ISSN: 2614-6754 (print)
ISSN: 2614-3097(online)

Halaman 2094-2102
Volume 6 Nomor 1 Tahun 2022

Optimasi Database dengan Metode *Index* dan Partisi Tabel Database Postgresql pada Aplikasi E-Commerce. Studi pada Aplikasi Tokopintar

Samidi[1], Fadly[2], Yusuf Virmansyah[3], Ronal Yulyanto Suladi[4], Ario Bambang Lesmana[5]

[1,2,3,4,5] Fakultas Teknologi Informasi, Universitas Budi Luhur
e-mail: samidi@budiluhur.ac.id[1], 2111600645@student.budiluhur.ac.id[2], 2111600314@student.budiluhur.ac.id[3], 2111600835@student.budiluhur.ac.id[4], 2111600868@student.budiluhur.ac.id[5]

Abstrak

Efektifitas dan juga efisiensi waktu dalam mengakses dan pengambilan data sangatlah penting bagi sebuah perusahaan retail berbasis online yang memiliki jumlah data transaksi kegiatan sangat besar setiap harinya, sepertihalnya yang terjadi pada Aplikasi E-Commerce Toko Pintar dalam sebuah aktivitas transaksi mutasi. Dalam penelitian ini, dilakukan sebuah optimasi database untuk mengoptimalkan kinerja database guna menunjang performa dari database itu sendiri. Optimasi yang dilakukan dalam penelitian ini bertujuan untuk mengatasi dan memperoleh informasi sekaligus solusi masalah waktu eksekusi dan waktu respon query jika data pada database yang akan diproses sangat besar yang berdampak pada lamanya proses query itu sendiri. Adapun cara yang digunakan untuk melakukan optimalisasi database dalam penelitian ini adalah dengan metode *Index* dan Prtition Table. *Index* adalah sebuah objek dalam sistem database yang dapat mempercepat proses pencarian (query) data. Dari hasil eksperimen dengan dataset sebesar 2.248.590 data didapatkan sebuah hasil yang sangat memuaskan dimana terdapat perbedaan waktu query yang signifikan sebelum dan sesudah menggunakan *index* dan partition table, dimana Database sebelum diberi *Index* dan Partition Table akan memakan waktu yang lama dibanding setelah menggunakan *Index* dan Partition Table. Kelemahan *Index* dan Partition Table terjadi hanya pada kondisi tertentu yaitu padasaat melakukan insert data kedalam table.

Kata Kunci: *Table Partisi, Index, Optimasi Basis Data*

Abstract

Effectiveness and also time efficiency in accessing and retrieving data is very important for an online-based retail company that has a very large amount of transaction data every day, as is the case with the Smart Store E-Commerce Application in a mutation transaction activity. In this study, a database optimization was carried out to optimize the performance of the database to support the performance of the database itself. The optimization carried out in this study aims to overcome and obtain information as well as a solution to the problem of execution time and query response time if the data in the database to be processed is very large which has an impact on the length of the query process itself. The method used to optimize the database in this research is the *Index* and Partition Table methods. *Index* is an object in the database system that can speed up the process of searching (querying) data. From the experimental results with a dataset of 2,248,590 data, a very satisfying result was obtained where there was a significant difference in query time before and after using the *index* and partition table, where the database before being given the *Index* and Partition Table would take a longer time than after using the *Index* and Partition Table. Partition Table. The weakness of the *Index* and Partition Table occurs only under certain conditions, namely when inserting data into the table.

Keywords : *Table Partition, Index, and Database Optimization*

PENDAHULUAN

Pesatnya pertumbuhan ritel modern di Indonesia dimana saat ini banyak kita temukan berada di tengah tengah pemukiman masyarakat menimbulkan persaingan dengan ritel tradisional. Hal ini tentunya perlu disikapi mengenai pertumbuhan ritel tradisional untuk dapat terus berdaya saing. Ritel tradisional dituntut untuk menyediakan produk yang lengkap dan jam buka yang lebih konsisten. Fenomena ini yang kemudian mendorong menghadirkan Aplikasi Tokopintar sebagai salah satu tawaran solusi pengembangan ekosistem ritel tradisional dengan memanfaatkan teknologi digital berbasis Andoid dengan layanan transaksi produk digital (seperti pulsa, token, paket data) dan belanja online kebutuhan dagang.

Seiring perkembangan jumlah transaksi dan jumlah warung yang mengakses aplikasi dalam waktu yang bersamaan mulai terasa penurunan performa pada aplikasi, khususnya terasa pada informasi mutasi yang setiap kali di akses oleh pengguna aplikasi setiap menyelesaikan transaksi.

Query yang dianggap hanya sebagai query sederhana memanggil informasi transaksi 1(satu) pelanggan melewatkan developer dalam memenuhi kaidah Relational Database Management System (RDBMS).

Untuk itu perlu adanya sebuah solusi untuk mengoptimalkan kinerja database, dan salah satu solusi yang diusulkan oleh penyedia database adalah partisi tabel[1] dan juga implementasi *Index*. *Index* adalah struktur data fisik yang terpisah yang memungkinkan query untuk mengakses satu atau lebih baris data dengan cepat [2][3], sedangkan Partisi table adalah pembagian tabel menjadi bagian-bagian yang berubah-ubah yang membentuk beberapa rentang data yang terpisah, misalnya: bulanan, triwulanan, atau tahunan.dan 1 kalimat apa itu tabel partisi[4].

Index dapat dianalogikan seperti daftar isi dalam sebuah buku, yang dapat memudahkan pembaca dalam mencari topic yang diinginkan, dengan *Index* kita tidak perlu melakukan pencarian dari table baris pertama hingga baris terakhir yang akan membutuhkan waktu[5].

Penelitian terdahulu yang dilakukan oleh Kusrini dan Januarito [6] tentang "Implementasi Penerapan *Index* Dalam Mempercepat Eksekusi Query Pada Basis Data . Dalam penelitian tersebut diperoleh hasil eksperimen dengan dataset sebesar 299.473 data didapatkan sebuah hasil yang sangat memuaskan dimana terdapat perbedaan waktu query sebelum dan sesudah menggunakan *index* dimana Database sebelum diberi *Index* akan memakan waktu yang lebih lama dibanding setelah menggunakan *Index*

Adapun Suhartati dan Yeyen Dwi Atma [7] membahas tentang "Optimasi Query Sederhana Guna Kecepatan Query pada Database Server. Dalam penelitian tersebut membahas bahwa untuk memperoleh suatu kinerja aplikasi dibutuhkan teknik optimasi query dengan mempraktikkan aturan penulisan query, sehingga diperoleh trik query sederhana yang bisa diterapkan untuk mendapatkan keuntungan kinerja secara optimal

Demikian pula Moh Silhan dan Isa Anshori [8] tentang Perbandingan Subset Query pada Multiple Relasi menggunakan Tabel Terpartisi dan Tabel Tidak terpartisi dengan Metode Cost-Based. Hasil perbedaan query yang di akses pada tabel non-partisi dan tabel yang telah di partisi waktu yang paling cepat adalah dari tabel yang dipartisi terbukti bahwa tabel yang di partisi waktu yang di perlukan 0.0736 sedangkan waktu yang diperlukan untuk akses data pada tabel non-partisi 0.1725. Untuk akses data yang di akses menggunakan card LAN lebih cepat dibandingkan akses dengan menggunakan jaringan wireless karena akses data dengan menggunakan wireless dipengaruhi jarak dari akses data.

Pada hasil penelitian sebelumnya lebih banyak mencoba mencari dan menganalisa waktu respon dengan melalukan uji perbandingan pada dua buah model data table dengan uji eksekusi query, sehingga penggalian informasi yang didapat dirasa kurang cukup. Dalam penelitian ini peneliti mencoba melakukan ekperimen lebih jauh dibandingkan penelitian sebelumnya, dimana peniliti mencoba membandingkan waktu respontime query dengan

menggunakan beberapa model data table sehingga diharapkan memperoleh informasi lebih baik terkait metode optimasi database mana yang lebih baik. Sekaligus membuktikan teori-teori yang memperkuat metode penelitian ini.

METODE PENELITIAN

Dalam penelitian ini digunakan metode eksperimental dimana Sample Tabel yang digunakan adalah Tabel Mutasi yang merupakan salah satu table utama yang sering di akses oleh pengguna aplikasi untuk melihat saldo yang terpotong akibat transaksi. Data yang dijadikan contoh adalah data Mutasi tanggal 10 Desember 2021 yang kemudian di duplicate untuk pemenuhan kebutuhan data sample. Proses penduplikasian dilakukan dengan menggunakan tools “notepad++” untuk mereplace beberapa isi field agar data tetap unique.

Adapun dalam melakukan pengujian eksekusi Query dengan membandingkan reponse time pada table yang :

1. tidak menggunakan *attribute key* ;
2. menggunakan *attribute Primary Key dan Foreign Key* tanpa *index*;
3. menggunakan *Primary Key,Foreign Key* dan juga *Index*;
4. menggunakan *Primary Key,Foreign Key, Index* dan penerapan Partition Table.

Lingkungan yang digunakan berupa server local yang menggunakan perangkat dengan spesifikasi sebagai berikut :

1. Sistem Operasi Windows 10 Pro 64 Bit ;
2. Processor Intel Core i5-8250U;
3. Ram 16GB; Hardisk 1TB;
4. Database PostgreSQL 14; dan
5. Tools pgAdmin 4.

Adapun langkah-langkah penelitian ini dimulai dengan memisahkan karakteristik model pada schema database yang terpisah : schema “test0” untuk tabel yang tidak menggunakan *attribute key*; schema “test1” untuk table yang menggunakan attribute *Primary Key dan Foreign Key* tetapi tanpa *index*; schema “test2” untuk table yang menggunakan *Primary Key, Foreign Key* dan juga *Index*; serta schema “test3” untuk table yang menggunakan *Primary Key,Foreign Key*, *Index* dan penerapan Partition Table. Kemudian dilanjutkan dengan pembuatan Query Insert kedalam table, setiap proses insert yang telah dilakukan selalu dicatat *respond* time terhadap: 1) Proses Insert; 2) Query *Single Table* dengan *Where Clause* ; 3) Query JOIN Table dengan *Where Clause*; 4) Query Update. Pada akhir pembahasan akan dilakukan perbandingan waktu respon time pada table yang tidak menggunakan *attribute key* dengan table yang menggunakan *attribute Primary Key* dan *Foreign Key* tanpa *index*, dan juga membandingkannya pada table yang menggunakan Primary Key,Foreign Key dan juga *Index*. Serta untuk memperkuat hasil penelitian kami pun melakukan perbandingan pada table yang menggunakan *Primary Key, Foreign Key*, *Index* dan penerapan Partition Table. Sehingga akan dapat menggambarkan sekaligus memastikan metode mana yang dapat dikatakan lebih baik dalam kasus optimasi database ini.

HASIL PEMBAHASAN

Object dalam penelitian ini dibatasi dalam 2 tabel Customer dan Mutasi, dimana table ini merupakan bagian utama yang sering di akses oleh pengguna aplikasi untuk melihat saldo yang terpotong akibat transaksi. Pada table mutasi dengan mengambil contoh transaksi 10 Desember 2021 terdapat 66.135 record. Pada table Customer memuat 300.000 data pelanggan. Pembuatan field mengambil nama field dan attribute yang sama dengan yang berjalan di aplikasi.

Schema Test0 : tabel yang tidak menggunakan attribute key

```
DBTEST/postgres@PostgreSQL 14
Query Editor  Query History
CREATE TABLE IF NOT EXISTS test0.customer
(
    cust_no character varying(20) , cust_name character varying(150) ,cust_email character varying(150) ,
    cust_no_hp character varying(15) COLLATE pg_catalog."default" NOT NULL,cust_address text ,cust_birth_place character varying(100) ,
    cust_birth_date date, cust_no_ktp character varying(17) ,cust_password character varying(100) ,cust_pin character varying(6) ,
    cust_imei character varying(100) ,cust_foto_ktp character varying(150) ,cust_foto_selfie character varying(150) ,
    cust_foto_ttd character varying(150) ,cust_foto_npwp character varying(150) ,cust_foto_toko character varying(150) ,
    cust_foto_profile character varying(150) ,cust_tipe character varying(20) ,otp character varying(100) ,
    cust_nama_toko character varying(150) ,cust_address_toko text ,created_at date DEFAULT ('now'::text)::date,
    is_deleted character varying(1) COLLATE pg_catalog."default" DEFAULT 0,
    is_active character varying(1) COLLATE pg_catalog."default" DEFAULT 1,
    kode_referral character varying(6) ,reg_id character varying(100) ,level character varying(5) ,cust_imei2 character varying(100) ,
    cust_imei3 character varying(100) ,cust_imei4 character varying(100) ,cust_imei5 character varying(100) ,token text ,
    tanggal_updated character varying(22) ,key_unik text ,otp_password character varying(100) ,is_masbro character varying(1) ,
    cust_jk character varying(50) ,cust_nama_jalan character varying(50) ,cust_postal_code character varying(50) ,
    cust_id_kel character varying(50) ,cust_id_kec character varying(50) ,cust_id_kab character varying(50) ,
    cust_id_prov character varying(50) ,cust_no_npwp character varying(25) ,cust_sim_id character varying(100) ,tanggal_login date,
    waktu_login character varying(10) ,total_percobaan_login double precision DEFAULT 0,type_daftar character varying(10) ,
    otp_ktp character varying(20) ,reg_id_daftar character varying(100)
)
```

```
DBTEST/postgres@PostgreSQL 14
Query Editor  Query History
CREATE TABLE IF NOT EXISTS test0.mutasi
(
    no_trans character varying(15),cust_no character varying(20) ,cust_no_hp character varying(20) ,tipe character varying(1) ,
    saldo_awal double precision,nominal double precision,saldo_akhir double precision,tipe_trans character varying(15),
    data_01 character varying(20) ,keterangan text ,imei character varying(20),tanggal_trans date,waktu_trans character varying(8) ,
    created_at character varying(20) ,deleted_at character varying(20) ,created_by character varying(20)
)
```

Schema Test1 : table yang menggunakan attribute *Primary Key dan Foreign Key* tetapi tanpa *index*

Proses pembuatan table sama dengan pembuatan table pada shcema ‘test0’ hanya perlu ditambahkan Primary Key dan Foreign Key seperti berikut :

```
DBTEST/postgres@PostgreSQL 14
Query Editor  Query History
ALTER TABLE test1.customer
ADD CONSTRAINT customer_pkey PRIMARY KEY (cust_no);
```

Pada Schema test1 ,selain Primary Key difungsikan juga Foreign Key dari table Customer ke table Mutasi.

```
DBTEST/postgres@PostgreSQL 14
Query Editor  Query History
ALTER TABLE test1.mutasi
ADD CONSTRAINT mutasi_pkey PRIMARY KEY (no_trans),
ADD CONSTRAINT fk_customer FOREIGN KEY(cust_no) REFERENCES test4.customer(cust_no);
```

Schema Test2 : table yang menggunakan attribute *Primary Key dan Foreign Key* dan menggunakan *index*.

Proses pembuatan table sama seperti pembuatan table pada schema ‘test0’ kemudian di tambahkan Primary Key dan Foreign Key seperti shcema ‘test1’ lalu di tambahkan pembuatan *index* seperti berikut :

```
DBTEST/postgres@PostgreSQL 14
Query Editor  Query History
create index idx_mutasi_tgltrans on test2.mutasi (tanggal_trans);
```

Schema Test3 : table yang menggunakan attribute *Primary Key , Foreign Key ,Index* dan penerapan Partition Table.

Proses pembuatan table sejak awal sudah harus didefinisikan Primary Key, Foreign Key dan dekralasi bahwa Table memiliki Partition.

```
DBTEST/postgres@PostgreSQL 14
Query Editor  Query History
CREATE TABLE IF NOT EXISTS test3.mutasi
(
    no_trans character varying(15),cust_no character varying(20) ,cust_no_hp character varying(20) ,tipe character varying(1) ,
    saldo_awal double precision,nominal double precision,saldo_akhir double precision,tipe_trans character varying(15),
    data_01 character varying(20) ,keterangan text ,imei character varying(20),tanggal_trans date,waktu_trans character varying(8) ,
    created_at character varying(20) ,deleted_at character varying(20) ,created_by character varying(20),
    CONSTRAINT mutasi_pkey PRIMARY KEY (no_trans,tanggal_trans),
    CONSTRAINT fk_customer FOREIGN KEY(cust_no) REFERENCES test3.customer(cust_no)
)
PARTITION BY RANGE (tanggal_trans);
```

Kemudian Proses create Partition di lakukan sejumlah dengan data yang akan dibuat.

```
DBTEST/postgres@PostgreSQL 14
Query Editor  Query History
CREATE TABLE test3.ptbl_mutasi_20211209 PARTITION OF test2.mutasi for values from ('2021-12-09') to ('2021-12-10');
CREATE TABLE test3.ptbl_mutasi_20211210 PARTITION OF test2.mutasi for values from ('2021-12-10') to ('2021-12-11');
CREATE TABLE test3.ptbl_mutasi_20211211 PARTITION OF test2.mutasi for values from ('2021-12-11') to ('2021-12-12');
CREATE TABLE test3.ptbl_mutasi_20211212 PARTITION OF test2.mutasi for values from ('2021-12-12') to ('2021-12-13');
CREATE TABLE test3.ptbl_mutasi_20211213 PARTITION OF test2.mutasi for values from ('2021-12-13') to ('2021-12-14');
CREATE TABLE test3.ptbl_mutasi_20211214 PARTITION OF test2.mutasi for values from ('2021-12-14') to ('2021-12-15');
```

Kemudian dilengkapi dengan Create *Index*:

```
DBTEST/postgres@PostgreSQL 14
Query Editor  Query History
create index idx_mutasi_tgltrans on test3.mutasi (tanggal_trans);
```

Dari ke-empat Schema diatas kemudian diujikan dalam 4 (empat) scenario :

Perintah insert data

Insert dilakukan dengan 2 Model :

Model 1 dengan Insert Into per Row data :

```
INSERT INTO test1.mutasi values('T176183305','C100190711',NULL,'K',419458,10975,408483,'002_PAYMENT','DP57901102',NULL,NULL,'2021-12-10','08:51:18','2021-12-10 08:51:18',NULL,NULL);
INSERT INTO test1.mutasi values('T176183319','C100193626',NULL,'K',754606,5400,749206,'002_PAYMENT','DP57901109',NULL,NULL,'2021-12-10','08:51:30','2021-12-10 08:51:30',NULL,NULL);
INSERT INTO test1.mutasi values('T176183329','C100196320',NULL,'K',130027,106500,23527,'012_ATMBERSAMA','DP57901114',NULL,NULL,'2021-12-10','08:51:34','2021-12-10 08:51:34',NULL,NULL)
INSERT INTO test1.mutasi values('T176183335','C100197594',NULL,'K',84625,5400,79225,'002_PAYMENT','DP57901118',NULL,NULL,'2021-12-10','08:51:35','2021-12-10 08:51:35',NULL,NULL);
INSERT INTO test1.mutasi values('T176183342','C100195935',NULL,'K',220456,20500,199956,'002_PAYMENT','DP57901124',NULL,NULL,'2021-12-10','08:51:43','2021-12-10 08:51:43',NULL,NULL);
INSERT INTO test1.mutasi values('T176183308','C100195062',NULL,'K',3409978,10975,3399003,'002_PAYMENT','DP57901103',NULL,NULL,'2021-12-10','08:51:19','2021-12-10 08:51:19',NULL,NULL);
INSERT INTO test1.mutasi values('T176183318','C100195190',NULL,'K',165016,20700,144316,'002_PAYMENT','DP57901107',NULL,NULL,'2021-12-10','08:51:28','2021-12-10 08:51:28',NULL,NULL);
INSERT INTO test1.mutasi values('T176183320','C100197236',NULL,'K',186086,5775,180311,'002_PAYMENT','DP57901108',NULL,NULL,'2021-12-10','08:51:30','2021-12-10 08:51:30',NULL,NULL);
INSERT INTO test1.mutasi values('T176183323','C100194705',NULL,'K',80426,50500,29926,'002_PAYMENT','DP57901112',NULL,NULL,'2021-12-10','08:51:30','2021-12-10 08:51:30',NULL,NULL);
INSERT INTO test1.mutasi values('T176183310','C100195332',NULL,'K',86306,5775,80531,'002_PAYMENT','DP57901104',NULL,NULL,'2021-12-10','08:51:21','2021-12-10 08:51:21',NULL,NULL);
INSERT INTO test1.mutasi values('T176183328','C100194991',NULL,'K',246843,10900,235943,'002_PAYMENT','DP57901113',NULL,NULL,'2021-12-10','08:51:33','2021-12-10 08:51:33',NULL,NULL);
INSERT INTO test1.mutasi values('T176183312','C100190103',NULL,'K',383879,10550,373329,'002_PAYMENT','DP57901105',NULL,NULL,'2021-12-10','08:51:23','2021-12-10 08:51:23',NULL,NULL);
INSERT INTO test1.mutasi values('T176183330','C100191385',NULL,'K',77002,10550,66452,'002_PAYMENT','DP57901115',NULL,NULL,'2021-12-10','08:51:34','2021-12-10 08:51:34',NULL,NULL);
```

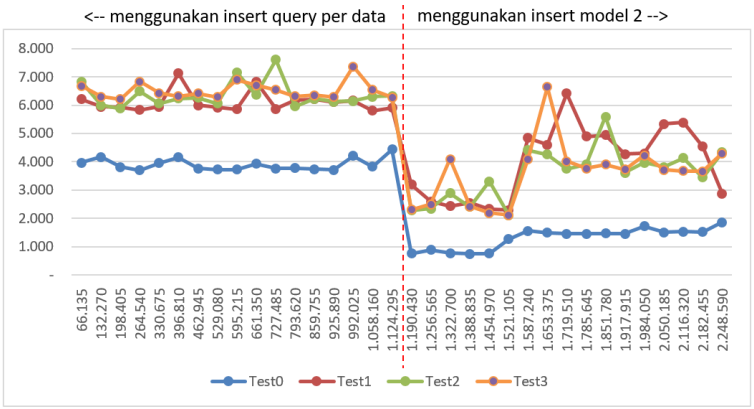
Model 2 dengan Insert data From data table yang sudah ada :

```
INSERT INTO test2.mutasi
select a.no_trans, a.cust_no, a.cust_no_hp, a.tipe, a.saldo_awal, a.nominal, a.saldo_akhir, a.tipe_trans, a.data_01,
      a.keterangan, a.imei, a.tanggal_trans, a.waktu_trans, a.created_at, a.deleted_at, a.created_by
from( select 'P'||substr(no_trans,2,9) as no_trans, cust_no, cust_no_hp, tipe, saldo_awal, nominal, saldo_akhir, tipe_trans, data_01, keterangan, imei,
      to_date('20210410','YYYYMMDD') as tanggal_trans, waktu_trans, created_at, deleted_at, created_by
     from test2.mutasi where tanggal_trans='2021-12-10') a;
```

ISSN: 2614-6754 (print)
ISSN: 2614-3097(online)

Halaman 2094-2102
Volume 6 Nomor 1 Tahun 2022

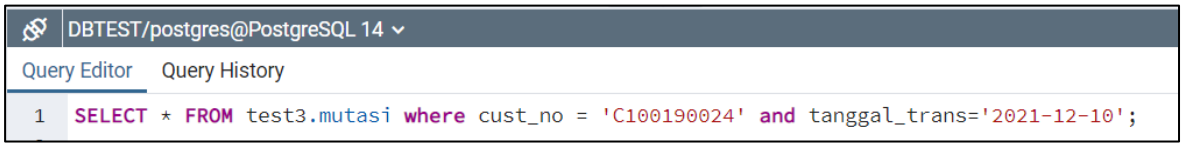
No	Jumlah Record	Catatan Waktu dalam Msec				Model Insert
		test0	test1	test2	test3	
1	66.135	3.964	6.204	6.834	6.677	Model Insert 1
2	132.270	4.172	5.947	5.990	6.296	
3	198.405	3.811	5.921	5.885	6.203	
4	264.540	3.697	5.834	6.491	6.825	
5	330.675	3.953	5.948	6.061	6.426	
6	396.810	4.151	7.119	6.233	6.310	
7	462.945	3.763	5.995	6.263	6.418	
8	529.080	3.727	5.917	6.053	6.290	
9	595.215	3.729	5.859	7.155	6.895	
10	661.350	3.931	6.826	6.354	6.702	
11	727.485	3.767	5.868	7.607	6.540	
12	793.620	3.769	6.175	5.954	6.305	
13	859.755	3.738	6.214	6.217	6.347	
14	925.890	3.709	6.111	6.122	6.285	
15	992.025	4.211	6.152	6.147	7.361	
16	1.058.160	3.829	5.798	6.302	6.534	
17	1.124.295	4.440	5.910	6.319	6.272	
18	1.190.430	755	3.186	2.282	2.306	Model Insert 2
19	1.256.565	883	2.600	2.349	2.493	
20	1.322.700	773	2.429	2.891	4.095	
21	1.388.835	752	2.551	2.405	2.415	
22	1.454.970	765	2.327	3.312	2.189	
23	1.521.105	1.272	2.291	2.110	2.118	
24	1.587.240	1.560	4.837	4.409	4.091	
25	1.653.375	1.492	4.598	4.259	6.640	
26	1.719.510	1.458	6.418	3.747	4.021	
27	1.785.645	1.452	4.891	3.897	3.760	
28	1.851.780	1.473	4.944	5.585	3.906	
29	1.917.915	1.454	4.274	3.615	3.725	
30	1.984.050	1.728	4.294	3.965	4.224	
31	2.050.185	1.513	5.338	3.819	3.709	
32	2.116.320	1.530	5.390	4.129	3.679	
33	2.182.455	1.514	4.541	3.449	3.665	
34	2.248.590	1.860	2.871	4.331	4.291	



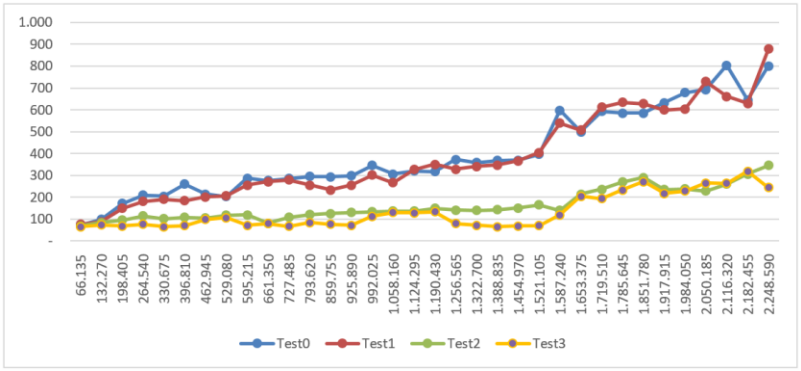
Gambar 1. Table dan Grafik Perbandingan waktu eksekusi query pada perintah insert data.

Proses insert pada tabel tanpa Key & *Index* akan lebih cepat, tetapi memiliki resiko data redundan karena tidak ada filter terhadap data yang duplicate. Pada tabel yang memiliki Key, tidak ditemukan kecendrungan yang significant antara yang menggunakan *index*, partition table ataupun tanpa menggunakan keduanya, selama menggunakan Primary Key dan Foreign Key.

Perintah Query Single Table dengan Where Clause



No	Jumlah Record	Catatan Waktu dalam Msec			
		test0	test1	test2	test3
1	66.135	73	76	69	66
2	132.270	98	88	84	74
3	198.405	171	150	95	69
4	264.540	209	181	114	77
5	330.675	205	190	103	66
6	396.810	261	184	108	70
7	462.945	214	202	105	99
8	529.080	204	207	118	107
9	595.215	288	256	119	72
10	661.350	277	271	82	80
11	727.485	286	280	109	67
12	793.620	295	256	122	85
13	859.755	294	234	126	77
14	925.890	298	256	130	72
15	992.025	346	301	133	113
16	1.058.160	306	266	136	130
17	1.124.295	321	327	136	129
18	1.190.430	317	351	150	134
19	1.256.565	373	328	142	80
20	1.322.700	358	341	141	72
21	1.388.835	368	348	144	65
22	1.454.970	370	366	151	69
23	1.521.105	396	404	166	71
24	1.587.240	598	539	141	120
25	1.653.375	499	508	215	205
26	1.719.510	593	612	237	194
27	1.785.645	585	635	270	234
28	1.851.780	585	628	291	271
29	1.917.915	633	600	236	218
30	1.984.050	679	604	238	229
31	2.050.185	693	730	229	265
32	2.116.320	803	661	260	264
33	2.182.455	645	629	306	319
34	2.248.590	801	878	346	244



Gambar 2. Table dan Grafik Perbandingan waktu eksekusi query pada perintah Query Single Table dengan Where Clause.

Pada Query sederhana yang hanya melibatkan 1 tabel dimana yang menjadi filternya adalah field kunci, ditemukan penggunakan *Index* dan Partition menghasilkan catatan waktu yang lebih bagi dibanding yang tidak menggunakan. Dan sekali lagi, proses Primary Key dan

Foreign Key digunakan sebagai tahapan menjaga Data Quality (memastikan data tidak duplicate) tetapi tidak berpengaruh pada catatan waktu Query.

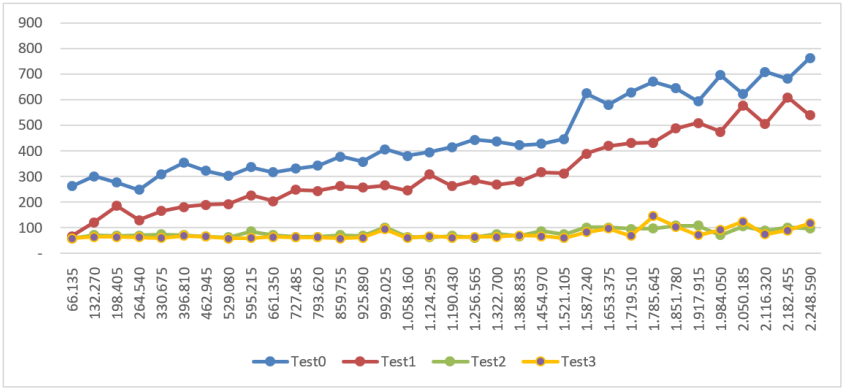
Perintah JOIN Table dengan Where Clause

DBTEST/postgres@PostgreSQL 14

Query Editor | Query History

```
1  SELECT a.no_trans ,c.cust_name ,a.tipe_trans, a.nominal
2   FROM test3.mutasi as a
3   left outer join test3.customer as c on a.cust_no = c.cust_no
4   where a.cust_no = 'C100190024' and a.tanggal_trans='2021-12-10'
```

No	Jumlah Record	Catatan Waktu dalam Msec			
		test0	test1	test2	test3
1	66.135	262	67	59	59
2	132.270	301	122	71	65
3	198.405	276	185	68	64
4	264.540	248	130	70	63
5	330.675	310	165	74	61
6	396.810	354	181	71	68
7	462.945	323	190	66	66
8	529.080	302	193	63	58
9	595.215	336	227	85	60
10	661.350	316	204	71	64
11	727.485	331	248	65	63
12	793.620	342	244	65	63
13	859.755	378	263	71	59
14	925.890	358	257	68	62
15	992.025	407	266	101	95
16	1.058.160	381	246	64	60
17	1.124.295	395	309	63	67
18	1.190.430	415	262	69	62
19	1.256.565	444	286	62	64
20	1.322.700	436	268	76	64
21	1.388.835	423	280	67	70
22	1.454.970	428	316	87	67
23	1.521.105	447	312	75	61
24	1.587.240	625	389	101	83
25	1.653.375	580	420	102	97
26	1.719.510	629	430	96	69
27	1.785.645	670	432	98	145
28	1.851.780	644	487	109	104
29	1.917.915	594	509	109	72
30	1.984.050	696	475	71	91
31	2.050.185	622	578	106	124
32	2.116.320	709	505	88	75
33	2.182.455	682	609	102	90
34	2.248.590	763	540	97	117



Gambar 3. Table dan Grafik Perbandingan waktu eksekusi query pada perintah JOIN Table dengan Where Clause.

Pada Query yang melibatkan JOIN TABLE terlihat perbedaan yang cukup jelas antara yang menggunakan KEY dan tidak menggunakan KEY, kemudian di tambah dengan penggunaan *INDEX* membuat kecepatan Query sangat significant. Belum ditemukan hal yang significant perbedaan table yang menggunakan *INDEX* dengan atau tanpa PARTITION.

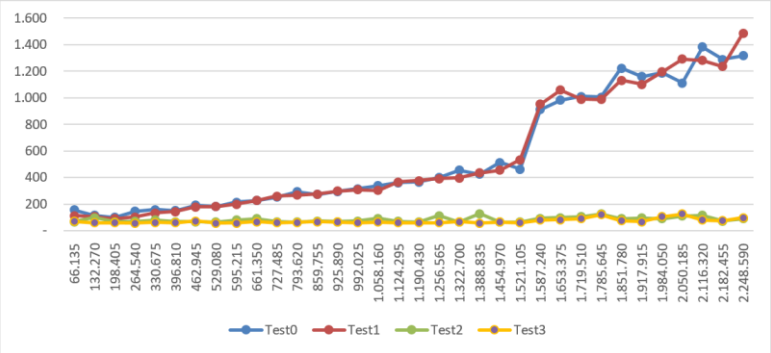
Query Update

DBTEST/postgres@PostgreSQL 14

Query Editor | Query History

```
1  update test3.mutasi set keterangan = 'test update' where cust_no='C100190024' and tanggal_trans='2021-12-10'
```

No	Jumlah Record	Catatan Waktu dalam Msec			
		test0	test1	test2	test3
1	66.135	155	112	67	69
2	132.270	114	105	97	59
3	198.405	100	90	66	60
4	264.540	144	103	70	56
5	330.675	158	132	80	62
6	396.810	149	142	66	59
7	462.945	192	178	66	71
8	529.080	182	181	65	56
9	595.215	214	198	78	57
10	661.350	225	225	91	67
11	727.485	251	259	66	59
12	793.620	291	268	63	61
13	859.755	271	274	72	66
14	925.890	295	297	67	65
15	992.025	316	309	71	60
16	1.058.160	337	304	92	65
17	1.124.295	361	365	69	59
18	1.190.430	366	376	64	59
19	1.256.565	399	391	113	58
20	1.322.700	453	396	65	66
21	1.388.835	425	433	128	56
22	1.454.970	512	455	64	64
23	1.521.105	462	533	65	58
24	1.587.240	911	952	93	81
25	1.653.375	982	1.057	97	82
26	1.719.510	1.009	990	104	90
27	1.785.645	1.005	986	125	120
28	1.851.780	1.222	1.132	88	75
29	1.917.915	1.158	1.100	96	67
30	1.984.050	1.187	1.195	88	104
31	2.050.185	1.110	1.291	111	125
32	2.116.320	1.382	1.282	116	80
33	2.182.455	1.289	1.235	72	73
34	2.248.590	1.317	1.485	91	97



Gambar 4. Table dan Grafik Perbandingan waktu eksekusi query pada perintah Query Update.

Index terbukti mampu untuk mempercepat proses update, dalam jumlah record yang diupdate dalam jumlah yang sama, *index* dapat stabil mengeksekusi proses update tanpa banyak terpengaruh dengan penambahan data. Berbeda dengan Tabel yang tidak memiliki *Index*, maka proses updatenya akan bertambah lama, seiring dengan pertumbuhan jumlah data.

Table 1. Rekapitulasi Hasil Penelitian

No	Kondisi Data	Operasi	Hasil Penelitian			
			1	2	3	4
1	2.248.590	Insert data	√	☐	☐	☐
2	2.248.590	Query Single Table dengan Where Clause	☐	☐	☐	√
3	2.248.590	JOIN Table dengan Where Clause	☐	☐	√	√
4	2.248.590	Query Update	☐	☐	√	√

Keterangan :
1. Tidak Menggunakan *Attribute Key* (Table : Test0);
2. Menggunakan *Attribute Primary Key Dan Foreign Key* Tanpa *Index* (Table : Test1);
3. Menggunakan *Primary Key,Foreign Key* Dan Juga *Index* (Table : Test2);
4. Menggunakan *Primary Key,Foreign Key*, *Index* Dan Penerapan Partition Table (Table : Test3).
5. Tanda ceklis √ diberikan untuk menunjukkan model data table mana yang memiliki hasil eksekusi query sangat baik.

SIMPULAN

Berdasarkan hasil penelitian yang dilakukan pada kasus Optimasi Database Dengan Metode *Index* dan Partisi Tabel Database Postgresql Pada Aplikasi E-Commerce. Studi Pada Aplikasi Tokopintar, dapat disimpulkan bahwa penggunaan *Primary Key, Foreign Key, Index* dan *Partition table* dianggap lebih baik dalam respon time eksekusi query dibandingkan 3 (tiga) table lainnya.

SARAN

Dalam penelitian ini masih didapatkan banyak hal yang dapat dikembangkan lagi, saran dari tim peneliti kepada peneliti berikutnya yang ingin melanjutkan penelitian ini adalah:

1. Dengan membedakan antara *index* dengan partition table tanpa menggunakan *index*
2. Membuat join yang lebih comlex untuk lebih dalam lagi mengetahui perbedaan *index* dan partition.

DAFTAR PUSTAKA

H. J. Watson, “Recent Developments in Data Warehousing,” Commun. Assoc. Inf. Syst., vol. 8, no. January, 2002, doi: 10.17705/1cais.00801.

P. Studi, S. Informasi, and U. Tarumanagara, “Perbandingan Optimasi Query,” pp. 113–117.

D. Petkovic, “Microsoft SQL Server 2008 , A Beginner ’ s Guide,” no. August, 2018.

P. Bednarczuk, “Optimization in Very Large Databases By Partitioning Tables,” Inform. Autom. Pomiary w Gospod. i Ochr. Środowiska, vol. 10, no. 3, pp. 95–98, 2020, doi: 10.35784/iapgos.2056.

R. Pamungkas, “Optimalisasi Query Dalam Basis Data My Sql Menggunakan Index,” Res. Comput. Inf. Syst. Technol. Manag., vol. 1, no. 1, p. 27, 2018, doi: 10.25273/research.v1i1.2453.

J. Freitas et al., “IMPLEMENTASI PENERAPAN INDEX DALAM MEMPERCEPAT EKSEKUSI QUERY PADA BASIS,” pp. 1–11.

S. Suhartati and Y. Dwi Atma, “Optimasi Query Sederhana Guna Kecepatan Query Pada Database Server,” Metik J., vol. 1, no. 1, pp. 13–17, 2017, [Online]. Available: http://jurnal.stmikbpn.ac.id/index.php/metik1/article/view/5.

I. Anshori, “Perbandingan Cross-Product Dan Subset Query Pada Multiple Relasi Menggunakan Partisi Tabel Dengan Metode Cost-Based,” J. Mhs. Fak. Sains dan Teknol., vol. 0, no. 0, pp. 19–23, 2013.