



Implementation of Database Tuning Through Indexing and Query Optimization in an Academic Questionnaire System

Penerapan Database Tuning Melalui Indexing dan Optimasi Query pada Sistem Kuesioner Akademik

Ikhwan^{1*}, Angga Pramadjaya²

^{1,2}Program Studi Sistem Informasi, Fakultas Ilmu Komputer, Universitas Pamulang, Indonesia

E-Mail: ¹dosen03358@unpam.ac.id, ²anggapramadjaya@unpam.ac.id

Received Feb 18th 2026; Revised Mar 24th 2026; Accepted Mar 30th 2026; Available Online Apr 24th 2026

Corresponding Author: Ikhwan

Copyright ©2026 by Authors, Published by Institut Riset dan Publikasi Indonesia (IRPI)

Abstract

An academic questionnaire system is designed to collect evaluation information from lecturers and students. These evaluations include students' assessments of lecturers' performance as well as evaluations by students or lecturers regarding campus services. Over time, the amount of data stored in the database continues to grow, leading to slower system performance. Therefore, this study aims to address this issue by implementing database tuning through indexing and query optimization. This method is applied to accelerate data retrieval when SQL queries are executed. Indexing can be applied to tables with a large number of rows, and appropriately designed queries can help generate indexes more quickly. There are 3,107,177 data records in this study; however, approximately 3,000,000 records are utilized for analysis. Testing is conducted using B-tree indexing on database tables and optimizing SQL SELECT queries using WHERE and GROUP BY clauses. The expected result is faster, more efficient data queries, improving overall database performance after implementing table indexing and query optimization.

Keywords: *Academic Questionnaire System, Database Tuning, Indexing, Optimization, Query.*

Abstrak

Kuesioner akademik adalah sistem yang dibuat untuk mengumpulkan informasi penilaian dari dosen dan mahasiswa. Penilaian tersebut meliputi penilaian mahasiswa terhadap kinerja dosen dan penilaian mahasiswa atau dosen terhadap pelayanan kampus. Seiring berjalannya waktu, jumlah data yang masuk ke basis data semakin bertambah dan membuat kinerja sistem menjadi lebih lambat. Untuk itu, penelitian ini bertujuan untuk mengatasi masalah tersebut dengan menggunakan penerapan *database tuning* melalui *indexing* dan optimasi *query*. Metode ini digunakan untuk mempercepat pencarian atau pengambilan data pada saat *query SQL* dijalankan. *Indexing* dapat diimplementasikan pada tabel dengan jumlah baris data yang banyak, dan penggunaan *query* yang tepat dapat membantu memberikan *indexing* yang cepat. Terdapat 3.107.177 baris data pada penelitian ini, namun pada penelitian ini hanya sekitar 3.000.000 data yang dipakai. Pengujian dilakukan dengan menggunakan *B-tree Indexing* pada tabel dan optimasi kueri *SQL SELECT* dengan penggunaan klausa *WHERE* dan *GROUP BY*. Hasil yang diharapkan berupa kueri data yang menjadi lebih cepat dan efektif untuk meningkatkan kinerja pada basis data setelah melakukan *Table Indexing* dan Optimasi *Query*.

Kata Kunci: *Indexing, Optimasi, Query, Sistem Kuesioner Akademik, Tuning Database*

1. PENDAHULUAN

Perkembangan teknologi informasi telah mendorong transformasi digital dalam pengelolaan sistem akademik di perguruan tinggi [1]. Salah satu implementasi penting dari transformasi tersebut adalah sistem kuesioner akademik yang digunakan untuk mengumpulkan data evaluasi dari mahasiswa dan dosen. Sistem ini berperan strategis dalam mendukung penjaminan mutu internal, pengambilan keputusan berbasis data, serta peningkatan kualitas layanan pendidikan [2]. Namun, seiring meningkatnya jumlah pengguna dan intensitas pengisian kuesioner setiap semester, volume data yang tersimpan dalam basis data mengalami pertumbuhan yang signifikan [3].

Pertumbuhan data yang masif dalam sistem kuesioner akademik menimbulkan tantangan terhadap performa basis data, terutama dalam hal waktu respon (*response time*) saat eksekusi *query* [4]. Permasalahan umum yang muncul meliputi lambatnya proses pencarian data, agregasi laporan evaluasi, serta keterlambatan dalam penyajian informasi bagi pihak manajemen [5]. Jika tidak ditangani dengan pendekatan teknis yang tepat, kondisi ini dapat menurunkan efisiensi sistem serta menghambat proses pengambilan keputusan yang membutuhkan data secara cepat dan akurat [6].

Salah satu pendekatan yang banyak digunakan untuk meningkatkan performa basis data adalah *database tuning* [7]. *Database tuning* merupakan proses optimasi struktur dan eksekusi basis data untuk meningkatkan efisiensi akses data tanpa mengubah arsitektur sistem secara keseluruhan. Dua teknik utama dalam *database tuning* yang sering diterapkan adalah *indexing* dan optimasi *query*. *Indexing* berfungsi untuk mempercepat proses pencarian data dengan menyediakan struktur akses yang lebih efisien terhadap tabel berukuran besar. Sementara itu, optimasi *query* dilakukan dengan menyusun perintah *SQL* secara lebih efektif, seperti penggunaan klausa *WHERE*, *GROUP BY*, serta pemilihan atribut yang relevan guna mengurangi beban pemrosesan [8].

Beberapa penelitian sebelumnya menunjukkan bahwa penerapan *indexing*, khususnya B-tree *indexing*, mampu secara signifikan menurunkan waktu eksekusi *query* pada tabel dengan jumlah record yang besar [9]. Penelitian lain juga menegaskan bahwa struktur *query* yang tidak optimal menjadi salah satu penyebab utama *bottleneck* pada sistem berbasis basis data relasional. Namun demikian, sebagian besar penelitian tersebut berfokus pada sistem *e-commerce*, sistem keuangan, atau *data warehouse*, sementara studi yang secara spesifik mengkaji penerapan *database tuning* pada sistem kuesioner akademik dengan karakteristik data evaluatif dan agregatif masih relatif terbatas [10].

Berdasarkan celah penelitian (*research gap*) tersebut, diperlukan kajian empiris yang menguji efektivitas penerapan *indexing* dan optimasi *query* pada sistem kuesioner akademik dengan volume data yang besar. Sistem ini memiliki karakteristik unik berupa frekuensi *query* agregasi yang tinggi, terutama dalam proses rekapitulasi penilaian dosen dan layanan kampus [11]. Oleh karena itu, pendekatan *tuning* yang tepat perlu dianalisis secara kuantitatif untuk mengetahui sejauh mana peningkatan performa yang dihasilkan. Penelitian ini bertujuan untuk menganalisis dan mengimplementasikan *database tuning* melalui penerapan B-tree *indexing* pada tabel berukuran besar serta optimasi *query SQL SELECT* menggunakan klausa *WHERE* dan *GROUP BY* pada sistem kuesioner akademik. Evaluasi dilakukan dengan membandingkan waktu eksekusi *query* sebelum dan sesudah proses *tuning* terhadap *dataset* yang berjumlah sekitar tiga juta record [12].

Kontribusi penelitian ini dibandingkan dengan penelitian sebelumnya, yaitu terletak pada penyediaan model implementatif *database tuning* yang teruji secara empiris pada sistem akademik berbasis data evaluasi, serta pemberian rekomendasi teknis yang dapat diadopsi oleh institusi pendidikan dalam meningkatkan kinerja sistem informasi akademik tanpa harus melakukan perubahan infrastruktur yang signifikan. Dengan demikian, hasil penelitian ini diharapkan dapat mendukung pengelolaan data akademik yang lebih efisien, responsif, dan berkelanjutan.

2. METODE PENELITIAN

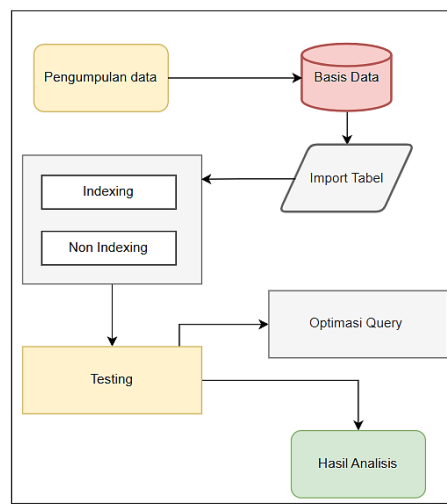
Dataset pada penelitian ini diambil dari sistem kuesioner akademik dengan jumlah baris data sebanyak 3.107.177 record. Namun, data yang digunakan pada penelitian ini sebanyak 3.000.000 record yang nantinya di-convert menjadi satu file dengan format file (.csv). Selanjutnya, dilakukan import data *SQL* ke server *localhost* pada database *MySQL*. Impor dilakukan dengan cara mengunggah file *SQL* ke dalam database *MySQL localhost* melalui tools DBMS. Setelah data masuk ke server database, dilakukan copy tabel: tabel pertama tidak berisi indeks (non-indeks) dan tabel kedua berisi indeks. Selanjutnya dilakukan tahap pengujian dengan menggunakan laptop Lenovo ideapad 110 dengan kapasitas RAM 4 GB, CPU AMD A9-9400, RADEON R5, dan sistem operasi Windows 10 Pro. Tools perangkat lunak yang digunakan yaitu Visual Studio Code, xampp dan dbeaver DBMS. Adapun tahapan penelitian dapat dilihat pada Gambar 1.

Pengujian pertama dilakukan tanpa menggunakan indeks (non-indeks) dan pada pengujian kedua dilakukan dengan menggunakan indeks. Selanjutnya pada sisi *query* perlu juga dilakukan pengujian dengan *query* tanpa optimasi dan pengujian kedua dengan *query* optimasi [13]. Perintah *SELECT WHERE* dan *GROUP BY* akan digunakan sebagai parameter pengujian. Pengujian ini dilakukan dengan membandingkan tabel yang sudah diindeks (non-indeks) dan membandingkan query *SQL* yang dioptimalkan dengan yang tidak dioptimalkan. Setelah dilakukan pengujian maka akan didapat total *response time* untuk dilakukan analisa perbandingan [14].

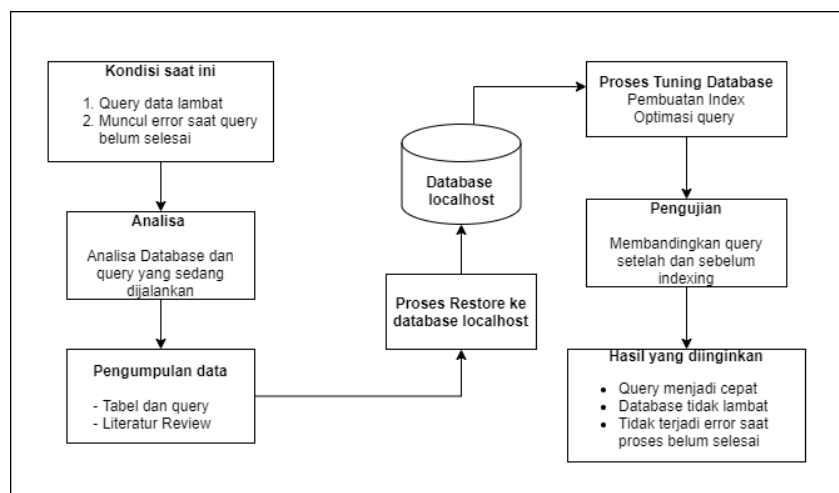
2.1 Kerangka Konsep

Dari gambaran kerangka konsep pada Gambar 2 tersebut dapat dijelaskan bahwa, masalah yang terjadi saat ini yaitu *query* data yang lambat yang menjadi penyebab munculnya error saat menjalankan *query* yang belum selesai. Setelah permasalahan tersebut diketahui, maka dapat dibuatkan analisis masalahnya, yaitu analisis terhadap database dan *query* yang sedang dijalankan apakah sudah optimal atau belum; jika belum

optimal, lakukan optimasi query agar performa query dapat menjadi lebih baik [15]. Selanjutnya, dilakukan pengumpulan data dan analisis dari penelitian sebelumnya agar dapat membandingkan tuning database dan optimasi query pada penelitian ini. Kegiatan ini dilakukan untuk menentukan solusi terbaik dalam pemecahan masalah. Pengumpulan data tabel dan *query* dilakukan dengan menarik data dari server formal ke server *localhost*. Tabel akan dilakukan backup dan selanjutnya di-copy ke server *localhost* untuk dilakukan *tuning* dan pengujian. Selain itu, *query* juga akan dibandingkan antara sebelum dan sesudah dilakukan optimasi untuk melihat apakah ada perbedaan yang signifikan terhadap masing-masing [16]. Untuk data sampel, tabel yang digunakan hanya dua tabel, yaitu *kue_responden_m* dan *kue_responden_d*. Dua tabel ini merupakan tabel transaksi sistem kuesioner akademik, di mana *kue_responden_d* merupakan tabel dengan data yang paling banyak. Karena tidak semua tabel harus dibuatkan indeks, penulis menentukan pada tabel transaksi. Setelah data masuk ke server *localhost*, selanjutnya dilakukan proses pembuatan indeks untuk tuning database dengan metode indeks yang digunakan dengan model B-tree dan pembuatan query SQL yang dioptimalkan [17].



Gambar 1 Tahapan Penelitian

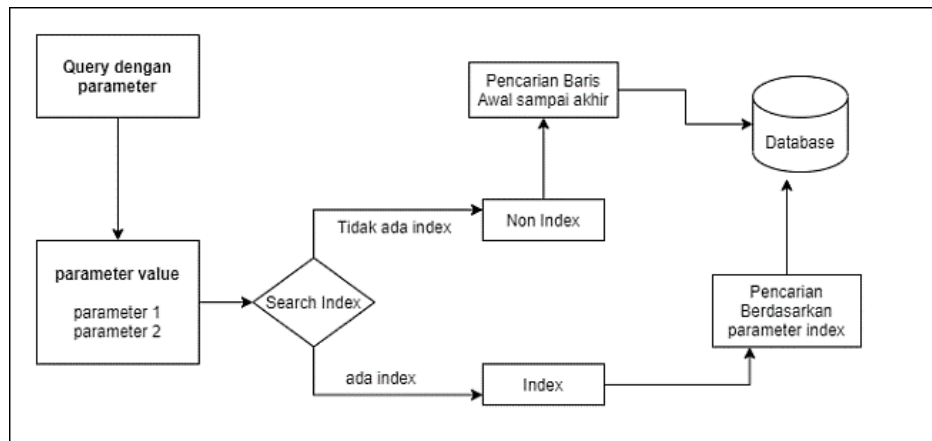


Gambar 2 Kerangka Konsep

2.2 Proses Query Data

Salah satu pendekatan yang dapat diterapkan adalah *database tuning* melalui *indexing* dan optimasi *query*. *Indexing* berfungsi untuk mempercepat proses pencarian data dengan menyediakan struktur akses yang lebih efisien dibandingkan dengan pemindaian tabel secara menyeluruh. Sementara itu, optimasi *query* bertujuan untuk menyederhanakan dan menyusun ulang perintah SQL agar dapat dieksekusi dengan biaya komputasi yang lebih rendah [18]. Kombinasi kedua pendekatan ini diharapkan mampu mengurangi waktu eksekusi, menurunkan beban I/O, serta meningkatkan stabilitas sistem [19]. Dengan demikian, kajian terhadap proses *query* data dalam Sistem Kuesioner Akademik menjadi penting untuk dianalisis secara sistematis. Evaluasi terhadap mekanisme eksekusi *query* sebelum dan sesudah penerapan *database tuning* dapat memberikan gambaran empiris mengenai efektivitas strategi yang diterapkan. Hasil analisis ini tidak

hanya relevan untuk peningkatan performa sistem kuesioner, tetapi juga dapat menjadi referensi dalam pengembangan sistem informasi akademik lainnya yang memiliki karakteristik data serupa [20].



Gambar 3 Proses Query Data

Sistem ini menggunakan *database MySQL* sebagai tempat penyimpanan dan manajemen data. Database merupakan suatu kumpulan data yang berisi informasi yang tersimpan secara sistematis. Untuk mengakses data dan entitas-entitas yang ada pada database, diperlukan *query Structured Query Language (SQL)*. Database menjadi komponen yang penting dalam sistem informasi, untuk itu perlu dilakukan pengecekan performa agar tidak terjadi *low performance* [17]. Seiring berjalannya waktu, kondisi ini membuat performa *database* menurun dan kinerja sistem menjadi lambat karena banyaknya data yang terus bertambah. Jika data sudah semakin banyak, *query* akan berjalan lebih lama dan lebih lambat, di sinilah penurunan performa sistem dan *database*. Permasalahan ini hampir dirasakan pada semua *database*, seperti Oracle, *SQLServer*, *Postgre SQL* maupun *database MySQL* [21]. Tidak ada kepedulian terhadap optimasi justru akan menurunkan performa pada sistem *database*. Hasil ini akan berakibat pada lambatnya pencarian data pada *database*. Begitu juga masalah yang dihadapi pada penelitian ini yaitu *query* data yang mulai lambat, performa *database* menjadi menurun dan efektifitas jadi berkurang [22]. Masalah ini dikarenakan tidak adanya indeks pada setiap tabel transaksi dan *query* yang tidak dioptimalkan. *Query* yang dijalankan akan membaca data baris per baris. Tabel tanpa indeks akan memengaruhi kinerja basis data. Untuk hasil *indexing* yang tepat, diperlukan tabel *indexing* dan kueri yang tepat untuk optimalnya kinerja.

Index bekerja dengan mencari daftar isi pada tabel tertentu yang telah ditentukan sebelumnya, daftar isi ini bertujuan untuk membuat *query* data melakukan pencarian berdasarkan daftar isi index yang telah dibuat [23]. Pada umumnya, indeks terbentuk saat kita membuat primary key pada tabel di *database*. Indeks akan dibentuk berdasarkan kolom-kolom yang ditentukan untuk pencarian. Namun, selain itu index juga bisa dibuat sendiri dengan menggunakan model *btree indexing* [24]. *B-tree* adalah pohon pencarian data pada basis data yang memungkinkan data yang telah disimpan dapat diakses dengan *search*, *insert*, dan *delete* secara terstruktur dan sistematis. perlunya dilakukan *tuning database* yang dapat meningkatkan kecepatan saat *query* atau pengambilan data pada *database* [25]. *Tuning* yang akan diterapkan pada penelitian ini menggunakan teknik *btree indexing* dan optimasi *query SQL* [6]. *Tuning* ini merupakan suatu proses untuk meningkatkan kinerja pada *database*, sehingga performa menjadi lebih optimal, cepat, efektif dan efisien. Gambar 3 menjelaskan skema bagaimana indeks dan *query* data berjalan pada *database*.

2.3 Perbandingan Query

Untuk menambah kualitas performa *query*, bisa ditambahkan dengan teknik optimasi *query*, yaitu suatu cara dalam pengolahan data untuk mencari jalur tercepat dalam pemrosesan pada saat pengambilan data di *database*. Optimasi *query* dapat dilakukan guna membantu percepatan pencarian data. Penggunaan *query* juga menentukan cepat atau tidaknya data yang akan ditampilkan, sehingga perlu diperhitungkan dalam hal penggunaan optimasi *query SQL* [26]. Penggunaan *query* tidak disarankan menggunakan *SELECT a.**. Ini dikarenakan semua kolom dan data akan diambil dan ditampilkan, sehingga beban *query* akan menjadi lebih besar dan lebih lambat. Untuk data yang banyak, perlu dipilih kolom mana saja yang akan digunakan dalam *query* data. Berikut ini adalah tabel perbandingan *SQL SELECT* pada tabel dengan *query* optimasi dan tanpa optimasi.

Tabel 1 sebelum optimasi menggambarkan *query SQL* dengan melakukan *SELECT* pada semua kolom. *Query* ini tidak cocok jika data yang ada di *database* jumlahnya banyak dan terus bertambah. Pada Tabel 1, setelah *query* dioptimalkan, dilakukan pemilihan kolom mana saja yang perlu ditampilkan. Pada tabel di atas dapat disimpulkan bahwa optimasi *query* dalam penulisan *SQL (Structured Query Language)*.

Proses pengujian *tuning* dilakukan dengan pengukuran waktu sebelum penambahan tabel indeks dan setelah *tuning* dilakukan serta sebelum dan sesudah optimasi *query* dengan memperhatikan *response time*. Hasil yang diinginkan dari penelitian ini yaitu suatu tabel yang sudah diindeks dan *query* yang sudah dioptimasi untuk meningkatkan performa *database*. Implementasi dari penelitian ini diharapkan mampu mengatasi *database* yang lambat dan mempercepat proses *query* agar sistem kuesioner akademik dapat berjalan dengan lancar.

Tabel 1. Perhitungan *Query*

Sebelum Optimasi	Setelah Optimasi
<i>SELECT</i> * FROM nama_tabel	<i>SELECT</i> nama_kolom, nama_kolom, .. FROM nama_tabel

3. HASIL DAN PEMBAHASAN

Pembuatan basis data dilakukan untuk menampung data-data kuesioner. Setelah basis data dibuat, langkah selanjutnya adalah menyiapkan file data dengan ekstensi .csv yang kemudian diimpor ke dalam tabel pada basis data. Perintah untuk membuat basis data ditunjukkan pada listing *SQL* pada Gambar 4.

```
create database kuesioner
```

Gambar 4 Perintah Create Database

Setelah diimpor ke dalam basis data, akan dihasilkan dua tabel, yaitu kue_responden_m_data sebagai master dan kue_responden_d_data sebagai detail. Tabel kue_responden_d_data memiliki jumlah record yang lebih banyak daripada tabel kue_responden_m_data, karena berisi data detail jawaban setiap responden. Sebelum proses pembuatan dan pengujian indeks dilakukan, terlebih dahulu disiapkan dan dibuatkan tabel yang akan dijadikan tabel uji pada penelitian ini. Nama tabel yang dibuat yaitu tabel kue_responden_m dan kue_responden_d. tabel kue_responden_m merupakan tabel induk responden sedangkan kue_responden_d merupakan tabel detail dari tabel transaksi. Kedua tabel ini saling berhubungan sehingga kedua tabel ini harus beriringan dalam sistem kuesioner akademik. Adapun *field* yang terdapat pada Tabel 2 dan 3.

Tabel 2. *Field* pada Tabel kue_responden_m

<i>Field</i>	Type	Size
ID	Int (key)	11
IDKUESIONER	Int	11
THNAKADEMIK	char	9
SEMESTER	Enum (‘Ganjil’, ‘GENAP’)	
KORESPONDEN	char	15
SMSTER	char	15
KONSENTRASI	char	15
REGULER	char	15
MATKUL	char	15
DOSEN	Char	15
KETERANGAN	Text	
IP	Char	20
TGL	timestamp	

Tabel 3. *Field* pada Tabel kue_responden_d

Nama <i>Field</i>	Type	Size
ID	Int (key)	11
THNAKADEMIK	char	9
SEMESTER	Enum (‘Ganjil’, ‘GENAP’)	
IDKUESIONER	int	11
IDRESPONDEN	int	11
SOAL	int	11
JAWABANCODE	int	11
MATKUL	char	15
DOSEN	Char	15
KETERANGAN	Text	

Nama <i>Field</i>	Type	Size
IP	Char	20
TGL	timestamp	

Rencana kolom-kolom pada tabel sudah disiapkan. Selanjutnya, membuat tabel di basis data dengan perintah menggunakan *query SQL*. Tabel 4 adalah *query SQL* untuk membuat tabel pada basis data kuesioner.

Tabel 4. Perintah Buat Tabel pada *Database*

Tabel kue_responden_m	Tabel kue_responden_d
<pre>CREATE TABLE kue_responden_m (`ID` int(11) NOT NULL, `IDKUESIONER` int(11) NOT NULL, `THNAKADEMIK` char(9) NOT NULL, `SEMESTER` enum('GANJIL','GENAP') NOT NULL, `KDRESPONDEN` char(15) NOT NULL, `SMSTER` char(15) DEFAULT NULL, `KONSENTRASI` char(15) DEFAULT NULL, `REGULER` char(15) DEFAULT NULL, `MATKUL` char(15) DEFAULT NULL, `DOSEN` char(15) DEFAULT NULL, `KETERANGAN` text, `IP` char(20) DEFAULT NULL, `TGL` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP)</pre>	<pre>CREATE TABLE kue_responden_d (`ID` int(11) NOT NULL , `THNAKADEMIK` char(9) DEFAULT NULL, `SEMESTER` enum('GANJIL','GENAP') DEFAULT NULL, `IDKUESIONER` int(11) DEFAULT NULL, `IDRESPONDEN` int(11) NOT NULL, `SOAL` int(11) NOT NULL, `JAWABANCODE` int(11) NOT NULL, `JAWABAN_VAL` char(5) DEFAULT NULL, `IP` char(20) DEFAULT NULL, `TGL` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP)</pre>

Setelah tabel-tabel yang dibutuhkan berhasil dibuat, langkah selanjutnya yaitu melakukan salinan data dari tabel kue_responden_m_data ke dalam tabel kue_responden_m dan dari tabel kue_responden_d_data ke dalam tabel_kue_res_responden_d.

3.1. Pembuatan Index

Setelah tabel yang dibutuhkan berhasil dibuat, selanjutnya dilakukan tahap penambahan indeks pada tabel kue_responden_m dan pada tabel kue_responden_d. Nama indeks pada tabel kue_responden_m adalah index_row1 dan index_row2, sedangkan pada tabel kue_responden_d diberikan indeks nama index_row3 dan index_row4. Setiap indeks memiliki urutan kolom masing-masing sesuai dengan kebutuhan sistem. Masing-masing indeks tersebut menggunakan tipe indeks b-tree. Indeks dibuat dengan menggunakan *query SQL*. Adapun bentuk *query SQL* yang penulis buat sebagai berikut.

1. Index ke 1 pada tabel kue_responden_m (Gambar 5).

```
create index index_row1 on kue_responden_m
(THNAKADEMIK, SEMESTER, KDRESPONDEN, SMSTER,
KONSENTRASI, REGULER, MATKUL, DOSEN) using BTREE
```

Gambar 5. Perintah Create Index index_row1

2. Index ke 2 pada tabel kue_responden_m (Gambar 6)

```
create index index_row2 on kue_responden_m
(THNAKADEMIK, SEMESTER, KDRESPONDEN) using BTREE
```

Gambar 6. Perintah Create Index index_row2

Kedua indeks pada tabel kue_responde_m akan memiliki perannya masing-masing, di mana index_row_1 akan membuat daftar isi indeks berupa *Field* mulai dari THNAKADEMIK, SEMESTER, KDRESPONDEN, SMSTER, KONSENTRASI, REGULER, MATKUL, DOSEN. Sedangkan pada indeks ke-2, yaitu index_row_2, akan membuat daftar isi indeks berupa *Field* mulai THNAKADEMIK, SEMESTER, KDRESPONDEN. Indeks-indeks tersebut akan berjalan sesuai dengan peta pencarian data pada *database* saat *query* dijalankan.

Selanjutnya, pembuatan indeks pada tabel kue_responden_d. Sama seperti halnya indeks pada tabel kue_responden_m, penulis akan menguji dengan dua indeks, yaitu index_row3 dan index_row4. Adapun bentuk *query SQL* yang dibuat adalah sebagai berikut.

3. Index ke 1 pada tabel kue_responden_d (Gambar 7)

```
create index index_row3 on kue_responden_d
(THNAKDEMIK, SEMESTER, IDKUESIONER, IDRESPONDEN, JAWABANCODE) using btree
```

Gambar 7. Perintah Create Index index_row3

4. Index ke 2 pada tabel kue_responden_d (Gambar 8)

```
create index index_row4 on kue_responden_d
(THNAKDEMIK, SEMESTER, IDRESPONDEN) using btree
```

Gambar 8. Perintah Create Index index_row4

Pada indeks kesatu, yaitu index_row_3, akan membuat daftar isi indeks dengan urutan *Field* THNAKDEMIK, SEMESTER, IDKUESIONER, IDRESPONDEN, JAWABANCODE. Sedangkan pada indeks kedua, yaitu index_row_4, akan membuat daftar isi dengan urutan berupa *Field* THNAKDEMIK, SEMESTER, IDRESPONDEN. Pada saat *query SQL* menjalankan perintah untuk mengambil data ke dalam *database* melalui fungsi *SELECT*, indeks akan bekerja dengan cara mencari daftar isi indeks yang dibuat dan mengurutkannya berdasarkan data yang memiliki prioritas, sehingga akan menambah efisiensi pengambilan data. *Query SQL* dengan klausa *WHERE* dan *GROUP BY* akan melakukan pencarian data berdasarkan indeks sehingga memungkinkan pengambilan data menjadi lebih cepat.

3.2. Optimasi Query

Optimasi *query* bertujuan untuk membuat jalur atau rute tercepat untuk pengambilan data yang mengarah pada *query SQL* yang diperlukan indeks. Indeks yang telah dibuat akan dipanggil melalui fungsi *SQL* dengan klausa *WHERE* atau *GROUP BY*. Untuk model *query* yang dibuat, penulis membuat usulan *query* yang digunakan berdasarkan *query* yang sudah dioptimalkan. Adapun *query* yang dioptimasi adalah sebagai berikut. *Query* yang tidak dirancang secara optimal dapat menyebabkan proses pemindaian tabel secara menyeluruh (full table scan), penggunaan sumber daya yang berlebihan, serta meningkatnya beban I/O dan CPU[26]. Kondisi ini diperparah apabila struktur *query* mengandung praktik yang kurang efisien, seperti penggunaan *SELECT ** secara tidak selektif, nested subquery yang kompleks, atau join tanpa dukungan indeks yang memadai. Dalam konteks Sistem Kuesioner Akademik, keterlambatan eksekusi *query* dapat berdampak pada lambatnya proses pelaporan dan analisis data. Hal ini berpotensi menghambat pengambilan keputusan akademik, terutama dalam evaluasi mutu pembelajaran dan perencanaan peningkatan kualitas dosen. Selain itu, pada periode pengisian kuesioner yang melibatkan akses simultan dari banyak mahasiswa, sistem berisiko mengalami penurunan performa atau bahkan kegagalan layanan apabila *query* tidak dioptimalkan[25]. Optimasi *query* dapat dilihat pada Gambar 9.

```
select a.THNAKDEMIK, a.SEMESTER, a.IDRESPONDEN, a.SEMESTER, a.KONSENTRASI,
a.REGULER, a.MATKUL, a.DOSEN, a.MATKUL, a.IDKUESIONER,
a.KONSENTRASI, sum(b.JAWABAN_VAL) as jawaban
from kue_responden_m a
left join kue_responden_d b on b.IDRESPONDEN = a.ID
where
a.TGL between date_add(date(now()), interval -2 year ) and curdate() and
a.THNAKDEMIK = '2021/2022' and
a.SEMESTER = 'Genap'
group by a.THNAKDEMIK, a.SEMESTER, a.IDRESPONDEN, a.SEMESTER, a.KONSENTRASI,
a.REGULER, a.MATKUL, a.DOSEN
```

Gambar 9. Optimasi Query

Query akan mengambil kolom tertentu untuk ditampilkan dan dijoin dengan tabel transaksi detail. Klausa *WHERE* dan *GROUP BY* digunakan untuk mengarahkan *query* ke indeks yang dibuat. Fungsi ini akan melakukan pencarian *Query SQL* melalui indeks. Untuk melihat apakah *query* tersebut mengarah pada indeks yang telah dibuat, penulis menganalisis dengan menggunakan *Execution Plan* dengan menggunakan *tools* dbeaver. Adapun hasil yang didapat dapat dilihat pada Gambar 10.

3.3 Response Time

Bagian ini membahas hasil eksperimen yang diperoleh dari implementasi metode *tuning database* menggunakan *indexing* dan optimasi *query*. Berdasarkan hasil pengujian, dapat diketahui bahwa metode yang

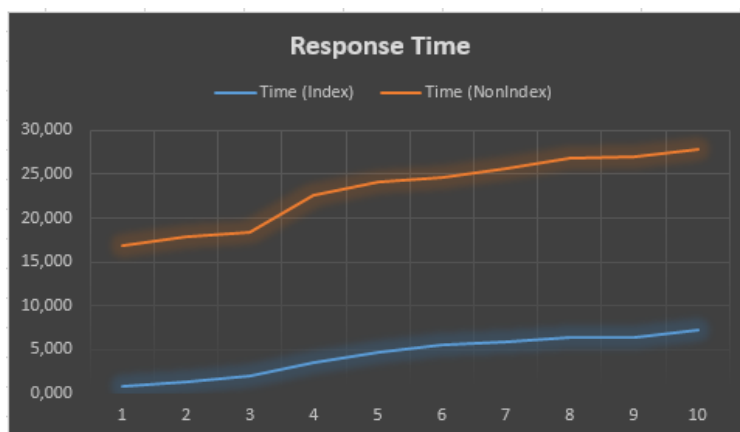
digunakan dalam penelitian ini mampu mencapai tujuan penelitian yang telah ditetapkan, yaitu meningkatkan performa *query* data pada sistem kuesioner akademik. Pengujian dilakukan menggunakan data sebanyak 3.000.000 record. *Query* dengan indeks dan *non-indeks* dijalankan sebanyak 10 kali percobaan, kemudian dihitung nilai rata-rata waktu eksekusinya. Hasil pengujian menunjukkan bahwa rata-rata waktu respons yang diperoleh adalah 4,34 detik untuk *query* dengan indeks. Sedangkan *query* tanpa indeks rata-rata waktu respons time yang dibutuhkan sekitar 23,23 detik (lihat Tabel 5). Hasil tersebut sejalan dengan tujuan penelitian ini, yaitu memaksimalkan *query* data pada *database* sistem kuesioner akademik agar berjalan optimal. Adapun waktu respons dari setiap percobaan ditampilkan pada tabel perbandingan dan grafik pada Gambar 11.

Possible Keys	Key	Name	Value
index_row1,index_row2	index_row1	ID	1
		Select Type	SIMPLE
		Table	a
		Type	SIMPLE
		Possible Keys	index_row1,index_row2
		Key	index_row1
		Key Length	37
		Ref	const,const
		Rows	1
		Filtered	100
		Extra	Using index condition; Using where; Using temporary; Using filesort

Gambar 10. *Excecution Plan*

Tabel 5. Perbandingan *Response Time*

No	Record	Time (Index)	Time (Non index)
1	300.000	0.832	16.900
2	500.000	1.262	17.965
3	1.000.000	1.981	18.400
4	1.500.000	3.470	22.647
5	2.000.000	4.670	24.209
6	2.400.000	5.539	24.670
7	2.600.000	5.882	25.611
8	2.800.000	6.290	26.906
9	2.900.000	6.310	27.101
10	3.000.000	7.200	27.849



Gambar 11. Grafik *Response Time*

Hasil pengujian pada penelitian ini menunjukkan bahwa rata-rata waktu eksekusi *query* tidak menggunakan index (non-index) adalah sebesar 23,23 detik, sedangkan rata-rata waktu eksekusi *query* dengan menggunakan index adalah sebesar 4,34 detik. Hasil rata-rata *query* dengan index didapat dari hasil 10 kali pengujian yaitu sebagai berikut.

$$\frac{0,832 + 1,262 + 1,981 + 3,470 + 4,670 + 5,539 + 5,882 + 6,290 + 6,310 + 7,200}{10} = 4,34s$$

Berdasarkan perhitungan tersebut, didapat nilai peningkatan performa sebesar:

$$\frac{23,23 - 4,34}{23,23} = 81,30\%$$

Dengan demikian, dapat disimpulkan bahwa penggunaan *indexing* dapat meningkatkan performa *query database* sebesar 81,30% dibandingkan dengan *query* tanpa menggunakan indeks. Kesimpulan ini didapat dari hasil tinjauan rata-rata waktu eksekusi.

3.4 Analisis Hasil Penelitian

Perbandingan hasil penelitian ini dengan penelitian sebelumnya yaitu kontribusi utama penelitian ini terletak pada penyusunan model implementatif *database tuning* yang telah teruji secara empiris pada sistem akademik berbasis data evaluasi. Selain itu, penelitian ini juga menghasilkan rekomendasi teknis yang bersifat aplikatif dan dapat diadopsi oleh institusi pendidikan dalam upaya meningkatkan kinerja sistem informasi akademik tanpa memerlukan perubahan infrastruktur yang signifikan. Dengan demikian, temuan penelitian ini diharapkan mampu mendukung pengelolaan data akademik yang lebih efisien, responsif, dan berkelanjutan.

4. KESIMPULAN

Berdasarkan hasil dari analisis dan pengujian, tabel dengan indeks dan optimasi kueri SQL akan meningkatkan response time kueri data dibandingkan dengan yang tidak menggunakan indeks. Sama seperti penelitian terdahulu, metode *Indexing* memiliki perbedaan response time *query* data yang cukup signifikan. Pada percobaan indeks pada *query SELECT* dengan klausa *WHERE* dan *GROUP BY* yang pertama dengan tabel yang non-indeks, kemungkinan terjadi perbedaan waktu respons yang signifikan karena fungsi *query* tidak mengarah ke indeks saat pengujian. Hasil pengujian menunjukkan peningkatan performa yang signifikan dengan rata-rata sekitar 81,30 s. Hasil kecepatan akses dapat dilihat dari rata-rata waktu respons tabel indeks dan nonindeks yang dihasilkan saat pencarian data.

REFERENSI

- [1] Z. Liu, J. Wang, H. Wu, Q. Ma, L. Peng, and Z. Tang, "Auto-tuning for HPC storage stack: an optimization perspective," *CCF Trans. High Perform. Comput.*, vol. 6, no. 6, pp. 608–631, 2024, doi: 10.1007/s42514-024-00198-8.
- [2] B. Milicevic and Z. Babovic, "A systematic review of deep learning applications in *database query* execution," *J. Big Data*, vol. 11, no. 1, pp. 1–38, 2024, doi: 10.1186/s40537-024-01025-1.
- [3] A. Anchlia, "Enhancing Query Performance Through Relational Database Indexing," *Int. J. Comput. Trends Technol.*, vol. 72, no. 8, pp. 130–133, 2024, doi: 10.14445/22312803/ijctt-v72i8p119.
- [4] M. Abbasi, M. V. Bernardo, P. Váz, J. Silva, and P. Martins, "Optimizing Database Performance in Complex Event Processing through Indexing Strategies," *Data*, vol. 9, no. 8, pp. 1–24, 2024, doi: 10.3390/data9080093.
- [5] Ellavarasan Asokan, "The Role of AI in Predictive Database Performance Tuning," *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, vol. 11, no. 2, pp. 356–369, 2025, doi: 10.32628/cseit25112365.
- [6] M. Mozaffari, A. Dignös, J. Gamper, and U. Störl, "Self-tuning Database Systems: A Systematic Literature Review of Automatic Database Schema Design and Tuning," *ACM Comput. Surv.*, vol. 56, no. 11, 2024, doi: 10.1145/3665323.
- [7] - Samidi and - Hariyanto, "Performance Tuning Oracle 11g Database Melalui Inisial Paramater, Structure Database dan SQL Tuning. Studi Pada ERP SISFORBUN Dana Pensiun Perkebunan (DAPENBUN)," *Techno.Com*, vol. 22, no. 2, pp. 400–408, 2023, doi: 10.33633/tc.v22i2.7831.
- [8] Y. Yan, J. Huang, H. Wang, J. Geng, K. Zhang, and T. Yu, "KnobCF: Uncertainty-Aware Knob Tuning," *IEEE Trans. Knowl. Data Eng.*, vol. 37, no. 12, pp. 7240–7254, 2025, doi: 10.1109/TKDE.2025.3608030.
- [9] S. R. Gaddam, "Database Performance Optimization: Strategies that Scale," *Int. J. Comput. Eng.*, vol. 7, no. 11, pp. 1–12, 2025, doi: 10.47941/ijce.2967.
- [10] N. Yulis, A. A. Ilham, A. Achmad, and F. A. Samman, "Enhancing Relational Database Efficiency through Algorithmic Query Tuning in Virtual Memory Systems," *Int. J. Informatics Vis.*, vol. 9, no. 4, pp. 1390–1400, 2025, doi: 10.62527/joiv.9.4.2850.
- [11] D. M. Gufron, M. Ramadhon, and S. Samidi, "Komparasi Database Performance Tuning Melalui Metode Object Relation Mapping pada SQL Server," *J. Pendidik. dan Teknol. Indones.*, vol. 4, no. 12, pp. 709–712, 2024, doi: 10.52436/1.jpti.537.
- [12] J. Lao et al., "GPTuner: A Manual-Reading Database tuning System via GPT-Guided Bayesian Optimization," *Proc. VLDB Endow.*, vol. 17, no. 8, pp. 1939–1952, 2024, doi: 10.14778/3659437.3659449.

- [13] Harminto Mulyo and Akhmad Khanif Zyen, “Pengaruh Hyperparameter *Tuning* Gradient Boosting Terhadap Prediksi Pemilihan Program Studi Mahasiswa Baru,” *Bull. Comput. Sci. Res.*, vol. 5, no. 2, pp. 131–137, 2025, doi: 10.47065/bulletincsr.v5i2.454.
- [14] Z. Yan, R. Xi, and M. Hou, *MCTuner: Spatial Decomposition-Enhanced Database tuning via LLM-Guided Exploration*, vol. 3, no. 6. arXiv, 2025. doi: 10.1145/3769807.
- [15] S. Lai, X. Wu, Y. Peng, and Z. Peng, “Survey on *Database Index Tuning* Techniques,” *Jisuanji Yanjiu yu Fazhan/Computer Res. Dev.*, vol. 61, no. 4, pp. 929–954, 2024, doi: 10.7544/issn1000-1239.202220931.
- [16] Irwan Muhammad and Nasution Padli, “Muhammad Irwan Padli Nasution Universitas Negeri Islam Sumatera Utara,” *J. Ilm. Nusantara (JINU)*, vol. 1, no. 4, pp. 705–710, 2024, [Online]. Available: <https://doi.org/10.61722/jinu.v1i4.18890>
- [17] D. Colley and C. Stanier, “Identifying New Directions in *Database Performance Tuning*,” *Procedia Comput. Sci.*, vol. 121, pp. 260–265, 2017, doi: 10.1016/j.procs.2017.11.036.
- [18] W. Wu and C. Wang, “Budget-aware *Query Tuning*: An AutoML Perspective,” *SIGMOD Rec.*, vol. 53, no. 3, pp. 20–26, 2024, doi: 10.1145/3703922.3703926.
- [19] Oluwafemi Oloruntoba, “AI-Driven autonomous *database* management: Self-tuning, predictive *query* optimization, and intelligent *indexing* in enterprise it environments,” *World J. Adv. Res. Rev.*, vol. 25, no. 2, pp. 1558–1580, 2025, doi: 10.30574/wjarr.2025.25.2.0534.
- [20] A. Azlansyah, “Optimalisasi Algoritma Naïve Bayes Dengan Hyperparameter *Tuning* Untuk Analisis Sentimen Ulasan Access By Kai,” vol. 9, no. 9, 2025, doi: 10.47065/josyc.v9i9.9999.
- [21] I. Trummer, “DB-BERT: making *database tuning* tools ‘read’ the manual,” *VLDB J.*, vol. 33, no. 4, pp. 1085–1104, 2024, doi: 10.1007/s00778-023-00831-y.
- [22] M. Raihan Siddik, M. Arief Hasan, A. Fajar Kesuma, N. Sari, S. Dwi Putri, and Q. Uyun Harahap, “Implementasi *Query Tuning* Untuk Peningkatan Performa Pada *Database* Barang Mini Market Nan,” *JATI (Jurnal Mhs. Tek. Inform.)*, vol. 9, no. 2, pp. 3183–3187, 2025, doi: 10.36040/jati.v9i2.13217.
- [23] M. Rama Putra, M. A. Hasan, V. Stefanus, A. Bhaskoro Maghribi, M. Irawan, and R. Menzona, “Pengaruh Indexed Views Terhadap Performa *Query* Pada Basis Data *SQL* Server Dengan Data Skala Besar,” *JATI (Jurnal Mhs. Tek. Inform.)*, vol. 9, no. 2, pp. 2586–2591, 2025, doi: 10.36040/jati.v9i2.13159.
- [24] V. Panwar, “AI-Driven *Query* Optimization: Revolutionizing *Database* Performance and Efficiency,” *Int. J. Comput. Trends Technol.*, vol. 72, no. 3, pp. 18–26, 2024, doi: 10.14445/22312803/ijctt-v72i3p103.
- [25] S. Patulus Rahmat Tambunan, M. Arief Hasan, R. Hartono Tambunan, M. Stephen Siahaan, M. Jordan Sirait, and F. Mahmud, “Implementasi Teknik *Query* Optimization Untuk Meningkatkan Performa *SQL* Server Pada Sistem Informasi Akademik,” *JATI (Jurnal Mhs. Tek. Inform.)*, vol. 9, no. 2, pp. 2437–2442, 2025, doi: 10.36040/jati.v9i2.13041.
- [26] A. Deska, I. Akbar, and - Samidi, “*Tuning Database* Pada Sistem Penerimaan Mahasiswa Baru Menggunakan Optimasi *Query* dan *Indexing*,” *Techno.Com*, vol. 22, no. 1, pp. 68–77, 2023, doi: 10.33633/tc.v22i1.7047.