* ini mempunyai tantangannya tersendiri, yaitu menjaga performa *query* supaya optimal seiring dengan bertambahnya volume data. Tanpa optimasi yang tepat eksekusi *query* bisa berjalan lambat dan *query* yang lambat bukan hanya menghambat produktivitas tetapi juga dapat menunda penyajian informasi bagi pengambil keputusan yang penting secara tepat waktu.

Platform basis data *open-source* untuk membangun data *warehouse* yang paling populer salah satunya adalah PostgreSQL karena keandalan, kekayaan fitur dan ekstensibilitasnya. PostgreSQL merupakan sistem manajemen basis data relasional (*Relational Database Management System* - RDBMS) yang canggih dengan dukungan SQL yang luas dan memiliki kemampuan dalam menangani beban kerja yang berat [2]. Penerapan partisi dan *indexing y*ang tepat di PostgreSQL berpotensi mengakselerasi performa *query* secara signifikan, namun implementasi data *warehouse* berskala besar pada PostgreSQL sering menghadapi masalah dalam performa sehingga diperlukan strategi dan evaluasi mendalam agar memberikan hasil optimal dan tidak berdampak negatif jika diterapkan sembarangan [3].

Teori dasar dalam desain data *warehouse* adalah pemodelan dimensional dengan menggunakan skema bintang (*star schema*) untuk menyederhanakan struktur data menjadi tabel fakta dan dimensi [4]. Seiring bertambahnya volume data pada tabel fakta, performa *query* lama-kelamaan akan mengalami degradasi yang menyebabkan response time melambat dan menjadi *bottleneck* utama dalam analisis data yang efektif.

Untuk menjamin data *warehouse* memberikan respon yang cepat berbagai teknik optimasi perlu dilakukan. Terdapat dua teknik yang biasanya digunakan untuk meningkatkan performa *query* pada *database beru*kuran besar yaitu partisi (*partition) dan* pengindeksan (*indexing).* Partisi merupakan proses dalam membagi sebuah tabel besar menjadi beberapa bagian yang lebih kecil supaya lebih mudah dikelola [5]. Sedangkan *indexing m*erupakan struktur data tambahan yang memungkinkan *direct access* ke *b*aris yang dicari tanpa harus melakukan *sequential scan* seluruh tabel sehingga pencarian data dilakukan secara lebih cepat [6].

Penelitian yang dilakukan oleh Hashem O, Rahourma K, dkk [3] menunjukkan bahwa peningkatan kinerja data *warehouse* yang kelebihan beban dapat dicapai dengan menerapkan *indexing d*an partisi, di samping teknik lain seperti kompresi dan *materialized view*, untuk mempercepat waktu pengambilan data dan eksekusi *query*. Pada tingkat aplikasi, penelitian yang dilakukan oleh Azizi B, Pratama A, dkk [7] menunjukkan bahwa pengelolaan koneksi basis data yang efisien (menggunakan pola desain *Singleton*) mampu menurunkan rata-rata waktu eksekusi secara drastis dibanding metode konvensional. Hal ini dapat menegaskan bahwasannya optimasi baik di level arsitektur data *warehouse* maupun di level aplikasi berperan penting dalam meningkatkan performa sistem secara keseluruhan.

Meski partisi dan *indexing m*erupakan teknik yang sudah matang tetapi sebagian besar literatur membahas keduanya secara terpisah sehingga terdapat kesenjangan penelitian yang secara spesifik menganalisis dan mengukur dampak dari strategi partisi dan *indexing s*ecara bersamaan dalam data *warehouse* yang dibangun diatas PostgreSQL.

Penelitian ini berfokus pada optimasi proses data *warehouse* menggunakan teknik partisi dan *indexing p*ada data *warehouse* yang dibangun pada platform PostgreSQL untuk meningkatkan performa *query*. Tujuan yang ingin dicapai pada penelitian ini adalah menganalisis sejauh mana penerapan partisi pada tabel data *warehouse* dan penambahan

*indexing p*ada kolom-kolom kunci dapat mempercepat waktu respon *query*. Dengan demikian penelitian ini diharapkan membuktikan secara kuantitatif efektivitas partisi dan *indexing,* serta menghasilkan panduan praktis untuk membangun data *warehouse* PostgreSQL yang lebih cepat dan efisien.

2. TINJAUAN PUSTAKA

2.1. Studi Literatur

Studi literatur adalah metode penelitian yang memanfaatkan sumber-sumber tertulis pada penelitian sebelumnya untuk memperoleh informasi dan analisis terkait topik penelitian. Penelitian sebelumnya telah menjelajahi optimasi data *warehouse* dan menunjukkan bahwa pemilihan strategi *indexing y*ang tepat dapat meningkatkan performa *query* analitik secara drastis. Studi yang dilakukan oleh Mostafa J, Wehbi S dkk [8] menunjukkan bahwa desain arsitektur basis data *time-series* yang memanfaatkan mekanisme partisi atau *chunking* mampu memberikan peningkatan kinerja signifikan dibandingkan pendekatan tanpa partisi, terutama pada beban kerja *query* berskala besar. Referensi yang digunakan meliputi jurnal, artikel ilmiah, dan buku terkait optimasi data *warehouse*, teknik partisi dan *indexing p*ada data *warehouse*. Penelitian ini berupaya menggabungkan kedua pendekatan partisi dan *indexing s*ecara spesifik pada platform PostgreSQL untuk memberikan panduan komprehensif dalam optimasi data *warehouse*.

2.2. Data *Warehouse*

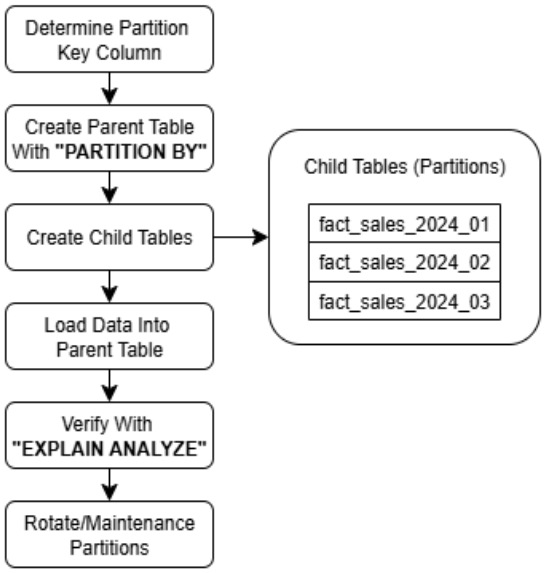
Data *warehouse* merupakan sebuah sistem repositori terintegrasi yang dirancang untuk mendukung analisis historis dan pengambilan keputusan. Memiliki empat karakteristik utama yaitu berorientasi subjek, terintegrasi, mempunyai rentang waktu dan bersifat *non-volatil*. Dalam data *warehouse* arsitektur yang umumnya sering digunakan adalah skema bintang (*star schema*) atau skema kepingan salju (*snowflake schema*), pada kedua skema tersebut terdiri dari tabel fakta *(fact table) yan*g berada di pusat dan juga tabel dimensi *(dimension tables)* yang berada di sekeliling tabel fakta [1].

2.3. PostgreSQL untuk Data *Warehouse*

Salah satu RDBMS (*Relational Database Management System*) *open-source* yang paling canggih dan andal adalah PostgreSQL. Dengan dukungan terhadap fitur SQL yang modern dan kemampuannya menangani beban kerja yang kompleks membuat PostgreSQL menjadi pilihan yang tepat untuk implementasi data *warehouse* [2]. Fitur seperti partisi dan berbagai jenis *index* memberikan fleksibilitas bagi administrator basis data untuk melakukan optimasi performa.

2.4. Partisi pada Data *Warehouse*

Memecah tabel dan *index* yang berukuran sangat besar menjadi bagian-bagian yang lebih kecil berdasarkan kriteria tertentu disebut dengan teknik partisi. Dalam konteks data *warehouse* partisi merupakan fitur esensial yang berguna untuk meningkatkan kinerja dan skalabilitas sistem. Terdapat berbagai strategi partisi pada PostgreSQL, strategi partisi memiliki keuntungan utama yaitu dapat mengurangi jumlah data yang perlu dibaca dari *disk* dengan memindai partisi-partisi yang relevan dengan kondisi *query*. Selain itu partisi memungkinkan eksekusi paralel yang efektif [9], setiap partisi bisa diproses secara bersamaan oleh beberapa pekerja dan hasilnya dapat digabung sehingga *query* agregasi besar dapat berjalan lebih cepat seiring bertambahnya core CPU.

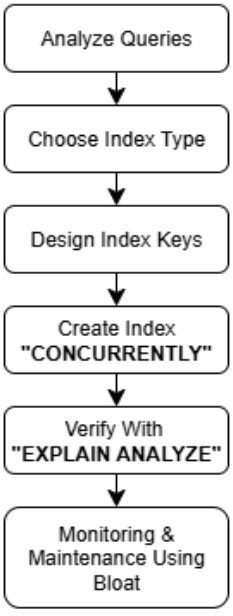


Gambar 1. Alur Partisi

Partisi dimulai dengan menentukan kolom kunci partisi (biasanya tanggal), kemudian membuat tabel induk dengan "*PARTITION BY",* dilanjutkan membuat tabel anak *(child tables)* sesuai rentang nilai yang ditentukan, setelah itu memuat data ke tabel induk *(parent)* sehingga otomatis diarahkan ke partisi *(child tables)*, terakhir memverifikasi dengan "*EXPLAIN ANALYZE*" untuk memastikan *pruning* berjalan dan melakukan rotasi/*maintenance* partisi secara berkala.

2.5. *Indexing* pada Data *Warehouse*

Indexing adalah teknik untuk mempercepat operasi *join* dengan pembuatan struktur data tambahan menggunakan kolom *foreign key* di tabel fakta dan *primary key* di tabel dimensi. Pada PostgreSQL *index* disimpan secara terpisah dari tabel utama dan berisi *pointer* ke lokasi data, *query* dijalankan dengan kondisi pada kolom yang ter-*index* lalu mesin *database dapa*t menggunakan *index ters*ebut untuk langsung menuju baris-baris yang memenuhi kondisi. Tanpa *index, database engine* harus melakukan pemindaian penuh *(full table scan)* untuk menemukan baris data yang dicari [10].



Gambar 2. Alur Indexing

Secara konsep *index* bekerja seperti *index* buku yang memungkinkan sistem basis data menemukan informasi yang dicari tanpa harus membaca halaman demi halaman secara berurutan. Alur *indexing* dimulai dari menganalisis *query* untuk menentukan kolom penting, lalu memilih tipe *index* sesuai dengan pola data, langkah selanjutnya mendesain kunci *index,* lalu membuat *index* dengan "*CREATE INDEX [CONCURRENTLY]"*, kemudian memverifikasi dengan "*EXPLAIN ANALYZE"*, serta memantau penggunaan dan *bloat* agar tidak terjadi *over-indexing.*

2.6. Pengukuran Performa

Tentu pengukuran performa pada data *warehouse* sangatlah penting untuk memahami serta mengevaluasi efisiensi dan kecepatan sistem khususnya dalam konteks optimasi *query* analitik yang kompleks pada data *warehouse* PostgreSQL. Dalam penelitian ini pengukuran dilakukan untuk menilai dua matriks utama yaitu waktu eksekusi *query*, jumlah blok yang diakses (*I/O*) dan pemanfaatan memori. Pengukuran performa dalam penelitian ini dirancang dengan sistematis untuk mengukur dan memvalidasi bagaimana dampak dari setiap teknik optimasi, baik secara terpisah maupun gabungan. Fokus utama penelitian ini adalah untuk mengkuantifikasi peningkatan efisiensi eksekusi *query* dengan menganalisis metrik-metrik kunci yang disediakan langsung oleh *database engine*.

Data yang dikumpulkan dari "*EXPLAIN ANALYZE*" pada setiap skenario dianalisis secara komparatif. Waktu eksekusi digunakan untuk meng*hit*ung peningkatan performa secara persentase. Sementara itu, analisis rencana eksekusi statistik *I/O* digunakan untuk memberikan justifikasi teknis. Parameter seperti *buffers hit* dengan *read* dianalisis untuk memahami proporsi data yang diakses dari *cache* dibandingkan dari *disk*, karena hal ini sangat menentukan kecepatan respon *query* [11]. Pengukuran performa berbasis matriks waktu eksekusi dan *I/O* merupakan pendekatan yang terstandardisasi untuk menilai efektivitas teknik optimasi dalam data *warehouse* relasional.

3. METODE PENELITIAN .

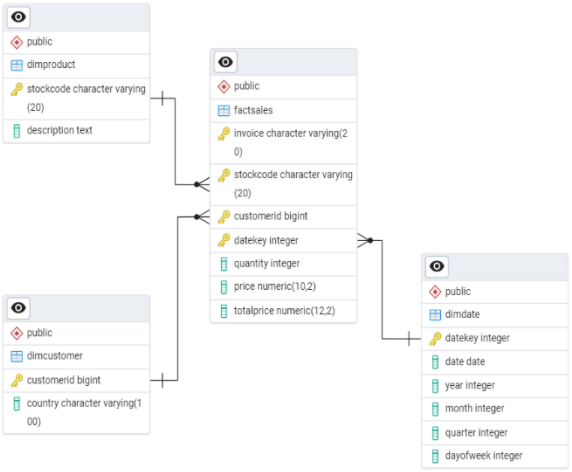
3.1. Perancangan Penelitian

Rancangan penelitian adalah kerangka sistematis yang digunakan oleh peneliti untuk melakukan penelitiannya [12]. Penelitian ini

menggunakan pendekatan eksperimen kuantitatif yang dirancang menggunakan studi komparatif empat skenario *(Four-Scenario Comparative Design),* desain ini dipilih supaya dapat mengisolasi dan mengukur pengaruh dari setiap teknik optimasi secara individu serta dampaknya saat dikombinasikan. Bertujuan untuk membandingkan dampak partisi dan atau *indexing t*erhadap performa *query* yang dijalankan pada data *warehouse* PostgreSQL. Berikut merupakan rancangan arsitektur data *warehouse* yang dioptimalkan dengan skema bintang (*star schema*), perbandingan optimasi, serta pengukuran performa yang lengkap. Pada penelitian ini digunakan data yang diambil dari **Online Retail II dataset** yang tersedia di **UCI Machine Learning Repository**. Dataset yang digunakan berisi transaksi penjualan *e-commerce*, yang mencakup informasi seperti ID transaksi, ID pelanggan, ID produk, kuantitas, harga, dan total transaksi. Sebelum digunakan, data melalui proses pembersihan untuk menghapus entri yang tidak relevan atau duplikat, serta memperbaiki kesalahan format dan data yang hilang.

3.1.1. Pembuatan Skema Data *Warehouse* Berbasis *Star Schema*

Arsitektur ini menggunakan *star schema* untuk menjadi pondasinya, dipilih karena strukturnya yang sederhana dan dioptimalkan untuk *query* analitik yang cepat. *Star schema* merupakan model basis data multidimensi yang terdiri dari satu tabel fakta besar yang terhubung dengan beberapa tabel dimensi.



Gambar 3. Star Schema

- Tabel Fakta (*Fact Table):*
  Tabel ini berisi transaksi penjualan yang akan dianalisis dan menghubungkan dengan tabel dimensi yang relevan melalui *foreign key* dengan *query* sebagai berikut.

```
CREATE TABLE FactSales (
    Invoice VARCHAR(20),
    StockCode
VARCHAR(20) REFERENCES
DimProduct(StockCode),
    CustomerID BIGINT
REFERENCES
DimCustomer(CustomerID),
    DateKey INT
REFERENCES
DimDate(DateKey),
    Quantity INT,
    Price NUMERIC(10,2),
    TotalPrice
NUMERIC(12,2),
    PRIMARY KEY
(Invoice, StockCode,
CustomerID, DateKey)
);
```

- Tabel Dimensi:
  - dim_date
    Tabel dimensi yang berfungsi untuk memberikan informasi terkait tanggal yang digunakan untuk mengelompokkan dan menganalisis transaksi berdasarkan waktu dengan *query* sebagai berikut.

```
CREATE TABLE
DimDate (
    DateKey INT
PRIMARY KEY,
    Date DATE,
    Year INT,
    Month INT,
    Quarter INT,
    DayOfWeek INT
);
```

  - dim_customer
    Tabel dimensi yang menyimpan informasi terkait

pelanggan yang melakukan transaksi dengan *query* sebagai berikut.

```
CREATE TABLE
DimCustomer (
    CustomerID
BIGINT PRIMARY KEY,
    Country
VARCHAR(100)
);
```

- dim_product
  Tabel dimensi yang menyimpan informasi tentang product yang dijual dengan *query* sebagai berikut.

```
CREATE TABLE
DimProduct (
    StockCode
VARCHAR(20) PRIMARY
KEY,
    Description
TEXT
);
```

3.1.2. Implementasi Studi Komparatif

Untuk membuktikan efektivitas optimasi, arsitektur ini dirancang untuk menguji dalam empat skenario yang berbeda. Optimasi yang diterapkan adalah partisi dan *indexing*. Empat skenario pengujiannya adalah sebagai berikut:

- Tanpa Optimasi (*Baseline*)
  Pada Skenario ini akan digunakan tabel fakta (FactSales) tanpa partisi dan *index*. *Query* akan dijalankan pada seluruh tabel tanpa adanya optimasi apapun.
- Dengan Partisi
  Di skenario ini partisi akan diterapkan pada tabel fakta (FactSales) untuk membagi data berdasarkan rentang waktu dengan *query* sebagai berikut.

```
ALTER TABLE fact_sales
RENAME TO
fact_sales_prepartitione
d

CREATE TABLE fact_sales
(
    invoice VARCHAR(50),
    stockcode
VARCHAR(50),
    customerid INTEGER,
    datekey INTEGER,
    quantity INTEGER,
    price NUMERIC,
    totalprice NUMERIC
) PARTITION BY RANGE
(datekey);

-- Tabel partisi (child)
CREATE TABLE
fact_sales_2009
PARTITION OF fact_sales
FOR VALUES FROM
(20090101) TO
(20100101);

CREATE TABLE
fact_sales_2010
PARTITION OF fact_sales
FOR VALUES FROM
(20100101) TO
(20110101);

CREATE TABLE
fact_sales_2011
PARTITION OF fact_sales
FOR VALUES FROM
(20110101) TO
(20120101);

INSERT INTO fact_sales
SELECT * FROM
fact_sales_prepartitione
d;
```

- Dengan *Indexing*
  Pada skenario *indexing a*kan ditambahkan *index pada* kolom yang sering digunakan dalam pencarian menggunakan *join*. *Index* akan dibuat pada kolom customerid, stockcode dan datekey pada tabel fakta (FactSales) dengan *query* sebagai berikut.

```
CREATE INDEX
idx_fact_sales_customer_
prepartitioned ON
fact_sales_prepartitione
d(customerid);
CREATE INDEX
idx_fact_sales_product_p
repartitioned ON
fact_sales_prepartitione
d(stockcode);
CREATE INDEX
idx_fact_sales_date_prep
artitioned ON
fact_sales_prepartitione
d(datekey);
```

- Kombinasi Partisi dan *Indexing*

  Skenario ini menggabungkan partisi dan *indexing y*ang digunakan secara bersamaan. Data di partisi berdasarkan tahun seperti pada langkah sebelumnya dan *index* dibuat pada kolom yang sering digunakan, dengan *query* sebagai berikut.

```
CREATE INDEX
idx_fact_sales_customer
ON
fact_sales(customerid);
CREATE INDEX
idx_fact_sales_product
ON
fact_sales(stockcode);
CREATE INDEX
idx_fact_sales_date ON
fact_sales(datekey);
```

3.1.3. Pengukuran Performa *Query*

Digunakan replikasi eksperimen dengan menguji ke-empat skenario dan menjalankan *query* yang sama berulang kali pada masing-masing skenario yang ada. Dalam mengukur performa akan digunakan dua matriks utama yaitu waktu eksekusi *(execution time)* untuk menunjukkan seberapa cepat *query* selesai dieksekusi dan rencana eksekusi (*query plan)* untuk memahami kenapa sebuah *query* berjalan cepat atau lambat. Selain itu akan diamati juga *buffers* untuk melihat memori yang digunakan.

3.2. Sumber dan Pengumpulan data

Sumber dan pengumpulan data merupakan proses untuk memperoleh informasi yang diperlukan untuk penelitian [13]. Sumber data terperinci yang digunakan pada penelitian ini seperti struktur skema data *warehouse* yang dibangun serta proses transformasi dan pemuatan data untuk pengujian.

3.2.1. Sumber Data

Data yang digunakan pada penelitian ini adalah dataset Online Retail II yang diambil dan tersedia secara publik di UCI Machine Learning Repository. Dataset ini berisi data transaksi nyata dari sebuah toko ritel daring yang berbasis di Inggris selama periode dua tahun. Untuk keperluan penelitian ini, digunakan versi data yang telah melalui proses pembersihan awal dan disediakan dalam format CSV. Data yang telah dibersihkan disimpan dalam format CSV dan dipisahkan menjadi tabel-tabel dalam struktur *star schema* yang telah disebutkan sebelumnya. Data kemudian dimuat ke dalam PostgreSQL untuk analisis lebih lanjut dalam eksperimen ini.

3.2.2. Struktur Data

Dataset yang sudah melalui proses pembersihan dan telah dipisahkan akan disusun menjadi beberapa tabel ke dalam model *star schema* untuk membangun lingkungan data *warehouse* sehingga mudah dianalisis. Struktur data yang digunakan dalam penelitian ini terdiri dari tabel dimensi dan fakta.

- Dimensi Pelanggan (DimCustomer):
  - CustomerID: Identifikasi unik untuk pelanggan.
  - Country: Negara tempat tinggal pelanggan.
- Dimensi Produk (DimProduct):
  - StockCode: Kode unik untuk setiap produk.
  - Description: Deskripsi produk.
- Dimensi Tanggal (DimDate):
  - DateKey: Kode unik untuk setiap tanggal (dalam format YYYYMMDD).
  - Date: Tanggal transaksi.
  - Year: Tahun transaksi.
  - Month: Bulan transaksi.
  - Quarter: Kuartal transaksi.
  - DayOfWeek: Hari dalam minggu.

- Tabel Fakta Penjualan (FactSales):
    - Invoice: Nomor invoice untuk transaksi.
    - StockCode: Kode produk yang dijual.
    - CustomerID: ID pelanggan yang melakukan transaksi.
    - DateKey: Kode tanggal transaksi.
    - Quantity: Jumlah produk yang terjual.
    - Price: Harga produk per unit.
    - TotalPrice: Total nilai transaksi.

3.3. Lingkungan Penelitian

Lingkungan penelitian merupakan kondisi fisik dan sosial dimana penelitian tersebut dilakukan, ini mencakup fasilitas uang mendukung serta faktor sosial [14]. Penelitian ini dilakukan pada lingkungan komputasi yang terkontrol untuk memastikan bahwa hasil pengujian kinerja bersifat konsisten, valid, dan dapat direplikasi. Dengan spesifikasi perangkat keras, perangkat lunak, dan konfigurasi basis data yang digunakan dirinci di bawah ini.

- CPU : AMD Ryzen 5 5500U with Radeon Graphics (2.1 GHz, 6 Cores, 12 Th*read*s)
- RAM : 16 GB DDR4 3200MHz
- Penyimpanan : 512 GB Solid State Drive (SSD)
- VGA : AMD Radeon™ Graphics (Integrated)
- OS : Windows 10 (Versi 10.0.19045)
- PostgreSQL : 17.6

Konfigurasi pada PostgreSQL digunakan untuk menyesuaikan dengan lingkungan eksperimen untuk memastikan pengujian berjalan sesuai dengan rencana eksekusi yang dianalisis dan untuk menjaga konsistensi, beberapa parameter konfigurasi diatur secara spesifik.

- Max_parallel_workers_per_gather : 2
- shared_*buffers* : 128MB
- work_mem : 4MB
- track_io_timing : on

3.4. *Query* Uji

Dalam konteks *database, query* uji merujuk pada serangkaian *query* SQL yang telah dirancang untuk menguji sebuah kinerja sistem opetimasi *query* [15]. Untuk mengevaluasi dan mengukur dampak performa optimasi keempat skenario dari teknik partisi dan *indexing* dibuatlah *query* uji untuk merepresentasikan beban kerja analitik yang umum.

```
EXPLAIN (ANALYZE, BUFFERS)
SELECT customerid,
SUM(totalprice)
FROM fact_sales
WHERE datekey BETWEEN 20110101
AND 20111231
GROUP BY customerid;
```

Tujuan dan fungsi *query* ini adalah untuk menghitung total nilai belanja (totalprice) yang diakumulasikan oleh setiap pelanggan (customerid) selama periode tahun 2011. Dengan menerapkan *query* yang identik di semua skenario, penelitian dapat memastikan bahwa perbandingan performa dilakukan secara adil, sehingga kesimpulan yang ditarik menjadi valid dan dapat diandalkan.

4. HASIL DAN PEMBAHASAN

4.1. Hasil Eksperimen Komparatif

Hasil dari penelitian ini untuk masing-masing skenario adalah sebagai berikut:

4.1.1. Tanpa Optimasi (*Baseline*)

Skenario yang pertama merepresentasikan konfigurasi tanpa optimasi apapun dan menjadi titik acuan (*baseline*) untuk perbandingan optimasi yang lain.

- Waktu Eksekusi : 180.896 ms
- *I/O* : Pembacaan dari *disk* (*read*=5967), sebagian besar data diakses dari *cache* (*hit*=641)

Pada skenario ini PostgreSQL melakukan parallel *sequential scan* kepada seluruh tabel fact_sales_pre*partitioned* sehingga *database engine harus* membaca keseluruhan tabel fakta secara sekuensial. Hasil ini menunjukkan bahwa seluruh tabel (data) dibaca dan dipindai lalu filter diterapkan setelah pemindaian penuh yang menyebabkan banyak data yang tidak relevan ikut diproses. Dengan adanya 5967 blok data yang harus dibaca dan dipindai secara penuh membuat waktu eksekusi lebih lama dan terjadi banyak *I/O* karena data harus dibaca dari

disk. Pada skenario ini pembacaan dan pemindaian menjadi *bottleneck utam*a.

4.1.2. Dengan Partisi

Skenario ini direpresentasikan dengan mempartisi/ membagi tabel fakta berdasarkan tahun.

- Waktu Eksekusi : 157.823 ms
- *I/O* : Pembacaan dari *disk* (*read*=0), sebagian besar data diakses dari *cache* (*hit*=3106)

Dalam skenario ini dilakukan partisi pada data berdasarkan tahun. Dengan menggunakan mekanisme *partition pruning* ini memungkinkan PostgreSQL hanya membaca/ memindai partisi yang relevan dan mengabaikan partisi data dari tahun lain sepenuhnya, sehingga *I/O disk* berkurang dengan sangat signifikan dan dapat mengakses semua data dari *cache*. Waktu eksekusi pada skenario kedua jauh lebih cepat dibandingkan skenario *baseline* karena pada skenario kedua hanya memproses data yang relevan. Semua data yang dibutuhkan untuk *query* berhasil ditemukan di dalam *cache* memori PostgreSQL sehingga tidak ada satupun operasi baca dari *disk* yang terjadi dan itu merupakan alasan utama terjadinya peningkatan performa.

4.1.3. Dengan *Indexing*

Skenario kali ini akan menguji efektivitas *indexing p*ada kolom datekey pada tabel tunggal yang besar.

- Waktu Eksekusi : 145.475 ms
- *I/O* : Pembacaan dari *disk* (*read*=2504), sebagian besar data diakses dari *cache* (*hit*=1132)

Index dibuat pada kolom datekey menggunakan parallel *index scan* untuk mencari data berdasarkan *index* yang ada. *Engine* tidak lagi membaca seluruh tabel, melainkan menggunakan struktur B-Tree *index* untuk menemukan lokasi baris data secara efisien. Meskipun *index* dapat mempercepat waktu eksekusi pencarian dibandingkan dengan skenario *baseline*, tetapi *engine* tetap harus mengambil blok-blok data dari *I/O disk* setelah lokasinya ditemukan karena tidak semua data terakomodasi dalam *cache*. Jumlah bacaan *disk* berkurang lebih dari setengah dibandingkan skenario *baseline* yang menyebabkan skenario ini menjadi lebih cepat.

4.1.4. Kombinasi Partisi dan *Indexing*

Pada skenario terakhir diterapkan kedua teknik optimasi secara bersamaan yaitu teknik partisi dan juga teknik *indexing.*

- Waktu Eksekusi : 81.169 ms
- *I/O* : Pembacaan dari *disk* (*read*=0), sebagian besar data diakses dari *cache* (*hit*=3106)

Skenario ini merupakan strategi yang bagus karena menghasilkan optimasi paling terbaik. PostgreSQL dapat *partition pruning* yang langsung mengisolasi pemindaian hanya pada partisi. Kemudian, di dalam partisi yang kecil tersebut, tetap dilakukan parallel *sequential scan* untuk mengakses data yang relevan dengan sangat cepat. Semua data diakses dari *cache* dan tidak diperlukan adanya pembacaan *disk*. Waktu eksekusi berkurang hampir dua kali lipat dibandingkan skenario tanpa optimasi (*baseline*).

4.2. Dampak Penerapan Teknik Optimasi

4.2.1. Dampak Partisi

Penerapan partisi terbukti sangat efektif untuk mengurangi waktu eksekusi dan meningkatkan efisiensi *I/O* karena hanya memindai partisi yang relevan berdasarkan rentang waktu yang ditentukan. Pada skenario kedua saat hanya menggunakan partisi sebagai teknik optimasi menunjukan waktu eksekusi yang lebih cepat dibandingkan skenario pertama yaitu tanpa partisi dan tanpa *index (baseline*), semua data yang diakses saat dilakukan partisi berasal dari *cache* yang menunjukkan manfaat besar dari *partition pruning.*

4.2.2. Dampak *Indexing*

*Indexing s*angat berpengaruh dalam mempercepat pemberian data meskipun ada *I/O disk* yang tetap terjadi. Ini menunjukkan bahwa meskipun *index* dapat mempercepat pencarian tetapi PostgreSQL tetap perlu membaca sebagian data dari *disk*. Dengan menambahkan *index* menggunakan parallel *index scan dapa*t mengurangi waktu eksekusi dibandingkan dengan skenario *baseline*, tetapi harap diingat bahwa skenario ini tidak dapat mengeliminasi seluruh *I/O disk* pada dataset besar.

4.2.3. Dampak Penerapan Partisi dan *Indexing*

Dengan menggunakan teknik kombinasi antara partisi dan *indexing* dapat memberikan peningkatan performa yang signifikan. Waktu eksekusi turun jauh dan seluruh data dapat diakses melalui *cache* tanpa ada pembacaan *disk*. Partisi dapat mengurangi jumlah data yang perlu di proses dan *indexing* dapat memungkinkan PostgreSQL mengakses data secara lebih cepat dalam setiap partisi. Keduanya saling melengkapi sehingga berhasil memberikan hasil terbaik.

4.3. Analisis Kompratif

Perbandingan hasil optimasi dari keempat skenario menunjukkan dampak dari masing-masing teknik optimasi.

Skenario Pengujian	Waktu Eksekusi	Peningkatan Performa	Blok I/O Disk
Baseline	180.90ms	0%	5967
Partisi	157.82ms	12.76%	0
Indexing	145.48ms	19.58%	2504
Partisi + Index	81.17ms	55.13%	0

Tabel 1. Hasil Optimasi Keempat Skenario

Dari data yang ada pada tabel diatas dapat disimpulkan poin pentingnya yang pertama manfaat terbesar partisi adalah mempersempit set data sehingga muat dalam *cache* memori yang secara efektif dapat menghilangkan latensi akibat bacaan *disk*. *Indexing s*aja terbukti lebih efektif dari pada partisi saja dalam waktu eksekusi, hal ini disebabkan oleh *index scan yang* secara fundamental lebih efisien dalam menemukan data spesifik dibandingkan *sequential scan* meskipun masih terhambat oleh *I/O disk*. Temuan utama pada penelitian ini adalah peningkatan performa sebesar 55.13% yang jauh lebih besar daripada teknik optimasi yang lain yang dilakukan secara terpisah. Ini dapat membuktikan bahwa kombinasi keduanya bukanlah aditif, melainkan multiplikatif dalam dampaknya, dengan partisi yang membuang 90% data yang tidak relevan dan memastikan data yang relevan ada di memori kemudian *query engine* melakukan *indexing* yang dapat bekerja dengan sangat cepat pada set data yang kecil di dalam *cache* tersebut.

5. KESIMPULAN

Penelitian ini secara sistematis telah menganalisis, mengevaluasi dan membuktikan dampak efektivitas dari teknik partisi dan *indexing* (serta kombinasinya) sebagai strategi optimasi performa *query* pada lingkungan data *warehouse* berbasis PostgreSQL dengan skema *star schema* menggunakan data Online Retail II yang telah dibersihkan. Pengukuran dilakukan dengan *"EXPLAIN(ANALYZE, BUFFERS)"*. Berdasarkan hasil pengujian dan analisis yang telah dilakukan, dapat ditarik beberapa kesimpulan kunci.

Adapun kesimpulan hasil utama yang diperoleh dari penelitian ini adalah sebagai berikut:

1. Implementasi kombinasi dari kedua teknik yaitu partisi dan *indexing b*erhasil mengurangi waktu eksekusi *query* secara drastis yaitu sebesar 55.13% dibandingkan tanpa optimasi.
2. Teknik partisi terbukti sangat efektif dalam penyaringan data, hal ini secara fundamental dapat mengubah cara basis data mengakses data yaitu dengan memastikan data yang relevan diproses di dalam *cache* memori yang cepat.
3. *Indexing m*enunjukkan kemampuan dalam mempercepat pencarian data secara signifikan namun efektivitasnya terbatas karena masih terhambat oleh latensi *I/O disk*.
4. Kelebihan:
   - Strategi optimasi terutama kombinasi partisi dan *indexing* memberikan peningkatan performa yang bukan sekadar penjumlahan manfaat masing-masing teknik, melainkan sebuah efek multiplikatif.
   - Seiring bertambahnya volume data, keuntungan dari *partition pruning* akan menjadi semakin besar, memastikan performa sistem tetap terjaga.
   - Dengan menghilangkan *I/O disk*, strategi ini mengurangi beban pada subsistem penyimpanan dan CPU, yang berarti efisiensi penggunaan sumber daya server secara keseluruhan.

5. Kekurangan:
   - Implementasi partisi menambah lapisan kompleksitas dalam administrasi basis data.
   - Penelitian ini berfokus pada *query* dengan filter rentang waktu.
   - Hasil penelitian ini didasarkan pada volume data dan konfigurasi perangkat keras tertentu.
6. Rekomendasi Pengembang:
   - Uji beragam pola dan berbagai distribusi data untuk menilai konsistensi manfaat.
   - Bandingkan tahunan dan bulanan untuk melihat kompromi antara *pruning* yang lebih tajam dengan planning overhead dan beban administrasi.
   - Replikasi penelitian ini pada dataset yang jauh lebih besar atau server berbeda (Linux, RAM lebih besar) untuk menilai skalabilitas dan memvalidasi apakah efek sinergis antara partisi dan *indexing* ini tetap konsisten.

UCAPAN TERIMA KASIH

Penulis mengucapkan terima kasih kepada dosen pembimbing bapak Chaerur Rozikin, pihak kampus, dan teman-teman tim penulis dalam penelitian ini yaitu Luthfi Sifa Khaerunisa, M. Ramadhan Syahrul Al Qodr, Juliana Widianti Dwi Putri, Joyce Rosita Firdaus serta semua pihak yang telah memberikan dukungan, bimbingan, dan motivasi hingga jurnal berjudul "Optimasi Proses Data *Warehouse* Menggunakan Partisi dan *Indexing p*ada PostgreSQL untuk Meningkatkan Performa *Query*" ini dapat terselesaikan dengan baik. Semoga karya ini bermanfaat bagi pengembangan ilmu pengetahuan.

Tanpa dukungan dari semua pihak tersebut penelitian ini tidak akan terealisasikan dengan baik. Rasa terima kasih yang tulus penulis sampaikan kepada semua elemen yang telah memberikan dorongan, motivasi, dan semangat hingga naskah ini dapat diselesaikan. Semoga segala bantuan dan dukungan yang diberikan memperoleh balasan kebaikan, dan hasil penelitian ini dapat memberikan manfaat bagi perkembangan keilmuan di bidang optimasi data *warehouse*.

DAFTAR PUSTAKA

[1] J. Inukonda, "Leveraging Dimensional Modeling for Optimized Healthcare Data Warehouse Cloud Migration: Data Masking and Tokenization," *International Journal of Science and Research (IJSR)*, vol. 13, no. 10, pp. 437–441, Oct. 2024, doi: 10.21275/sr241004233606.

[2] S. V. Salunke and A. Ouda, "A Performance Benchmark for the PostgreSQL and MySQL Databases," Oct. 01, 2024, *Multidisciplinary Digital Publishing Institute (MDPI)*. doi: 10.3390/fi16100382.

[3] O. M. Hashem, K. Rahouma, N. S. Abd, and E. Hameed, "Enhancing Quality Factors in Data Warehousing Through Study and Improvement," *Journal of Advanced Engineering Trends*, vol. 44, no. 1, pp. 187–193, Jan. 2025.

[4] W. Wijaya, J. Wiratama, and S. F. Wijaya, "Implementation of Data Warehouse and Star Schema for Optimizing Property Business Decision Making," *G-Tech: Jurnal Teknologi Terapan*, vol. 8, no. 2, pp. 1242–1250, Apr. 2024, doi: 10.33379/gtech.v8i2.4091.

[5] M. N. Gundapaneni, "Data Partitioning: Optimizing Performance in Large Database Systems," *European Modern Studies Journal*, vol. 9, no. 4, pp. 956–964, Aug. 2025, doi: 10.59573/emsj.9(4).2025.90.

[6] V. C, S. Unnikrishnan, and J. V N, "Basic Indexing Techniques in Relational Database," *INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH IN TECHNOLOGY*, vol. 6, no. 12, May 2020, [Online]. Available: https://www.guru99.com/indexing-in-database.

[7] B. Azizi, A. A. Pratama, G. M. Dysa, and C. Carudin, "ANALISIS PENERAPAN DESIGN PATTERN SINGLETON DALAM PENGELOLAAN KONEKSI DATABASE UNTUK EFISIENSI MEMORI," *Jurnal Informatika dan Teknik Elektro Terapan*, vol. 13, no. 3, pp. 2127–2136, Jul. 2025, doi: 10.23960/jitet.v13i3.6705.

[8] J. Mostafa, S. Wehbi, S. Chilingaryan, and A. Kopmann, "SciTS: A Benchmark for Time-Series Databases in Scientific

Experiments and Industrial Internet of Things," Jun. 2022, doi: 10.1145/3538712.3538723.

[9] T. Vu and A. Eldawy, "R*-Grove: Balanced Spatial Partitioning for Large-scale Datasets," Jul. 2020, [Online]. Available: http://arxiv.org/abs/2007.11651

[10] Vasudevan Senathi Ramdoss, "Optimizing database queries: Cost and performance analysis," *International Journal of Science and Research Archive*, vol. 2, no. 2, pp. 293–297, Aug. 2021, doi: 10.30574/ijsra.2021.2.2.0025.

[11] H. Nicholson, P. Chrysogelos, and A. Ailamaki, "HPCache: memory-efficient OLAP through proportional caching revisited," in *VLDB Journal*, Springer Science and Business Media Deutschland GmbH, Nov. 2024, pp. 1775–1791. doi: 10.1007/s00778-023-00828-7.

[12] M. Riyaz Ansari, K. Rahim, R. Bhoje, and S. Bhosale, "A STUDY ON RESEARCH DESIGN AND ITS TYPES," *International Research Journal of Engineering and Technology (IRJET)*, vol. 9, no. 7, pp. 1132–1135, Jul. 2022, [Online]. Available: www.irjet.net

[13] B. All Habsy and M. Nursalim, "Jenis-Jenis Metode Pengumpulan Data (Qualitative Research)," *Jurnal Pendidikan Tambusai*, vol. 9, pp. 9932–9938, Feb. 2025.

[14] L. Barać, "Research Environment," Split, 2023, pp. 1–17. doi: 10.1007/978-3-031-22412-6_1.

[15] J. Wehrstein, T. Eckmann, R. Heinrich, and C. Binnig, "JOB-Complex: A Challenging Benchmark for Traditional & Learned Query Optimization," Jul. 2025, [Online]. Available: http://arxiv.org/abs/2507.07471