Jurnal Pendidikan dan Teknologi Indonesia (JPTI)
Vol. 4, No. 12, December 2024, Hal. 709-712
DOI: https://doi.org/10.52436/1.jpti.537
p-ISSN: 2775-4227
e-ISSN: 2775-4219

Komparasi *Database Performance Tuning* Melalui *Metode Object Relation Mapping* pada SQL Server

Dian Muhammad Gufron[1], Muhammad Ramadhon[*2], Samidi[3]

[1,2,3]Fakultas Teknologi Informasi, Universitas Budi Luhur, Indonesia
Email: [1]gufron.dian@gmail.com, [2]doon13@gmail.com, [3]samidi@budiluhur.ac.id

Abstrak

Pada proses *database performance tuning* terdapat beberapa permasalahan apabila diterapkan pada sistem berbasis *object relational mapping* (ORM) yang menyebabkan efektivitas peningkatan kinerja tidak seefektif sebagaimana diterapkan pada *relational database* konvensional. Metode *indexing* dan *table partitioning* akan diterapkan dalam proses *database performance tuning* untuk mengukur efektivitas peningkatan kinerja database. Hasil penelitian menunjukkan bahwa *database performance tuning* dengan kombinasi metode *indexing* dan *table partitioning* berhasil meningkatkan kinerja *database* dengan peningkatan kinerja yang paling tinggi.

Kata kunci: *database performance tuning, indexing, object relational mapping, table partitioning*

Comparison of Database Performance Tuning Through The Object Relational Mapping Method on SQL Server

Abstract

In the database performance tuning process, there are several problems when applied to an object relational mapping (ORM) based system which causes the effectiveness of performance improvement to be not as effective as applied to conventional relational databases. The indexing and table partitioning methods will be applied in the database performance tuning process to measure the effectiveness of improving database performance. The results showed that database performance tuning using combination of indexing and table partitioning methods succeeded in improving database performance with the highest performance improvement..

Keywords: *database performance tuning, indexing, object relational mapping, table partitioning.*

1. PENDAHULUAN

Object relational mapping (ORM) merupakan metode *programming* untuk menjembatani antara *object relational* dan *object oriented* pada *relational database management system* konvensional. ORM menambah *layer* di antara aplikasi dan data sehingga menambah kompleksitas dalam proses *data retrieval* dan menambah kemungkinan isu performa pada area aplikasi yang pada dasarnya memiliki kemungkinan isu performa terbesar dibandingkan area lain seperti desain sistem, *database*, serta sistem operasi dan perangkat keras [1]. Oleh karena itu, *database performance tuning* perlu dilakukan agar kinerja *relational database* yang digunakan dalam ORM tersebut dapat berjalan dengan optimal.

Terdapat beberapa metode yang dilakukan dalam melakukan *database performance tuning* di antaranya: normalisasi, *cost-based optimisation* (CBO), *indexing, table partitioning, load balancing, sort tuning, table views, cache management, setbased logic,* dan *parallelism* [2]. Metode-metode tersebut telah banyak diterapkan dalam *database performance tuning* pada *relational database* konvensional dan telah terbukti dapat memberikan peningkatan kinerja. Akan tetapi, untuk penerapannya pada sistem berbasis ORM yang memiliki kompleksitas lebih tinggi dibandingkan *relational database* konvensional, perlu dilakukan kajian yang lebih mendalam untuk mengukur efektivitas metode *database performance tuning* dalam memberikan peningkatan kinerja pada sistem berbasis ORM tersebut.

Penelitian terkait pernah dilakukan oleh Samidi dkk. [3] yang meneliti proses *database tuning* dengan menggunakan metode *indexing* dan *table partitioning* pada PostgreSQL. Hasil penelitian menunjukkan bahwa penggunaan *primary key, foreign key, index* dan *table partition* meningkatkan *response time* yang lebih baik dalam eksekusi *query*. Kemudian, penelitian lain yang dilakukan oleh Samidi dkk. [4] yang membandingkan waktu respons *database* sebelum dan sesudah proses *database tuning* pada Sistem Informasi Rumah Sakit, diperoleh hasil waktu pengujian yang lebih cepat signifikan setelah penyetelan menggunakan *index* dan optimasi

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

query. Penelitian selanjutnya dilakukan oleh Samidi dkk. [5] yang membahas proses *database tuning* menggunakan metode SQL *subquery* klausa IN dan EXIST dan *indexing*. Dari penelitian tersebut diperoleh hasil bahwa metode *subquery* klausa IN dan EXIST memiliki rata-rata waktu pengujian lebih cepat setelah diberikan pengindeksan. Penelitian lainnya yang dilakukan oleh Islam K dkk. [6] yang meneliti terkait perbandingan performa *query* pada RDBMS SQL Server 2016, Oracle 12c dan MySQL 5.7 menggunakan metode pengujian waktu eksekusi berbagai jenis *query*. Berdasarkan penelitian tersebut, diperoleh hasil bahwa di antara ketiga RDBMS tersebut, MySQL memiliki kinerja yang paling cepat dan memiliki waktu eksekusi *query* paling baik. Selanjutnya, penelitian yang dilakukan oleh Samidi dkk. [7] yang meneliti tentang *database performance tuning* menggunakan metode *distributed sharding* pada *database* MySQL menghasilkan temuan bahwa *response time* menggunakan metode *single database* biasa lebih baik dari pada menggunakan metode *distributed sharding.*

Indexing dan *table partitioning* merupakan metode yang dapat digunakan untuk meningkatkan performa *query* pada *relational database* [8][9]. Dalam penelitian ini, *indexing dan table partitioning* akan diimplementasikan sebagai metode *database performance tuning* pada sistem berbasis ORM sehingga diharapkan kinerja *database* dapat meningkat. *Database performance tuning* telah banyak diterapkan pada *relational database* konvensional dan telah terbukti dapat memberikan peningkatan kinerja pada *database* tersebut. Akan tetapi, untuk penerapannya pada sistem berbasis ORM yang lebih kompleks, belum banyak ditemukan hasil yang dapat membuktikan bahwa metode tersebut dapat memberikan peningkatan kinerja yang sama pada sistem berbaris ORM sebagaimana diterapkan dalam *relational database* konvensional.

Dalam penelitian ini, pengukuran kinerja yang dilakukan dengan mengukur *query execution time*. Penulis tidak mengukur parameter lain yang mungkin terpengaruh akibat *database performance tuning* seperti utilitas prosesor, konsumsi memori, dan I/O *operation* baik dari sisi *database server* maupun dari sisi sistem sebagai *database client*. *Relational Database Management System* (RDBMS) yang digunakan dalam penelitian ini adalah Microsoft SQL Server 2022 Developer Edition. Sistem yang digunakan berjalan pada *platform* .NET 6 dengan menggunakan ORM Entity Framework Core 7.0.

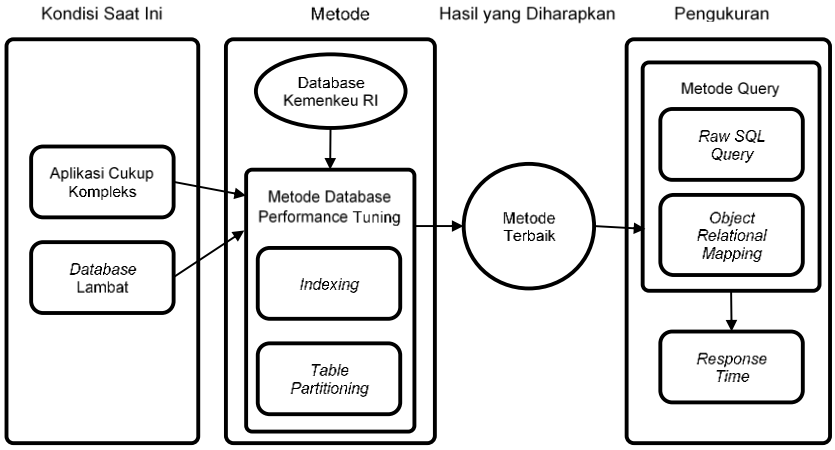
Berangkat dari adanya gap praktik dan penelitian sebelumnya, maka perumusan masalah dalam penelitian ini adalah seberapa besar peningkatan kinerja metode *indexing* dan *table partition* dalam *database performance tuning* apabila diterapkan pada sistem berbasis ORM? Penelitian ini bertujuan untuk menerapkan *database performance tuning* dengan metode *indexing* dan *table partition* pada sistem berbasis ORM sehingga kinerja *database* meningkat. Dengan dilakukannya penelitian ini diharapkan dapat meningkatkan kinerja sistem informasi cuti pegawai Kementerian Keuangan yang memiliki sistem berbasis ORM sehingga dapat memberikan *service* yang optimal terkait sistem informasi pengelolaan cuti bagi para pegawai Kementerian Keuangan di seluruh Indonesia.

Database performance tuning merupakan kegiatan yang bertujuan untuk meningkatkan kinerja *database* dengan cara meningkatkan nilai tertentu yang dapat digunakan sebagai ukuran aspek kinerja *database* seperti *throughput, response time*, utilisasi sumber daya, dan *processing power* [10][11]. Terdapat beberapa kategori metode dalam melakukan *databasze performance tuning* yaitu sebagai berikut [2].

1. Pertimbangan rancangan basis data: normalisasi.
2. Optimasi eksekusi *query*: pengoptimasi berbasis beban, indeksasi, mempartisi, menyeimbangkan beban, memvariasikan tingkat isolasi transaksi.
3. Rancangan *query* SQL: menyetel penyortiran dan agregasi, tampilan tabel, manajemen *cache*, logika berbasis himpunan bukan logika iteratif, paralelisme, dan penggunaan yang tepat dari pentipean data.

Pada penelitian ini akan digunakan metode *tuning* yang bersifat *physical programming* yaitu metode yang melibatkan optimasi penyimpanan fisik dalam sistem *database* seperti *partitioning, indexing*, kompresi data dan klasterisasi data [12] termasuk juga dalam metode ini *tablespace* dalam sistem *database* Oracle [13]. *Index* merupakan struktur akses yang bersifat *auxilary* yang digunakan untuk mempercepat pengambilan *record* berdasarkan kondisi tertentu [14]. Fungsi indeks dalam basis data mirip dengan fungsi indeks dalam buku teks [4] yang dapat menyederhanakan penelusuran sebuah data karena jika tanpa *index*, data harus ditelusuri baris demi baris dalam sebuah tabel yang dapat menyebabkan penelusuran data menjadi lebih lambat [15] terlebih jika tabel memiliki jumlah baris yang banyak. Oleh karena itu, *index* merupakan komponen yang cukup penting dalam usaha mencapai kinerja *database* yang optimal [16]. *Index* dapat didefinisikan baik secara implisit melalui *primary key* atau *unique constraint* maupun secara eksplisit melalui perintah SQL *create index* [17]. *Indexing* merupakan kegiatan memilih atau membuat *index* pada *database* yang bertujuan untuk meringankan beban kerja *database* dalam memproses *query* [18]. *Indexing* harus dilakukan secara cermat agar dapat menyeimbangkan antara beban kerja *read* dan *write* pada *database* [19]. Sedang *table partitioning* merupakan proses untuk membagi sebuah tabel menjadi bagian-bagian yang lebih kecil berdasarkan kriteria tertentu [20] sehingga tabel dapat disimpan, dikelola, dan diakses pada tingkat ketelitian yang lebih baik [21].

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



Gambar 1. Kerangka Konsep

Dari rumusan masalah yang telah dijelaskan di atas, maka penulis memiliki konsep penyelesaian yang dirangkum dalam kerangka konsep penelitian ini seperti terlihat pada Gambar 1, berawal dari kondisi saat ini yang menunjukkan aplikasi yang cukup kompleks dan kinerja database lambat. Terhadap database tersebut dilakukan *performance tuning* dengan metode *indexing* dan *partition table* sehingga diharapkan diperoleh metode *performance tuning* terbaik berdasarkan pengukuran *response time* melalui metode *query raw SQL query* dan *object relational mapping*.

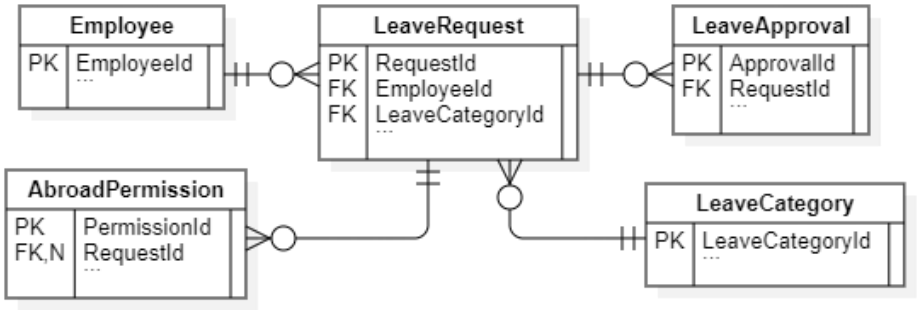
2. METODE PENELITIAN

Dalam penelitian ini, metode yang digunakan adalah metode eksperimental. Pada metode ini, peneliti melakukan perubahan-perubahan terhadap variabel-variabel untuk diteliti akibat dari perubahan tersebut [22]. *Database* yang menjadi obyek dalam penelitian ini adalah *database* Kementerian Keuangan yang digunakan dalam sistem informasi cuti pegawai. Sebagaimana ditunjukkan oleh Gambar 2, tabel yang akan diteliti adalah tabel permohonan cuti pegawai yaitu tabel LeaveRequest dan persetujuan cuti yaitu LeaveApprovel beserta tabel-tabel terkait yang akan dilakukan *join* dengan tabel permohonan cuti pegawai tersebut yang terdiri dari tabel data pegawai yaitu tabel Employee dan tabel izin luar negeri apabila selama cuti pegawai melakukan perjalanan ke luar negeri yaitu tabel AbroadPermission.

Instrumentasi yang digunakan dalam penelitian ini berupa server *database* dengan sistem operasi Microsoft Windows Server 2016 Datacenter 64 bit, prosesor Intel Xeon E5-2667 2.9 GHz (4 core), RAM 24 GB, HDD 399 GB. RDBMS yang digunakan adalah Microsoft SQL Server 2017 Developer Edition (64-bit). Pengujian dilakukan melalui program *console* yang berjalan pada *platform* .NET 6 yang ditulis dengan bahasa C#. Sedangkan ORM yang digunakan adalah Entity Framework Core 7.

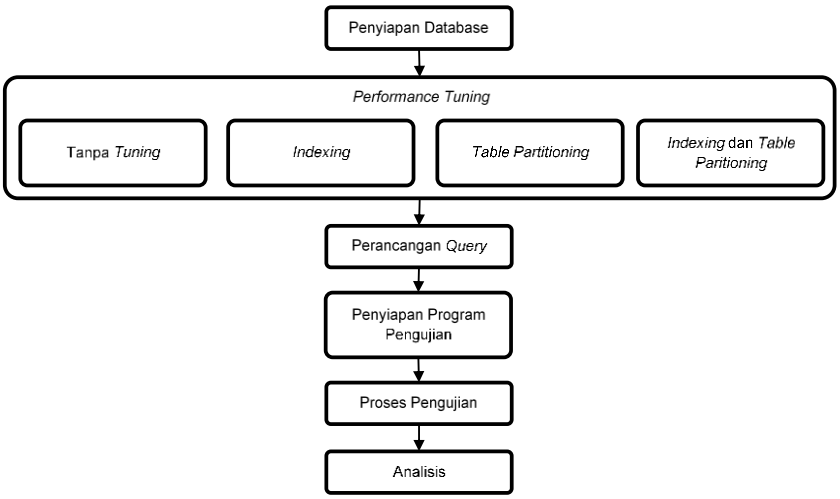
Skenario pengujian akan dilakukan pada tabel yang telah dilakukan *performance tuning* dengan *indexing*, *table partition* dan gabungan keduanya yaitu:

1. tabel yang tidak lakukan *indexing* maupun *table partitioning*,
2. tabel yang telah dilakukan *indexing* tetapi tidak dilakukan *table partitioning*,
3. tabel yang telah dilakukan *partitioning* tetapi tidak dilakukan *indexing*, dan
4. tabel yang telah dilakukan *indexing* dan *partitioning*.



Gambar 2. *Entity Relationship Diagram* (ERD) Sistem Informasi Cuti Pegawai

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



Gambar 3. Alur Penelitian

Seperti terlihat pada Gambar 3, langkah-langkah yang akan ditempuh pada penelitian ini dimulai dengan tahap penyiapan database kemudian dilanjutkan tahap *performance tuning*, tahap perancangan *query*, tahap penyiapan program pengujian, tahap pengujian dan diakhiri tahap analisis.

3. HASIL DAN PEMBAHASAN

3.1. Tahap Penyiapan *Database*

Langkah yang akan dilakukan pertama kali adalah penyiapan *database* yaitu menyalin *database* master ke dalam *database* pengujian untuk keperluan penelitian. Menyalin *database* dilakukan dengan cara melakukan *backup database* master kemudian membuat *database* baru lalu melakukan *restore backup* database ke database penelitian. Pertama-tama melakukan *backup database* yang dilakukan dengan menjalankan perintah BACKUP DATABASE. Kemudian membuat *database* baru yang akan digunakan untuk penelitian dengan menjalankan perintah CREATE DATABASE. Lalu yang terakhir adalah melakukan *restore* hasil *database* hasil *backup* ke dalam *database* penelitian yaitu dengan menjalankan perintah berikut RESTORE DATABASE.

3.2. Tahap *Performance Tuning*

Selanjutnya, dari *database* penelitian tersebut, pada tabel LeaveRequest dan LeaveApproval serta tabel-tabel terkait akan dilakukan dilakukan *performance tuning* dengan metode *indexing* dan *table partitioning*. Untuk langkah *tuning* pertama adalah dengan metode *indexing*. Terdapat lima kandidat kolom dari dua tabel yang akan dilakukan *indexing* yaitu kolom RequestId dari tabel LeaveApproval dan kolom EmployeeId, BeginDate, EndDate, RequestId dari tabel AbroadPermission. Kelima kolom tersebut merupakan kolom yang berkaitan erat dengan tabel LeaveRequest sehingga sangat berpotensi dapat mempercepat akses database apabila dilakukan *indexing*. Adapun perintah SQL yang akan dijalankan dalam proses *indexing* tersebut adalah dengan menjalankan perintah CREATE UNIQUE INDEX pada tabel LeaveApproval dan perintah CREATE INDEX pada tabel AbroadRequest dan AbroadPermission.

Langkah *tuning* berikutnya adalah *tabel partitioning*. *Partitioning* yang akan dilakukan adalah dengan membuat partisi tabel berdasarkan tahun mulai cuti. Partisi akan dibagi menjadi enam bagian yaitu tahun 2017 ke bawah, antara tahun 2017 dan 2018, antara 2018 dan 2019, antara tahun 2020 dan 2021, antara tahun 2021 dan 2022, serta tahun 2022 ke atas. Dalam SQL Server, langkah pertama untuk melakukan *table partitioning* adalah dengan membuat fungsi partisi dan skema partisi yaitu dengan menjalankan perintah CREATE PARTITION FUNCTION dan perintah CREATE PARTITION. Kemudian, menambahkan tabel partisi ke dalam tiap-tiap tabel menggunakan perintah ALTER TABLE dengan ADD CONSTRAINT dan perintah CREATE CLUSTERED INDEX Akan tetapi, sebelum dapat menambahkan partisi, apabila sudah terdapat *clustered index* pada tabel yang telah ada, perlu dihapus dulu *clustered index* tersebut menggunakan perintah ALTER TABLE dengan DROP CONSTRAINT.

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

3.3. Tahap Perancangan *Query*

Langkah berikutnya adalah merancang *query* yang akan dijalankan dalam proses pengujian. Perintah yang akan dijalankan adalah SELECT dengan kombinasi perintah JOIN, OUTER APPLY serta perintah SQL lainnya untuk mendapatkan *record* sesuai dengan kebutuhan sistem informasi cuti. Adapun *query* yang akan dijalankan adalah menampilkan daftar permintaan cuti semua pegawai yaitu dengan *script* SQL sebagai berikut.

```
SELECT
   m.RequestId,
   ... dst
FROM
   Employee AS pml RIGHT OUTER JOIN
   Employee AS pms1 RIGHT OUTER JOIN
   LeaveRequest AS m INNER JOIN
   LeaveCategory AS j
     ON m.LeaveCategoryId = j.LeaveCategoryId LEFT OUTER JOIN
   AbroadPermission AS l2
     ON m.RequestId = l2.RequestId
     ON pms1.EmployeeId = m.FirstApproverId
     ON pml.EmployeeId = m.EmployeeIdPlt LEFT OUTER JOIN
   Employee AS pms2
     ON m.SecondApproverId = pms2.EmployeeId LEFT OUTER JOIN
   Employee AS psl RIGHT OUTER JOIN
   LeaveApproval AS s LEFT OUTER JOIN
   Employee AS psb
     ON s.ValidatedBy = psb.EmployeeId
     ON psl.EmployeeId = s.EmployeeIdPlt
     ON m.RequestId = s.IDPermohonan
    OUTER APPLY
    (
     SELECT TOP(1) * FROM AbroadPermission a
       WHERE m.EmployeeId = a.EmployeeId
         AND m.BeginDate = a.BeginDate
         AND m.EndDate = a.EndDate
     ORDER BY a.PermissionId DESC
    ) l
ORDER BY m.RequestId DESC;
```

Pada *query* daftar permintaan cuti pegawai di atas, terdapat klausa join yang menghubungkan antara tabel Employee, LeaveRequest, LeaveApproval, LeaveCategory dan AbroadPermission dengan metode LEFT OUTER JOIN, RIGHT OUTER JOIN ataupun INNER JOIN. Selain dengan klausa join, terdapat klausa OUTER APPLY yang menghubungkan tabel LeaveRequest dengan AbroadPermission berdasarkan ID pegawai, tanggal mulai cuti dan tanggal selesai cuti.

3.4. Tahap Penyiapan Program Pengujian

Selanjutnya, *query* tersebut akan dijalankan dan diterjemahkan menjadi *query* ORM. *Query* tersebut akan dijalankan oleh sebuah program .NET console. Langkah pertama yang dilakukan dalam penyiapan program pengujian adalah membuat *project* .NET console. Lalu menambahkan *package* Microsoft.Data.SqlClient yang diperlukan untuk menjalankan *raw query* dan Microsoft.EntityFrameworkCore.SqlServer untuk menjalankan ORM *query*. Selanjutnya, menulis program untuk mengkoneksikan dengan server, menjalankan *query* dan menampilkan serta menyimpan hasilnya ke dalam sebuah file csv yang akan dipanggil setiap kali menjalankan *query*. Adapun *query* ORM yang akan dijalankan adalah sebagai berikut.

```
from m in context.LeaveRequest
  let i = context.AbroadPermission
    .Where(l => m.EmployeeId == l.EmployeeId
      && m.BeginDate == l.BeginDate
      && m.EndDate == l.EndDate)
```

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

```
    .OrderByDescending(mil => mil.PermissionId)
    .FirstOrDefault()
orderby m.RequestId descending
select new
{
  RequestId = m.RequestId,
  ... dst
}
```

Query ORM daftar permintaan cuti pegawai di atas menghasilkan proyeksi yang sama dengan *query* SQL sebelumnya. Terdapat klausa let diikuti dengan where yang ekuivalen dengan klausa OUTER APPLY pada SQL.

3.5. Tahap Pengujian

Tahap berikutnya adalah tahap pengujian. Jumlah batasan *record* dalam proses pengujian *query* untuk setiap metode *performance tuning* dan metode *query* akan ditingkatkan secara bertahap. Setiap proses pengujian akan dilakukan beberapa kali dan dicatat *response time*-nya kemudian dihitung rata-ratanya. Pengujian *query* akan dilakukan pada database yang tidak dilakukan *tuning*, kemudian database yang dilakukan *indexing*, kemudian database yang dilakukan *indexing* dan *table partitioning* serta terakhir database yang dilakukan *table partitioning* saja. Pengujian dilakukan untuk *query* yang menggunakan *raw* SQL dan *query* ORM. Untuk metode pengujian menggunakan *raw* SQL diperoleh hasil sebagaimana ditunjukkan pada

Tabel 1. Sedangkan untuk pengujian *query* ORM diperoleh hasil sebagai sebagaimana ditunjukkan pada Tabel 2.

Tabel 1. Hasil Pengujian Setiap Metode *Tuning* dengan Pengujian Metode *Raw* SQL

Jumlah *Record*	***Response Time* (detik)**			
	Tanpa *Tuning*	***Indexing***	***Indexing + Table Partitioning***	***Table Partitioning***
10	0,080	0,050	0,066	0,117
50	0,117	0,009	0,008	0,232
100	0,136	0,011	0,010	0,316
500	0,856	0,047	0,040	1,414
1.000	1,518	0,090	0,080	2,806
5.000	7,231	0,405	0,330	14,355
10.000	14,112	0,832	0,654	28,527
35.000	50,538	2,735	2,452	98,046
50.000	69,659	3,737	3,743	135,050
75.000	102,712	5,575	5,708	202,367
95.000	134,388	7,143	6,703	262,745
110.000	156,101	8,227	7,844	297,048
135.000	193,846	10,375	9,298	363,734
160.000	217,498	11,363	11,043	424,536
185.000	252,200	13,684	12,529	496,845
210.000	291,153	15,765	14,402	572,987
235.000	334,204	17,567	16,215	645,598
260.000	372,076	19,578	18,031	730,905
285.000	389,433	22,745	18,975	815,090
310.000	437,113	24,082	20,241	885,414
335.000	471,849	23,376	22,211	938,820
349.582	499,195	24,766	21,962	954,662

Tabel 2. Hasil Pengujian Setiap Metode *Tuning* dengan Pengujian Metode ORM

Jumlah *Record*	***Response Time* (detik)**			
	Tanpa *Tuning*	***Indexing***	***Indexing + Table Partitioning***	***Table Partitioning***
10	0,395	0,356	0,348	0,569
50	0,330	0,095	0,084	0,204
100	0,317	0,026	0,014	0,086
500	1,491	0,085	0,048	0,130

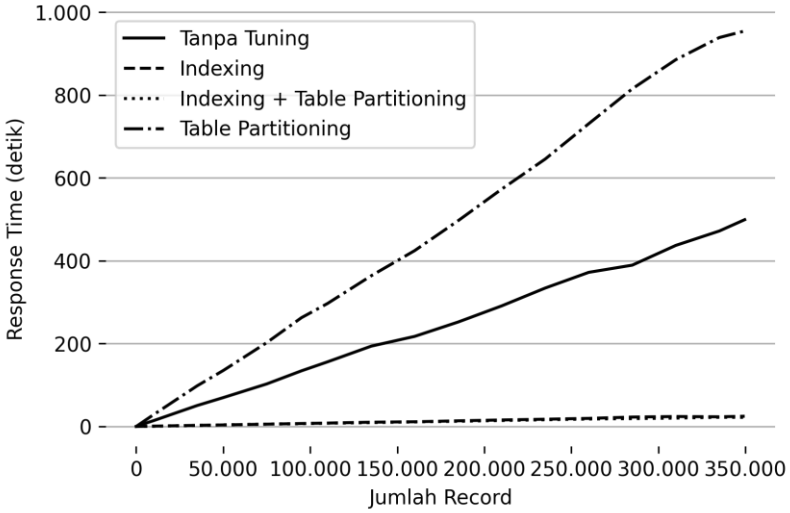
p-ISSN: 2775-4227
e-ISSN: 2775-4219

Jumlah *Record*	*Response Time* (detik)			
	Tanpa *Tuning*	*Indexing*	*Indexing + Table Partitioning*	*Table Partitioning*
1.000	3,028	0,182	0,097	0,266
5.000	15,517	0,607	0,597	0,970
10.000	30,237	1,126	0,934	1,570
35.000	111,126	3,783	3,308	5,954
50.000	162,074	5,378	4,622	8,420
75.000	240,069	8,458	7,026	13,501
95.000	297,148	10,763	8,859	16,531
110.000	352,401	12,643	10,327	20,297
135.000	434,397	15,312	12,433	25,885
160.000	511,901	17,593	14,657	30,571
185.000	589,770	21,543	17,660	34,797
210.000	684,122	23,868	22,476	38,139
235.000	738,487	25,493	21,610	42,132
260.000	797,517	27,535	23,936	46,513
285.000	866,015	30,213	25,992	49,965
310.000	913,782	33,711	28,619	52,005
335.000	1.009,840	34,397	32,240	57,523
349.582	1.081,683	36,297	32,213	60,200

3.6. Tahap Analisis

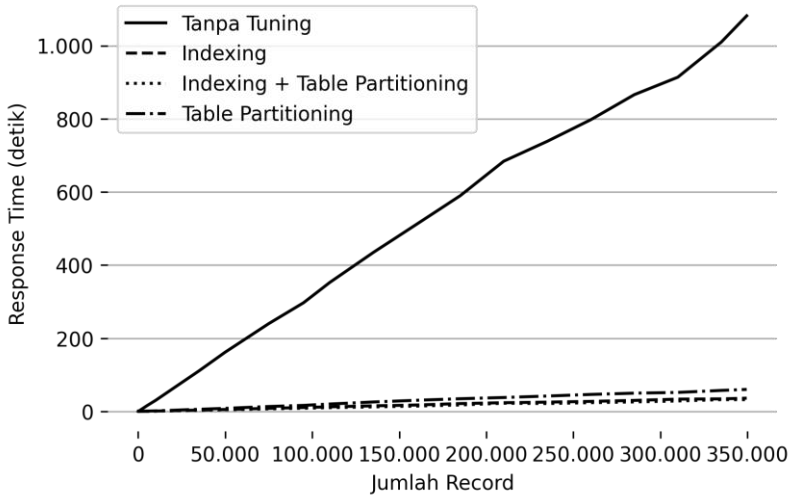
Dari data hasil pengujian, kemudian diolah lebih lanjut dan disajikan dalam sebuah grafik untuk mendapatkan *insight* dan simpulan mengenai kinerja tiap-tiap *database tuning*. Berdasarkan Gambar 4, terdapat peningkatan *respons time* yang sangat signifikan pada database setelah dilakukan *database tuning* dengan metode *indexing* dan metode gabungan antara *indexing* dan *table partitioning*. Namun sebaliknya, pada pengujian dengan metode *raw* SQL ini, terdapat penurunan kinerja yang cukup signifikan setelah dilakukan *database tuning* dengan metode *table partitioning* saja tanpa *indexing*. Pada metode ini metode yang memberikan peningkatan kinerja paling tinggi merupakan metode gabungan kemudian metode *indexing* saja.

Untuk pengujian dengan metode ORM, sebagaimana ditunjukkan oleh Gambar 5, baik *tuning* yang dilakukan dengan metode *indexing*, *table partitioning* dan gabungan kedua metode tersebut, ketiga-tiganya menghasilkan peningkatan kinerja yang sangat signifikan. Metode *table partitioning* yang memberikan penurunan kinerja pada metode pengujian dengan *raw* SQL, pada metode ORM ini memberikan peningkatan kinerja yang sangat signifikan walaupun sedikit di bawah metode *indexing* dan metode gabungan. Pada metode pengujian ini, metode yang memiliki *response time* paling cepat merupakan metode gabungan *indexing* dan *table partitioning* kemudian metode *indexing* saja lalu *table partitioning*.



Gambar 4. Grafik Kinerja Setiap Metode *Tuning* dengan Metode Pengujian Raw SQL

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



Gambar 5. Grafik Kinerja Setiap Metode *Tuning* dengan Metode Pengujian ORM

4. KESIMPULAN

Berdasarkan penelitian yang dilakukan penulis, dapat disimpulkan bahwa metode kombinasi *indexing* dan *table partitioning* merupakan metode terbaik dalam proses *database performance tuning* baik dengan metode pengujian *raw* SQL maupun ORM. Metode terbaik selanjutnya adalah metode *indexing* saja. Untuk metode *table partitioning* saja, terdapat perbedaan hasil yang saling bertolak belakang antara metode pengujian *raw* SQL dengan metode ORM. Pada metode pengujian *raw* SQL, *table partitioning* memberikan penurunan kinerja yang cukup signifikan pada database. Sedangkan pada metode ORM, *table partitioning* memberikan peningkatan kinerja yang sangat signifikan sedikit di bawah metode *indexing* dan metode gabungan..

DAFTAR PUSTAKA

[1] S. J. Kamatkar, A. Kamble, A. Viloria, L. Hernández-Fernandez, and E. G. Cali, "Database Performance Tuning and Query Optimization," in *Data Mining and Big Data*, Y. Tan, Y. Shi, and Q. Tang, Eds., Cham: Springer International Publishing, 2018, pp. 3–11. doi: 10.1007/978-3-319-93803-5_1.

[2] D. Colley and C. Stanier, "Identifying New Directions in Database Performance Tuning," *Procedia Comput Sci*, vol. 121, pp. 260–265, Jan. 2017, doi: 10.1016/J.PROCS.2017.11.036.

[3] Samidi, Y. Virmansyah, R. Y. Suladi, and A. B. Lesmana, "Optimasi Database dengan Metode Index dan Partisi Tabel Database Postgresql pada Aplikasi E-Commerce. Studi pada Aplikasi Tokopintar," *Jurnal Pendidikan Tambusai*, vol. 6, no. 1, pp. 2094–2102, 2022.

[4] Samidi, D. Iskandar, M. Fachruroji, W. A. S. Wibowo, and A. A. Khaerani, "Database Tuning in Hospital Applications Using Table Indexing and Query Optimization," *Jurnal Pendidikan Tambusai*, vol. 6, no. 1, pp. 1960–1967, 2022.

[5] Samidi, D. Andharu, F. Widyarto, S. S. Syahdinullah, and S. Y. W. Eka, "Optimasi Database Menggunakan SQL Kueri Klausa IN dan EXIST pada Database Oracle 12c. Studi Kasus pada Aplikasi di Badan Nasional Pencarian dan Pertolongan (BASARNAS)," *Jurnal Pendidikan Tambusai*, vol. 6, no. 1, pp. 2297–2306, 2022.

[6] K. Islam, K. Ahsan, S. Bari, M. Saeed, and S. A. Ali, "Huge and Real-Time Database Systems: A Comparative Study and Review for SQL Server 2016, Oracle 12c & MySQL 5.7 for Personal Computer," *Journal of Basic & Applied Sciences*, vol. 13, pp. 481–490, Sep. 2017, doi: 10.6000/1927-5129.2017.13.79.

[7] S. Samidi, R. Y. Suladi, and A. B. Lesmana, "Implementation of Database Distributed Sharding Horizontal Partition in MySQL. Case Study of Application of Food Serving On Kemkes," *Jurnal Sisfotek Global*, vol. 12, no. 1, pp. 50–57, Mar. 2022, doi: 10.38101/sisfotek.v12i1.477.

[8] G. Paludo Licks, J. Colleoni Couto, P. de Fátima Miehe, R. de Paris, D. Dubugras Ruiz, and F. Meneguzzi, "SmartIX: A database indexing agent based on reinforcement learning," *Applied*

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

Intelligence, vol. 50, no. 8, pp. 2575–2588, Aug. 2020, doi: 10.1007/s10489-020-01674-8.

[9] A. Uzzaman, M. M. I. Jim, N. Nishat, and J. Nahar, "Optimizing SQL Databases for Big Data Workloads: Techniques and Best Practices," *Academic Journal on Business Administration, Innovation & Sustainability*, vol. 4, no. 3, pp. 15–29, Jun. 2024, doi: 10.69593/ajbais.v4i3.78.

[10] K. Mózsi and A. Kiss, "A Session-based Approach to Autonomous Database Tuning," *Acta Polytechnica Hungarica*, vol. 17, no. 1, pp. 7–24, 2020, doi: 10.12700/APH.17.1.2020.1.1.

[11] S. van Wouw, J. Viña, A. Iosup, and D. Epema, "An Empirical Performance Evaluation of Distributed SQL Query Engines," in *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, New York, NY, USA: ACM, Jan. 2015, pp. 123–131. doi: 10.1145/2668930.2688053.

[12] I. Šušter and T. Ranisavljević, "Optimization of MySQL Database," *Journal of Process Management and New Technologies*, vol. 11, no. 1–2, pp. 141–151, Jun. 2023.

[13] W. Khan, W. Ahmad, B. Luo, and E. Ahmed, "SQL Database with Physical Database Tuning Technique and NoSQL Graph Database Comparisons," in *Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, IEEE, 2019.

[14] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*. Addison-Wesley, 2015.

[15] 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.

[16] X. Zhou, C. Chai, G. Li, and J. Sun, "Database Meets Artificial Intelligence: A Survey," *IEEE Trans Knowl Data Eng*, vol. 34, no. 3, pp. 1096–1116, Mar. 2022, doi: 10.1109/TKDE.2020.2994641.

[17] M. Kvet and K. Matiasko, "Analysis of current trends in relational database indexing," in *2020 International Conference on Smart Systems and Technologies (SST)*, IEEE, Oct. 2020, pp. 109–114. doi: 10.1109/SST49455.2020.9264034.

[18] Z. Sadri, L. Gruenwald, and E. Lead, "DRLindex: Deep reinforcement learning index advisor for a cluster database," in *ACM International Conference Proceeding Series*, Association for Computing Machinery, Aug. 2020. doi: 10.1145/3410566.3410603.

[19] M. Valavala and W. Alhamdani, "Automatic Database Index Tuning Using Machine Learning," in *Proceedings of the 6th International Conference on Inventive Computation Technologies, ICICT 2021*, Institute of Electrical and Electronics Engineers Inc., Jan. 2021, pp. 523–530. doi: 10.1109/ICICT50816.2021.9358646.

[20] P. Bednarczuk, "Optimization in Very Large Databases by Partitioning Tables," *Informatyka, Automatyka, Pomiary w Gospodarce i Ochronie Środowiska*, vol. 10, no. 3, pp. 95–98, Sep. 2020, doi: 10.35784/iapgos.2056.

[21] M. Olma, M. Karpathiotakis, I. Alagiannis, M. Athanassoulis, and A. Ailamaki, "Adaptive partitioning and indexing for in situ query processing," in *VLDB Journal*, Springer, Jan. 2020, pp. 569–591. doi: 10.1007/s00778-019-00580-x.

[22] R. Pamungkas, "Optimalisasi Query dalam Basis Data MySQL Menggunakan Index," *Journal of Computer, Information System, & Technology Management*, vol. 1, no. 2, pp. 27–31, Apr. 2018.