IMPLEMENTASI *INVERTED INDEX* DENGAN SISTEM MANAJEMEN BASISDATA UNTUK MENDUKUNG MODEL PEMEROLEHAN *BOOLEAN*

JB Budi Darmawan
Jurusan Teknik Informatika, Fakultas Sains dan Teknologi, Universitas Sanata Dharma
E-mail : b.darmawan@staff.usd.ac.id, jbbudi@yahoo.com

ABSTRAK

Sistem pemerolehan informasi menawarkan kemampuan menyediakan informasi yang dibutuhkan pemakai. Kebanyakan sistem pemerolehan informasi dan mesin pencari web menggunakan inverted index yang terbukti sangat efisien untuk menjawab query. Implementasi sistem pemerolehan menggunakan sistem manajemen basisdata akan memperoleh kelebihan yang ditawarkan oleh RDBMS. Dalam paper ini peneliti mencoba melakukan penerapan inverted index ke dalam RDBMS untuk mendukung model pemerolehan boolean untuk operasi dasar AND, OR dan NOT. Operasi SQL didukung dengan operasi relational algebra dicoba diterapkan pada RDBMS untuk mendukung query seperti yang diharapkan saat menggunakan inverted index. Ujicoba dengan menggunakan corpus 5336 dokumen berita teknologi menghasilkan hampir 2 juta baris untuk penerapan inverted index ke dalam RDBMS. Implementasi operasi boolean dasar AND, OR atau NOT menunjukkan bahwa peningkatan jumlah operator boolean yang digunakan dari nol sampai enam membutuhkan waktu yang meningkat secara linier dengan tingkat korelasi di atas 0,99. Dengan spesifikasi sistem yang digunakan, untuk query dengan kata yang dimiliki sekitar 1 sampai 2 dokumen, waktu yang dibutuhkan untuk penggunaan satu operator sekitar 0,042 detik sampai sekitar 0,145 detik untuk enam operator. Sedangkan untuk query dengan kata yang dimiliki sekitar 5000 dokumen, waktu yang dibutuhkan untuk penggunaan satu operator sekitar 0,458 detik sampai sekitar 1,989 detik untuk enam operator. Salah satu alternatif penerapan inverted index ini dapat digunakan pada sistem yang sesuai dengan kebutuhan.

Kata Kunci : *Inverted index*, Sistem pemerolehan *boolean*, DBMS, RDBMS

1. PENDAHULUAN

Sistem pemerolehan informasi menawarkan kemampuan menyediakan informasi yang dibutuhkan pemakai. Kebanyakan sistem pemerolehan informasi dan mesin pencari web menggunakan *inverted index* yang terbukti sangat efisien untuk menjawab *query*[1]. Implementasi *inverted index* dapat diterapkan ke dalam DBMS dengan menawarkan beberapa kelebihan selain kekurangannya [2]. Beberapa kelebihan yang ditawarkan adalah:

a. Ruang lingkup indeks dapat diperluas. Perluasan skema indeks dengan perluasan tambahan kolom maupun relasi untuk melebarkan spektrum dari fungsional yang ditawarkan dapat dilakukan dengan mudah jika sebuah DBMS digunakan.
b. Sebuah DBMS menangani lapisan fisik sehingga tidak diperlukan pembuatan dan penggabungan indeks partial untuk mengkonstruksi indeks dari sebuah corpus yang besar.
c. Merubah maupun menhapus banyak dokumen adalah operasi yang mahal dalam sebuah *inverted index*. Dimana biayanya dalam O(n) dengan n adalah ukuran dari koleksi dalam word. Dalam DBMS, operasi ini dapat dilakukan lebih efisien.
d. Sistem pemerolehan informasi klasik memisahkan indeks untuk menjawab *query* dan indeks untuk memperbarui. Dengan DBMS perbedaan dan duplikasi ini tidak diperlukan, indeks tunggal dapat digunakan karena tidak harus membuat indeks *partial*.
e. Kemampuan DBMS yang dapat memanfaatkan sistem *multicore* dan *cluster* merupakan keuntungan yang transparan bagi sistem pemerolehan informasi yang menggunakannya. Seperti Oracle dapat memanfaatkan sistem *multicore* maupun *cluster* ini [3].

Disamping kelebihan diatas ada beberapa hal yang harus dipertimbangkan seperti:

a. *Inverted index* terdiri dari data dalam bentuk (t, occ) dimana t adalah *term* atau kata dan occ adalah *occurrence* atau dokumen dari t dalam *corpus*. Implementasi dalam sebuah relational DBMS akan menempati lebih banyak ruang penyimpan daripada sebuah *inverted index*. Sebagai contoh data $(t,\{d_1,d_3,d_5\})$, dalam sebuah relational DBMS akan direpresentasikan dalam tiga tuple $[t,d_1]$, $[t,d_3]$, $[t,d_5]$ yang berakibat pemborosan ruang penyimpan.

b. Sebagai bagian dari penggunaan ruang penyimpan yang lebih besar, waktu tanggapan *query* akan lebih tinggi untuk DBMS yang berdasarkan indeks, karena semakin banyak operasi I/O yang harus dilakukan.

Inverted index menawarkan akses *term-based* yang lebih efisien untuk melakukan perhitungan jawaban dari sebuah indeks. Namun beberapa tugas lain yang memerlukan akses *document-based* dapat dilakukan dengan lebih cepat dalam sistem DBMS [2].
Penggunaan *Relational Database Management System* (RDBMS) menuntut digunakannya bahasa *Structured Query Language* (SQL) untuk melakukan *Query*. Struktur data *inverted index* berupa pasangan *term* dan *posting list* ($t,\{d_1,d_3,d_5\}$) yang direpresentasikan dalam RDBMS akan menjadi tiga tuple $[t,d_1]$, $[t,d_3]$, $[t,d_5]$ menuntut digunakannya operasi SQL yang sesuai untuk memenuhi kebutuhan *query* [2].

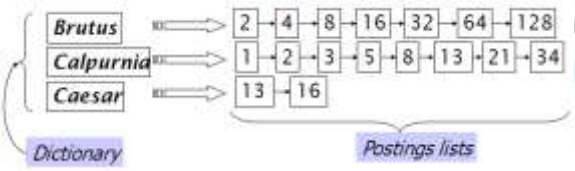
Model pemerolehan *boolean* (*boolean retrieval*) merupakan model yang menggunakan struktur data *inverted index*. Model ini merupakan model utama yang disediakan oleh penyedia informasi besar selama tiga dekade sampai awal 1990. Tetapi sistem ini tidak hanya menggunakan operasi *boolean* dasar (AND, OR, dan NOT) [4].

Penelitian ini bertujuan untuk mencoba mengimplementasikan dan mengamati unjuk kerja penggunaan konsep *relational algebra* untuk menjawab *query* dalam RDBMS seperti yang diharapkan saat menggunakan *inverted index* untuk model pemerolehan *boolean* dengan operasi *boolean* dasar (AND, OR dan NOT). Penelitian ini bermanfaat sebagai alternatif penerapan *inverted index* ke dalam RDBMS untuk memperoleh kelebihan yang ditawarkan oleh DBMS.

2. LANDASAN TEORI

2.1 Relational Algebra untuk *Inverted index*

Operasi model pemerolehan *boolean* dasar meliputi operasi AND, OR dan NOT. Untuk *inverted index* yang disajikan pada Gambar 1, dapat dilakukan operasi-operasi *boolean* dasar.



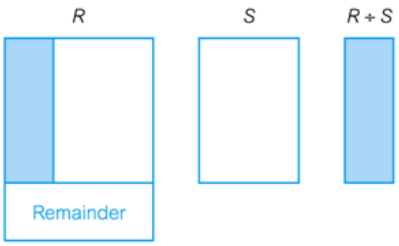
Gambar 1. Representasi *inverted index* [4]

Operasi AND dengan *n operand* akan melibatkan *n posting list*. Operasi Brutus AND Calpurnia dapat dilakukan dengan algoritma interseksi untuk kedua *posting list* Brutus dan Calpurnia seperti tersaji pada Gambar 2. Operasi ini menghasilkan dokumen 2 dan 8.

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

Gambar 2. Algoritma interseksi dari dua posting list *p*1 dan *p*2 [4]

Salah satu alternatif implementasi *operator* AND untuk representasi data dalam RDBMS dari *inverted index* seperti tersaji pada Gambar 1, dapat dilakukan dengan menggunakan *operasi division relational algebra* seperti tersaji pada Gambar 3.



Gambar 3. Representasi operasi *division relational algebra* [5]

Operasi *division* mendefinisikan suatu relasi melalui atribut C yang berisi sekumpulan *tuple* dari R yang cocok dengan kombinasi dari setiap *tuple* di S. Operasi ini disimbolkan dengan persamaan (1) [5],

$$R \div S \quad (1)$$

dimana C pada persamaan (2) adalah sekumpulan atribut dari R yang bukan atribut dari S.

$$C = A\text{-}B \quad (2)$$

Operasi *division* dapat diekpresikan menggunakan operasi dasar pada persamaan (3), (4) dan (5)

$$T_1 \leftarrow \Pi_C(R) \quad (3)$$
$$T_2 \leftarrow \Pi_C((S \times T_1) - R) \quad (4)$$
$$T \leftarrow T_1 - T_2 \quad (5)$$

Dimana T adalah hasil dari operasi *division*. Operasi division ini membutuhkan operasi SQL yang cukup komplek yang melibatkan operasi *projection*, *cartesian product* dan operasi minus.

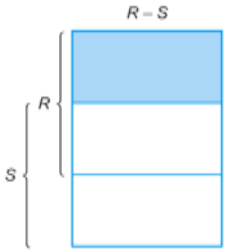
Alternatif lain untuk implementasi *operator* AND untuk representasi data dalam RDBMS dari *inverted index* seperti tersaji pada Gambar 1, dapat menggunakan operasi *intersection* [5] menggunakan operasi dasar persamaan (6). Operasi *intersection* lebih sederhana dibandingkan operasi *division*, operasi ini dapat diimplementasikan menggunakan *operator INTERSECT* pada operasi SQL.

$$R \cap S \quad (6)$$

Dalam implementasi *operator* OR untuk representasi data dalam RDBMS dari *inverted index* seperti tersaji pada Gambar 1, operasi *boolean* dasar OR untuk model pemerolehan *boolean* dapat langsung diterapkan pada predikat dari perintah SQL.

Implementasi *operator* NOT untuk representasi data dalam RDBMS seperti tersaji pada Gambar 1, dapat dilakukan dengan menggunakan operasi *set difference relational algebra* seperti tersaji pada Gambar 4. Operasi *set difference* mendefinisikan suatu relasi yang berisi tuple dalam relasi R tetapi tidak di S. Operasi ini disimbolkan dengan persamaan (7) [5]. Implementasi dari operasi ini dapat menggunakan *operator* MINUS dari SQL.

$$R - S \quad (7)$$



Gambar 4. Representasi operasi *set difference relational algebra* [5]

3. METODE PENELITIAN

Dalam penelitian ini dilakukan tahap-tahap sebagai berikut:

1. Studi pustaka penerapan konsep relational algebra untuk menjawab *query* dalam RDBMS seperti yang diharapkan saat menggunakan *inverted index* untuk model pemerolehan *boolean* dengan operasi *boolean* dasar.
2. Pengumpulan dokumen-dokumen sebagai *corpus* diambil dari berita-berita Kompas Tekno [6].

3. Implementasi penerapan konsep relational algebra yang telah dibahas dalam landasan teori menggunakan SQL RDBMS untuk mendukung *inverted index*.
4. Pengamatan unjuk kerja waktu *query* dan jumlah hasil *query* sebagai implementasi pada langkah 3 untuk operasi AND, OR dan NOT dilakukan pada tiga kelompok kata berdasarkan jumlah dokumen yang memenuhi suatu kata atau *document frequencies (df)*. Ketiga kelompok kata ini adalah kelompok kata yang mempunyai *df* 1 sampai 2, *df* kurang lebih 2500 dan *df* kurang lebih 5000. Operasi AND, OR dan NOT dilakukan dengan menggunakan 1 sampai 7 *operand* kata atau dengan kata lain menggunakan 0 sampai 6 *operator*.

Penelitian ini dilakukan dengan menggunakan sebuah komputer dengan spesifikasi sebagai berikut:

a. Perangkat lunak
   1. 6.1 [7]
   2. Sistem operasi, Microsoft Windows XP SP2
   3. Oracle 10G Release 2 [3]
   4. Oracle SQL Developer (2.1.1.64) [3]
   5. Java JDK 1.6.0 dan JDBC [3]
b. Netbeans Perangkat keras
   1. Prosesor Intel Core 2 Quad 6600
   2. Memori RAM 2 GB/5300 DDR2
   3. Hardisk 160 GB SATA 2
   4. Motherboard chipset Intel DP35DP

4. HASIL PENELITIAN DAN PEMBAHASAN

4.1 Data yang Digunakan

Struktur Tabel TECHNO yang digunakan pada percobaan ini disajikan pada Gambar 5. Tabel ini terdiri dari kolom WORD yang mewakili kata dan kolom DOCUMENT_ID yang mewakili dokumen berupa id. Kedua kolom ini digunakan sebagai primary key yang sekaligus akan membangkitkan indeks.

COLUMN_NAME	DATA_TYPE
WORD	VARCHAR2(20 BYTE)
DOCUMENT_ID	NUMBER(9,0)

Gambar 5. Struktur Tabel TECHNO menggunakan Oracle SQL Developer [3]

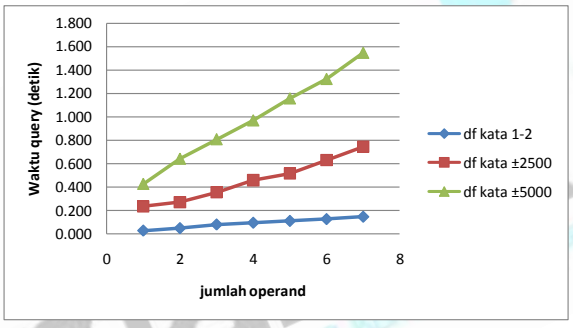
Dari 5336 dokumen Kompas Tekno[6], didapatkan 51262 kata yang berbeda yang menghasilkan 1903001 baris hasil representasi *inverted index* ke dalam RDBMS Tabel TECHNO. Panjang kata yang dihasilkan mempunyai jangkauan 2 sampai 19 huruf.

4.2 Hasil Percobaan

Untuk operasi AND dilakukan dengan operasi SQL *intersection* "SELECT DISTINCT document_id FROM techno WHERE word = 'term1' INTERSECT SELECT DISTINCT document_id FROM techno WHERE word = 'term2'". Pada operasi AND ini digunakan *operand* 'term1' dan 'term2'. Jumlah *operand* kata yang digunakan sampai sejumlah 7 *operand*. Hasil pengamatan waktu *query* yang diperoleh untuk operasi AND dengan jumlah *operand* satu sampai tujuh atau dengan kata lain jumlah *operator* boolean yang digunakan dari nol sampai enam untuk ketiga kelompok df disajikan pada Tabel 1. Representasi dalam bentuk grafik waktu *query* terhadap jumlah *operand* untuk *operator* AND untuk kelompok df kata 1-2, ±2500 dan ±5000 ini disajikan pada Gambar 6. Dari hasil pengamatan waktu *query* pada Tabel 1, penambahan jumlah *operand* untuk *operator* AND akan memperlama waktu akses secara linier dengan korelasi > 0.99 untuk ketiga kelompok *df* kata. Peningkatan jumlah df kata dari 1-2, ±2500 sampai ±5000 yang digunakan sebagai *operand* untuk *operator* AND juga memperlama waktu akses seperti terlihat pada Gambar 6.

Tabel 1: Waktu query operasi AND dengan 1 sampai 7 operand kata untuk ketiga kelompok df kata

df kata	Jumlah *operand* untuk operasi AND	Rata- rata Waktu query	Hasil query	Korelasi (r)
1-2	1	0,026	1	0,999028
	2	0,047	1	
	3	0,079	1	
	4	0,094	1	
	5	0,109	1	
	6	0,125	1	
	7	0,145	1	
±2500	1	0,235	2547	0,991882
	2	0,271	1096	
	3	0,354	631	
	4	0,458	379	
	5	0,515	188	
	6	0,631	181	
	7	0,745	181	
±5000	1	0,426	5336	0,993051
	2	0,641	5336	
	3	0,807	5336	
	4	0,969	5336	
	5	1,157	5336	
	6	1,323	5336	
	7	1,547	5336	



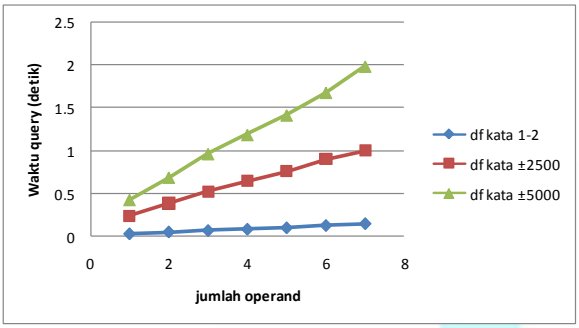
Gambar 6. Grafik waktu *query* terhadap jumlah *operand* untuk *operator* AND untuk kelompok df kata 1-2, ±2500 dan ±5000

Untuk operasi OR dilakukan dengan operasi SQL "SELECT DISTINCT document_id FROM techno WHERE word = 'term1' OR word = 'term2'". Pada operasi OR ini digunakan *operand* 'term1' dan 'term2'. Jumlah *operand* kata yang digunakan sampai sejumlah 7 *operand*. Hasil pengamatan waktu *query* yang diperoleh untuk operasi OR dengan jumlah *operand* satu sampai tujuh untuk ketiga kelompok *df* disajikan pada Tabel 2. Representasi dalam bentuk grafik waktu *query* terhadap jumlah *operand* untuk *operator* OR untuk kelompok df kata 1-2, ±2500 dan ±5000 ini disajikan pada Gambar 7. Dari hasil pengamatan waktu *query* pada Tabel 2, penambahan jumlah *operand* untuk *operator* OR akan memperlama waktu akses secara linier dengan korelasi > 0.99 untuk ketiga kelompok *df* kata. Peningkatan jumlah df kata dari 1-2, ±2500 sampai ±5000 yang digunakan sebagai *operand* untuk *operator* OR juga memperlama waktu akses seperti terlihat pada Gambar 7.

Tabel 2 : Waktu query operasi OR dengan 1 sampai 7 operand kata untuk ketiga kelompok df kata

df kata	Jumlah *operand* untuk operasi OR	Rata- rata Waktu query	Hasil query	Korelasi (r)
1-2	1	0,026	1	0,996328
	2	0,047	1	
	3	0,063	1	
	4	0,078	1	
	5	0,094	1	
	6	0,120	1	
	7	0,141	2	

±2500	1	0,235	2547	0,998844
	2	0,380	3691	
	3	0,521	4433	
	4	0,646	4772	
	5	0,760	5133	
	6	0,901	5133	
	7	1,000	5133	
±5000	1	0,426	5336	0,999157
	2	0,688	5336	
	3	0,963	5336	
	4	1,188	5336	
	5	1,417	5336	
	6	1,682	5336	
	7	1,989	5336	

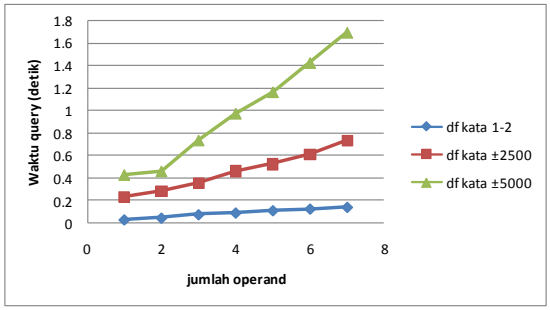


Gambar 7. Grafik waktu *query* terhadap jumlah *operand* untuk *operator* OR untuk kelompok df kata 1-2, ±2500 dan ±5000

Untuk operasi NOT dilakukan dengan operasi SQL *MINUS* "SELECT DISTINCT document_id FROM techno WHERE word = 'term1' MINUS (SELECT DISTINCT document_id FROM techno WHERE word = 'term2')". Pada operasi NOT ini digunakan *operand* 'term1' dan 'term2'. Jumlah *operand* kata yang digunakan sampai sejumlah 7 *operand*. Hasil pengamatan waktu *query* yang diperoleh untuk operasi NOT dengan jumlah *operand* satu sampai tujuh untuk ketiga kelompok *df* disajikan pada Tabel 3. Representasi dalam bentuk grafik waktu *query* terhadap jumlah *operand* untuk *operator* NOT untuk kelompok df kata 1-2, ±2500 dan ±5000 ini disajikan pada Gambar 8. Dari hasil pengamatan waktu *query* pada Tabel 3, penambahan jumlah *operand* untuk *operator* NOT akan memperlama waktu akses secara linier dengan korelasi > 0.99 untuk ketiga kelompok *df* kata. Peningkatan jumlah df kata dari 1-2, ±2500 sampai ±5000 yang digunakan sebagai *operand* untuk *operator* NOT juga memperlama waktu akses seperti terlihat pada Gambar 8.

Tabel 3: Waktu query operasi NOT dengan 1 sampai 7 operand kata untuk ketiga kelompok df kata

df kata	Jumlah *operand* untuk operasi NOT	Rata- rata Waktu query	Hasil query	Korelasi (r)
1-2	1	0,026	1	0,994935
	2	0,042	0	
	3	0,073	0	
	4	0,089	0	
	5	0,110	0	
	6	0,125	0	
	7	0,141	0	
±2500	1	0,235	2547	0,994834
	2	0,287	1451	
	3	0,359	849	
	4	0,463	479	
	5	0,526	297	
	6	0,614	297	
	7	0,734	297	
±5000	1	0,426	5336	0,991704
	2	0,458	0	
	3	0,735	0	
	4	0,974	0	
	5	1,166	0	
	6	1,427	0	
	7	1,698	0	



Gambar 8. Grafik waktu *query* terhadap jumlah *operand* untuk *operator* NOT untuk kelompok df kata 1-2, ±2500 dan ±5000

Pada Gambar 8. terlihat bahwa untuk kelompok df kata ±5000 saat terjadi peningkatan jumlah *operand* dari satu ke dua *operand* terjadi peningkatan waktu *query* yang lebih kecil dibandingkan dengan peningkatan waktu *query* saat terjadi peningkatan jumlah *operand* dari dua sampai tujuh *operand*. Hal ini disebabkan jumlah hasil *query* berpengaruh pada waktu akses, seperti terlihat dari Tabel 3 kelompok df kata ±5000 terjadi penurunan hasil *query* yang tajam saat operasi dengan jumlah *operand* satu yang menghasilkan jumlah hasil *query* 5336 menjadi jumlah hasil *query* 0 untuk operasi dengan jumlah *operand* dua. Sedangkan untuk operasi dengan jumlah *operand* dua sampai tujuh dengan jumlah hasil *query* yang sama yakni 0, menghasilkan peningkatan waktu *query* yang relatif sama.

Dengan spesifikasi sistem yang digunakan, untuk *query* dengan kata yang dimiliki sekitar 1 sampai 2 dokumen (*df kata*), waktu yang dibutuhkan untuk penggunaan satu *operator* sekitar 0,042 detik (Tabel 1) sampai sekitar 0,145 detik untuk enam *operator* (Tabel 1). Sedangkan untuk *query* dengan kata yang dimiliki sekitar 5000 dokumen (*df kata*), waktu yang dibutuhkan untuk penggunaan satu *operator* sekitar 0,458 detik (Tabel 3) sampai sekitar 1,989 detik (Tabel 2) untuk enam *operator*.

5. PENUTUP

5.1 Simpulan

Penerapan *inverted index* ke dalam RDBMS dengan kelebihan yang ditawarkannya untuk mendukung model pemerolehan boolean dengan operasi dasar AND, OR dan NOT menunjukkan hasil di bawah ini.
Peningkatan jumlah *operator* boolean yang digunakan dari nol sampai enam *operator* membutuhkan waktu yang meningkat secara linier dengan tingkat korelasi di atas 0,99.
Menggunakan spesifikasi sistem yang digunakan dengan ujicoba menggunakan corpus 5336 dokumen berita teknologi yang menghasilkan jumlah baris hampir 2 juta untuk penerapan *inverted index* ke dalam RDBMS, untuk *query* dengan kata yang dimiliki sekitar 1 sampai 2 dokumen, waktu yang dibutuhkan untuk penggunaan satu *operator* sekitar 0,042 detik sampai sekitar 0,145 detik untuk enam *operator*. Sedangkan untuk *query* dengan kata yang dimiliki sekitar 5000 dokumen, waktu yang dibutuhkan untuk penggunaan satu *operator* sekitar 0,458 detik sampai sekitar 1,989 detik untuk enam *operator*.
Penerapan *inverted index* ke dalam RDBMS menjadi salah satu alternatif penerapan *inverted index* yang dapat digunakan pada sistem yang sesuai dengan kebutuhan untuk mendapatkan kelebihan yang ditawarkan oleh RDBMS.

5.2 Penelitian selanjutnya

Penelitan selanjutnya dapat dilakukan untuk penerapan *inverted index* menggunakan teknologi DBMS yang lebih baik dari RDBMS seperti ORDBMS atau OODBMS.

DAFTAR PUSTAKA

[1]. Ricardo Baeza-Yates and Berthier Ribeiro-Neto, 1999, Modern Information *Retrieval*, Addison Wesley.
[2]. Papadakos, et. all, 2008, Mitos: Design and Evaluation of a DBMS-based Web Search Engine. IEEE.
[3]. Oracle, http://www.oracle.com, 2010. diakses terakhir 27 Desember 2010.
[4]. Christopher D. Manning, Prabhakar Raghavan and Hinrich Schütze, 2008, *Introduction to Information Retrieval*, Cambridge University Press.
[5]. Thomas Connoly & Carolyn Begg, Database Systems : A Practical Approach to Design, Implementation, and Management, 4th edition, 2005, Pearson Education Limited, England.
[6]. Kompas Tekno, http://tekno.kompas.com, 2010. diakses terakhir 27 Desember 2010.
[7]. Netbeans, http://netbeans.org, 2010. diakses terakhir 27 Desember 2010.