PERBANDINGAN TEKNIK INDEXING BITMAP DAN B-TREE PADA ORACLE DATABASE

COMPARISON OF BITMAP AND B-TREE INDEXING TECHNIQUES IN ORACLE DATABASE

Siti Oktavia Eka Putri[1*], Nabila Athifah Zahra[1], Nur Racana Kuslaila[1], Siti Mukaromah[1]
*E-mail: sitioktavia25@gmail.com

[1]Sistem Informasi, Fakultas Ilmu Komputer, Universitas Pembangunan Nasional Veteran Jawa Timur

Abstrak

Relational Database Management System (RDBMS) berperan penting dalam manajemen dan pemeliharaan file data dalam skala besar serta berperan untuk mempercepat pengaksesan data yang dibutuhkan. RDBMS mendukung untuk memproses query pencarian dengan metode indexing. Ada beberapa teknik indexing yang dapat digunakan seperti bitmap, btree, hash, dan sekuensial yang memiliki kelebihan dan kekurangan masing-masing. Oleh karena itu, dalam artikel ini akan dibahas mengenai membandingkan teknik indexing bitmap dan b-tree untuk mengetahui teknik mana yang paling cepat sehingga dapat meningkatkan efektivitas dan efisiensi dalam administrasi basis data. Perbandingan dilakukan menggunakan lima jenis query dan tiga jenis kardinalitas data dari skema HR menggunakan Oracle *database* dengan mencatat *running time* selama query dieksekusi. Hasil dari pengujian menunjukkan bahwa teknik bitmap lebih cepat daripada teknik b-tree pada semua tabel yang memiliki tingkat *cardinality* rendah, normal, dan tinggi.
Kata Kunci : *bitmap, btree, index, oracle, basis data*

Abstract

The Relational Database Management System (RDBMS) plays a crucial role in managing and maintaining large-scale data files and serves to expedite the retrieval of required data. RDBMS supports processing search queries using indexing methods. Several indexing techniques are available, such as bitmap, B-tree, hash, and sequential, each with its advantages and disadvantages. Therefore, this article will compare bitmap and b-tree indexing techniques to find out which technique is the fastest so that it can increase effectiveness and efficiency in database administration. The comparisons were made using five types of queries and three types of data cardinality from the HR schema using the Oracle database by recording the running time during excecuting the queries. The test results show that the bitmap technique is faster than the b-tree technique for all tables that have low, standard, and high cardinality levels.
Kata Kunci : *bitmap, btree, index, oracle, database*

1. PENDAHULUAN

Database merupakan komponen penting dalam membangun suatu aplikasi. Database berisi kumpulan data yang menggambarkan aktivitas atau proses bisnis dalam suatu aplikasi. Misalnya database universitas akan berisi objek mengenai mahasiswa, mata kuliah, dosen, dan lain-lain [1]. Hampir semua pengelolaan database menggunakan Relational Database Management System (RDBMS). Umumnya database yang digunakan antara lain MySQL, Oracle, PostgreSQL, MariaDB, dan lain sebagainya. Penggunaan software RDBMS memiliki kelebihan dan kekurangan masing-masing, harus menyesuaikan dengan kebutuhan ukuran data dan performa

yang diinginkan. Database yang bagus adalah database yang memiliki *identifier* atau identitas yang unik atau dikenal dengan primary key. Primary key ini berfungsi agar data mudah dikenali dan mudah ketika dipanggil. Namun, pengelolaan database masih menjadi masalah besar khususnya ketika mengoptimalkan desain database karena sangat sulit untuk konsisten mengecek antara desain sistem dengan desain database untuk memenuhi kebutuhan pengguna [2].
Beberapa kriteria yang menunjukkan bahwa database itu baik adalah adanya konsistensi data, data nya tidak redundant, serta performa nya yang cepat. Semakin besar data yang dimiliki maka akan membutuhkan waktu yang besar pula. Oleh karena itu dibutuhkan pemrosesan query dan mengoptimalkan perintah untuk menemukan data yang secara efisien dan akurat dari database yang besar [3]. Salah satu contoh perintah SQL yang bisa digunakan untuk mengoptimasi pencarian adalah indexing. Ada beberapa teknik yang bisa digunakan untuk melakukan indexing antara lain bitmap, btree, sekuensial, hash, dan gabungan. Dalam artikel ini membahas tentang pengujian terhadap performa teknik bitmap dan btree dengan ukuran data dan query yang bervariasi. Dengan menguji kedua teknik tersebut dapat diketahui teknik mana yang memiliki performa pencarian paling cepat dan efektif.

2. METODOLOGI

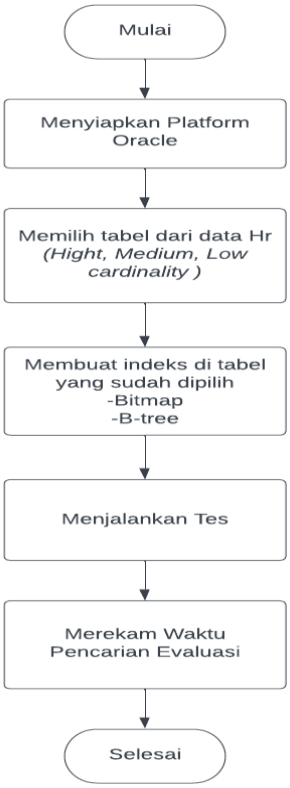
Berdasarkan jenisnya, metode yang digunakan yaitu metode *literature review* dan *experimental* untuk melakukan perbandingan ini. Dalam metode tersebut terdapat beberapa alur untuk menguji waktu pencarian data dan melakukan eksperimen terhadap teknik Bitmap dan teknik B-tree.

2.1 Literature Review

Artikel ini melakukan pengamatan terhadap pustaka terdahulu untuk memperoleh informasi yang dibutuhkan dalam menganalisis kinerja dari dua teknik tersebut. Metode ini disebut Literature Review yang merupakan cara untuk mengevaluasi dan menginterpretasikan berdasarkan pertanyaan khusus, area topik atau fenomena dan melengkapi kekurangan dari artikel terdahulu [4].

2.2 Experimental

Metode *Experimental* merupakan metode yang dilakukan untuk menguji kedua teknik tersebut. Pada tahap *eksperimental* ini melewati beberapa alur dalam melakukan analisis perbandingan teknik *Bitmap* dan *B-tree*.



Gambar 1. Diagram Alir Evaluasi Pengindeksan Bitmap dan B-tree

Alur dari metode ini yaitu menyiapkan platform oracle dan membuat database dengan mencheklist HR skema. Dari database *HR skema* kita memilih tiga tabel yang memiliki data dengan tingkat *cardinality* yang rendah, sedang, dan tinggi. Setelah memilih tabel dari data HR dilanjutkan melakukan indexing dengan dua teknik yang berbeda, yang pertama adalah membuat index bitmap, lalu yang kedua adalah index b-tree pada masing-masing tabel yang akan digunakan untuk pengujian. Kemudian kita mencoba teknik tersebut secara bergantian memakai berbagai macam query dari yang sederhana hingga yang kompleks untuk melakukan tes perbandingan kecepatan waktu antara teknik *bitmap* dengan teknik *b-tree*. Setelah kedua teknik tersebut diimplementasikan maka akan diketahui teknik mana waktu yang menunjukkan waktu pencarian paling cepat. Teknik yang menunjukkan waktu pencarian tercepat berarti teknik tersebut memiliki performa yang lebih unggul.

3. HASIL DAN PEMBAHASAN

3.1 Teknik Index Bitmap

Index bitmap dihasilkan dari turunan urutan nilai kunci yang menjelaskan jumlah nilai berbeda pada kolom. Dikutip dari penelitian terdahulu [5], setiap baris dalam index bitmap diberi nomor berurutan yang dimulai dengan bilangan bulat 0. Jika bit (satuan terkecil pada database) disetel ke '1', maka menunjukkan bahwa baris dengan RowId yang sesuai berisi nilai kunci. Jika tidak, bit akan disetel ke '0'. Index bitmap dibuat untuk meningkatkan efisiensi dan efektivitas kinerja berbagai jenis query, termasuk kueri yang melibatkan range, agregation, dan join.

RowId	C	B0	B1	B2	B3
0	2	0	0	1	0
1	1	0	1	0	0
2	3	0	0	0	1
3	0	1	0	0	0
4	3	0	0	0	1
5	1	0	1	0	0
6	0	1	0	0	0
7	0	1	0	0	0
8	2	0	0	1	0

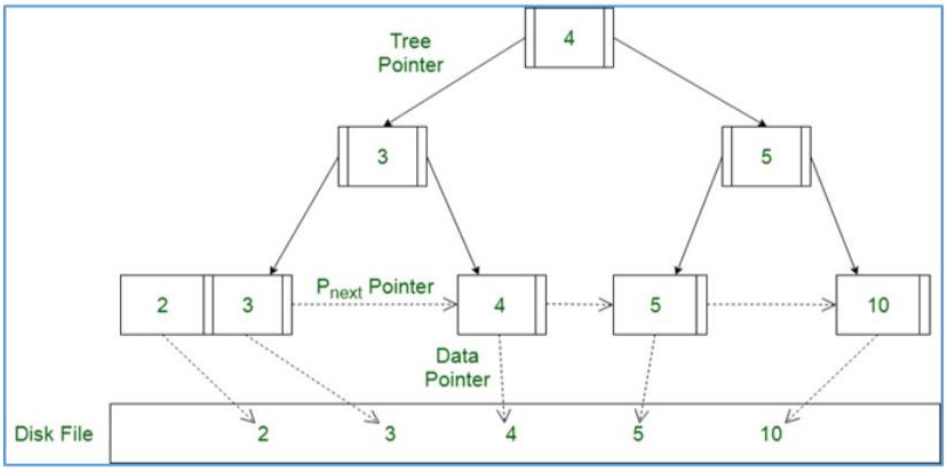
Gambar 1. Basic bitmap[5]

Row	Column	Bitmap index			
		=1	=2	=3	=4
1	1	1	0	0	0
2	2	0	1	0	0
3	4	0	0	0	1
4	3	0	0	1	0
5	2	0	1	0	0
6	4	0	0	0	1

Gambar 2. Contoh index bitmap yang terdiri dari 6 baris[6]

3.2 Teknik Index B-tree

B-tree adalah struktur pohon rekursif yang digunakan untuk menyimpan pointer index dan nilai dalam database. Index b-tree memungkinkan pengambilan data secara efisien dengan mengikuti petunjuk dari akar ke simpul daun, menjadikannya pilihan yang cocok untuk kolom dengan nilai kardinalitas tinggi, khususnya dalam desain pergudangan data [7]. Pencarian B-tree mengoptimalkan pengambilan data dengan membandingkan dan mengambil blok beberapa kunci indeks per langkah pencarian, yang bertujuan untuk meminimalkan pencarian hard disk dan mengurangi kesalahan halaman dalam sistem basis data memori tradisional dan utama [8].



Gambar 3. Algoritma B-tree [8]

3.3 Implementasi

Pada tahap implementasi ini dimaksudkan untuk menguji teknik indexing mana yang terbukti lebih membantu kinerja database terutama dalam pencarian.

3.3.1 Membuat Tabel dan Index pada Database

Langkah pertama adalah membuat tabel untuk digunakan sebagai objek pengujian. Dibutuhkan tiga tabel yang memiliki data dengan tingkat *cardinality* yang rendah, sedang, dan tinggi. Semakin tinggi tingkat *cardinality* sebuah tabel, maka datanya semakin unik dan bervariasi. Oleh karena itu, digunakanlah tingkat *cardinality* untuk acuan menguji efektivitas teknik indexing jika data yang berkaitan semakin kompleks [7].

Pada pengujian implementasi kali ini akan memanfaatkan basis data yang disediakan oleh Oracle. Untuk tabel dengan tingkat *cardinality* yang rendah, digunakan tabel *departments* di mana tabel tersebut memiliki entri data yang cukup sederhana. Tabel *locations* akan digunakan sebagai pengujian tabel tingkat *cardinality* sedang, di mana tabel tersebut berisi informasi tentang lokasi fisik dari departemen dalam organisasi. Untuk tabel dengan tingkat *cardinality* yang tinggi, akan digunakan tabel *employee* pada skema hr karena berisi informasi tentang karyawan dalam organisasi di mana setiap karyawan memiliki entri unik dalam tabel ini dengan banyak kolom yang berbeda, seperti employee_id, first_name, last_name, email, dan sebagainya.

Selanjutnya dilakukan indexing dengan dua teknik yang berbeda, yang pertama adalah membuat index bitmap, lalu yang kedua adalah index b-tree pada masing-masing tabel yang akan digunakan untuk pengujian.

```
SQL> create bitmap index idx_department on hr.departments(depart
ment_id, department_name);

Index created.
```

Gambar 4. Indexing bitmap pada tabel hr.departments (low cardinality)

```
SQL> create bitmap index idx_location on hr.locations(location_id, city)

Index created.

Elapsed: 00:00:00.03
```

Gambar 5. Indexing bitmap pada tabel hr.locations (normal cardinality)

```
SQL> create bitmap index idx_employee on hr.employees(employee_id,first_name,last_name);

Index created.

Elapsed: 00:00:00.02
```

Gambar 6. Indexing b-tree pada tabel hr.employees (high cardinality)

```
SQL> create index idx_department on hr.departments(department_id, department_name);

Index created.

Elapsed: 00:00:00.05
```

Gambar 7. Indexing b-tree pada tabel hr.departments (low cardinality)

```
SQL> create index idx_location on hr.locations(location_id, city);

Index created.

Elapsed: 00:00:00.03
```

Gambar 8. Indexing b-tree ada tabel hr.locations (normal cardinality)

```
SQL> create index idx_employee on hr.employees(employee_id, first_name, last_name);

Index created.

Elapsed: 00:00:00.07
```

Gambar 9. Indexing b-tree pada tabel hr.employees (high cardinality)

3.3.2 Query untuk Perbandingan

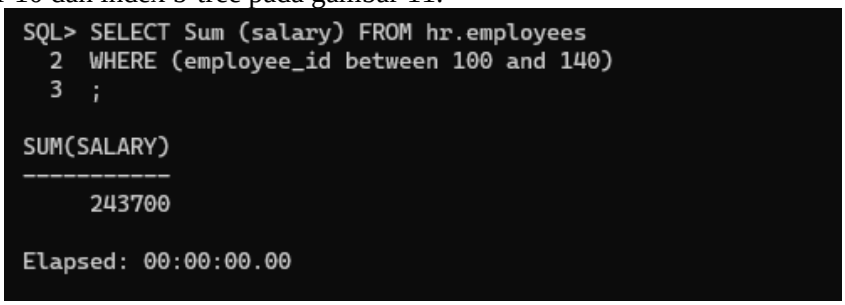
Diperlukan berbagai macam query dari yang sederhana hingga yang kompleks untuk diuji. Query pada Tabel 1 merupakan adopsi dari penelitian terdahulu [7] yang telah disesuaikan dengan kebutuhan perbandingan ini.

Tabel 1. Query untuk pengujian pada masing-masing tabel

	hr.departments	hr.locations	hr.employees
Query 1	SELECT * FROM hr.departments WHERE department_id = 70;	SELECT * FROM hr.locations WHERE location_id = 3000;	SELECT * FROM hr.employees WHERE employee_id = 100;
Query 2	SELECT count (*) FROM hr.departments;	SELECT count (*) FROM hr.locations;	SELECT count (*) FROM hr.employees;
Query 3	SELECT sum (manager_id) FROM hr.departments WHERE department_id > 100 and department_id < 200;	SELECT sum (location_id) FROM hr.locations WHERE location_id > 1000 and location_id < 1400;	SELECT sum (salary) FROM hr.employees WHERE employee_id > 100 and employee_id < 140;
Query 4	SELECT department_id, department_name, COUNT(*) FROM hr.departments GROUP BY department_id, department_name;	SELECT location_id, city, COUNT(*) FROM hr.locations GROUP BY location_id, city;	SELECT employee_id, first_name, COUNT(*) FROM hr.employees GROUP BY employee_id, first_name;
Query 5	SELECT Sum (manager_id) FROM hr.departments WHERE (department_id between 100 and 200);	SELECT sum (location_id) FROM hr.locations WHERE (location_id between 1000 and 1400);	SELECT Sum (salary) FROM hr.employees WHERE (employee_id between 100 and 140);

3.3.3 Uji Perbandingan

Uji perbandingan dilakukan pada masing-masing tabel dengan tingkat *cardinality* yang berbeda menggunakan lima macam query yang telah ditentukan. Pertama, satu per satu query akan dicoba pada masing-masing tabel dengan index bitmap. Saat output muncul dicatat juga *running time* yang diperlukan. *Running time* akan otomatis muncul saat *auto timing* diaktifkan dengan query SET TIMING ON. Setelah melakukan pencatatan pada tabel yang memiliki index bitmap, index tersebut didrop terlebih dahulu. Saat semua tabel sudah tidak memiliki index bitmap, dibuat index b-tree untuk pengujian selanjutnya. Pengujian index b-tree memiliki langkah, tabel, dan query yang sama dengan pengujian index bitmap. Contoh uji perbandingan index bitmap dapat dilihat pada gambar 10 dan index b-tree pada gambar 11.



Gambar 10. uji index bitmap query 5 pada tabel employee

```
SQL> SELECT Sum (salary) FROM hr.employees
  2  WHERE (employee_id between 100 and 140)
  3  ;

SUM(SALARY)
-----------
     243700

Elapsed: 00:00:00.05
```

Gambar 11. uji index b-tree query 5 pada tabel employee

3.4 Hasil Analisis

Hasil uji perbandingan dapat dilihat pada Tabel 2 berikut ini.

Table 2. Hasil perbandingan *running time*

	Low Cardinality (departments)		Normal Cardinality (locations)		High Cardinality (Employees)	
	Bitmap	**Btree**	**Bitmap**	**Btree**	**Bitmap**	**Btree**
Query 1	00:00:00:00	00:00:00:03	00:00:00:00	00:00:00:01	00:00:00:01	00:00:00:02
Query 2	00:00:00:01	00:00:00:01	00:00:00:00	00:00:00:00	00:00:00:00	00:00:00:00
Query 3	00:00:00:00	00:00:00:02	00:00:00:00	00:00:00:00	00:00:00:00	00:00:00:00
Query 4	00:00:00:01	00:00:00:02	00:00:00:00	00:00:00:01	00:00:00:02	00:00:00:05
Query 5	00:00:00:00	00:00:00:03	00:00:00:00	00:00:00:00	00:00:00:00	00:00:00:05

Dari uji perbandingan index bitmap dan index b-tree dengan menggunakan beberapa query yang berbeda pada tabel yang memiliki tingkat *cardinality* rendah, normal, dan tinggi dihasilkan *running time* yang berbeda dipengaruhi oleh kompleksitas query dan *cardinality* tabel. *Running time* yang diperlukan index bitmap untuk menjalankan query cenderung sangat rendah, yaitu kurang dari sama dengan 2 milidetik. Sedangkan pada index b-tree masih cenderung di antara 3 sampai 5 milidetik.

Pada query yang memiliki tingkat kompleksitas yang rendah hingga sedang seperti query 1, query 2, dan query 3, index bitmap memiliki rata-rata *running time* sekitar 0,22 milidetik. Pada tabel yang memiliki tingkat *cardinality* yang tinggi seperti table employees, cenderung membutuhkan waktu lebih banyak. Sedangkan index b-tree pada query 1, query 2, dan query 3 memiliki rata-rata 1,22 milidetik yang berarti lebih lama daripada index bitmap. Index b-tree membutuhkan waktu lebih lama pada tabel dengan *cardinality* yang rendah seperti tabel departments.

Query 4 dan query 5 memiliki kompleksitas lebih tinggi, *running time* yang dibutuhkan juga lebih banyak. Namun, dengan index bitmap, *running time* masih cenderung membutuhkan 0 milidetik. Untuk menjalankan query 4 dan query 5, index bitmap membutuhkan *running time* dengan rata-rata 0,5 milidetik. Sedangkan, index b-tree membutuhkan rata-rata *running time* 2,67 milidetik. Selisih *running time* terbesar antara index bitmap dan index b-tree terdapat pada query 5 yang memiliki kompleksitas tinggi dan *cardinality* tabel yang juga tinggi (tabel *employees*) dengan selisih 5 milidetik.

Terbukti index bitmap dapat mengeksekusi query lebih cepat bahkan dengan tingkat kompleksitas yang lebih tinggi dan dengan tabel yang memiliki *cardinality* yang tinggi. Hasil ini juga diperkuat oleh beberapa artikel dan penelitian terdahulu yang mengatakan bahwa index bitmap jauh lebih efisien [9] [10].

4. KESIMPULAN DAN SARAN

Berdasarkan perbandingan yang telah dieksperimen pada dua teknik yaitu bitmap dan b-tree dengan menggunakan platform oracle ini telah membahas tingkat kecepatan performa dari dua teknik tersebut yang dilakukan pada masing-masing tabel dengan tingkat *cardinality* yang berbeda. Melalui eksperimen atau uji perbandingan, dapat disimpulkan bahwa kecepatan performa dari teknik index Bitmap lebih cepat dan efisien dari pada menggunakan teknik index b-tree. Index bitmap juga dapat meningkatkan efektivitas kerja database pada query yang sederhana hingga yang kompleks serta pada tabel yang memiliki *cardinality* rendah hingga tinggi terbukti lebih cepat mengeksekusi query. Untuk topik artikel selanjutnya bisa menggunakan lebih banyak tabel dan query yang beragam lagi.

5. DAFTAR RUJUKAN

[1] Mostafa, S. A. (2020). A Case Study on B-Tree Database Indexing Technique. *Journal of Soft Computing and Data Mining*, *1*(1), 27-35

[2] Hashim, R. T. (2018). A Comparative Study of Indexing using Oracle and MS-SQL Server for Relational Database Management Systems. *International Journal of Computer Science and Mobile Computing*, *7*(12), 341-350.

[3] Sumantri, R. B. B., Subari, G., Mahardika, F., & Jayusman, H. (2023). PERBANDINGAN EFISIENSI WAKTU PROSES PENGAKSESAN DATA ANTARA QUERY BERBENTUK JOIN DENGAN SUBSELECT. *METHOMIKA: Jurnal Manajemen Informatika & Komputerisasi Akuntansi*, *7*(1), 25-33.

[4] Wahyudin, Y., & Rahayu, D. N. (2020). Analisis Metode Pengembangan Sistem Informasi Berbasis Website: A Literatur Review. *Jurnal Interkom: Jurnal Publikasi Ilmiah Bidang Teknologi Informasi Dan Komunikasi*, *15*(3), 119-133.

[5] Zaker, M., Phon-Amnuaisuk, S., & Haw, S. C. (2008). An adequate design for large data warehouse systems: Bitmap index versus b-tree index. International Journal of Computers and Communications, 2(2), 39-46.

[6] Chen, Z., Wen, Y., Cao, J., Zheng, W., Chang, J., Wu, Y., ... & Peng, G. (2015). A survey of bitmap index compression algorithms for big data. *Tsinghua Science and Technology*, *20*(1), 100-115.

[7] Morteza, Z., Somnuk, P. A., & Su-Cheng, H. (2008). Investigating Design Choices between Bitmap index and B-tree index for a Large Data Warehouse System.

[8] Mostafa, S. A. (2020). A Case Study on B-Tree Database Indexing Technique. *Journal of Soft Computing and Data Mining*, *1*(1), 27-35.

[9] Abdullah, W. A., Sahib, N. M., & Abass, J. M. (2019). Creation of Optimal Materialized Views Using Bitmap Index and Firefly Algorithm in Data Warehouse. *SCCS 2019 - 2019 2nd Scientific Conference of Computer Sciences*, 171–176. https://doi.org/10.1109/SCCS.2019.8852592

[10] Pibiri, G. E., & Kanda, S. (2021). Rank/select queries over mutable bitmaps. *Information Systems*, *99*, 101756. https://doi.org/10.1016/J.IS.2021.101756