Penerapan Fungsi Hash dalam Pengindeksan Data Terenkripsi pada ORM Hibernate

Nugroho Satriyanto - 13514038[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]nugroho.s@outlook.co.id

***Abstract*—Basis data adalah salah satu penyimpanan persisten yang paling sering digunakan. Operasi basis data haruslah efisien sehingga operasi-operasi yang digunakan dalam API atau klien aplikasi dapat berjalan dengan efisien juga. Beberapa metode yang dapat menyelesaikan permasalahan ini adalah penggunaan hash dan penggunaan pohon biner. Untuk mengoptimasi permasalahan pencarian dapat digunakan dengan penggunaan hash sementara untuk menyelesaikan permasalahan penyortiran dapat menggunakan hash terurut atau pohon biner.**

***Keywords*—indeks, hash, basis data, query**

I. PENDAHULUAN

Penyimpanan data yang persisten adalah salah satu yang perlu dipikirkan pada saat pembuatan aplikasi. Banyak metode penyimpanan yang dapat digunakan pada suatu aplikasi seperti file sistem, spreadsheet, atau basis data yang merupakan metode yang paling sering digunakan.

Penggunaan basis data sebagai metode penyimpanan menghasilkan masalah baru dikarenakan perusahaan sering kali memerlukan adanya administrator basis data (DBA) untuk mengurus basis data dikarenakan basis data yang ukurannya besar dan menjadi tidak mungkin diurus oleh programmer.

Salah satu ancaman dari adanya administrator basis data ini adalah administrator basis data umumnya memiliki hak penuh atas basis data. Hal ini menunjukkan bahwa administrator basis data memiliki hak untuk melihat seluruh isi basis data. Selain itu, tipe ancaman ini termasuk salah satu ancaman paling umum dan masuk dalam urutan keenam threats and risks pada keamanan basis data. [1]

Permasalahan yang timbul karena hak administrator basis data adalah ketika ada data yang tidak boleh dilihat oleh administrator basis data tetapi perlu bisa didekripsi untuk menyelesaikan masalah yang mungkin timbul, misalnya alamat email, nomor telepon, saldo, atau riwayat transaksi pengguna. Informasi ini memiliki nilai yang dapat mengundang orang berniat jahat dan dapat merusak reputasi perusahaan jika informasi ini bocor. Informasi ini perlu disimpan sehingga dapat ditampilkan pada aplikasi tetapi akses terhadap informasi ini harus dibatasi sehingga kemungkinan kebocoran dapat dikurangi.

Salah satu cara mengatasi permasalahan ini adalah dengan mengenkripsi basis data. Hal ini mengakibatkan seluruh data terenkripsi dan administrator basis data tetap dapat menjalankan tugasnya tanpa dapat mengetahui isi atau makna dari data itu. Permasalahan dari mengenkripsi seluruh data adalah performansi yang akan menurun dikarenakan indeks pada basis data menjadi tidak dapat digunakan untuk mempercepat pemrosesan query.

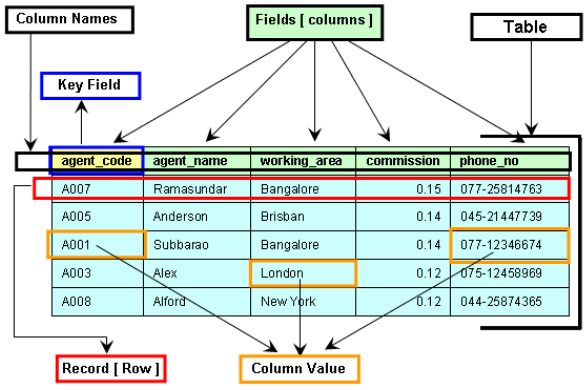
Metode lain yaitu mengenkripsi hanya sebagian data dapat menyelesaikan sebagian besar masalah ini. Namun, terkadang data sensitif tersebut diperlukan untuk menjadi indeks seperti alamat email yang akan mempercepat query untuk pengecekan apakah email tersebut telah terdaftar. Oleh karena itu, diperlukan metode untuk melakukan pengindeksan terhadap data terenkripsi.

II. DASAR TEORI

A. Basis Data

Basis data, sering disingkat DB, adalah koleksi dari sekumpulan informasi yang dapat dengan cepat menseleksi sebagian dari data, menambahkan, menghapus, atau memodifikasi yang diminta. [2] Umumnya basis data dapat dikelompokkan dalam beberapa kategori seperti basis data relasional, basis data berorientasi objek, basis data graf, basis data terdistribusi, dan lain sebagainya.

Dalam basis data relasional, tempat penyimpanan dapat dibagi menjadi tabel yang dapat dibagi menjadi kolom dan baris. Baris pada tabel merepresentasikan satu kesatuan data yang disimpan, sementara kolom pada tabel merepresentasikan kelompok jenis data yang disimpan pada tabel.

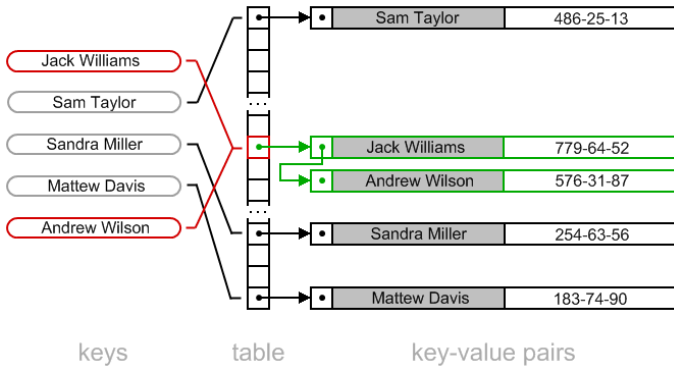


Gambar 1 komponen dari basis data (sumber: https://www.w3resource.com/sql/sql-basic/the-components-of-a-table.php)

Selain tabel dan komponennya, komponen lain basis data adalah indeks. Index merupakan objek struktur data tersendiri yang tidak bergantung kepada struktur tabel. Setiap index terdiri dari nilai kolom dan penunjuk (atau ROWID) ke baris yang berisi nilai tersebut. Penunjuk tersebut secara langsung menunjuk ke baris yang tepat pada tabel, sehingga menghindari terjadinya full table-scan. Akan tetapi lebih banyak index pada tabel tidak berarti akan mempercepat query. Semakin banyak index pada suatu tabel menyebabkan kelambatan pemrosesan perintah-perintah DML (Data Manipulation Language), karena setiap terjadi perubahan data maka index juga harus disesuaikan. [3]

Penggunaan indeks dapat menjadi suatu cara untuk mempercepat query seperti query select, order by, atau join. Oleh karena itu, walaupun indeks akan menggunakan memori lebih, tetapi jika ditempatkan pada kolom yang sering mengalami operasi seperti select, order by, atau join akan sangat membantu.

Indeks yang umum terdapat pada basis data adalah indeks hash table dan indeks pohon biner. Indeks hash table memanfaatkan fungsi hash untuk memetakan suatu nilai pada suatu alamat pada array. Suatu alamat pada array atau table yang disebut bucket yang berisi beberapa pointer yang menunjuk alamat record yang berkesesuaian.



Gambar 2 hash table pada basis data (sumber: http://www.algolist.net/Data_structures/Hash_table/Chaining)

Adapun untuk indeks jenis pohon biner memanfaatkan keterurutan dari suatu data. Hal ini menyebabkan penggunaan pohon biner dapat melakukan pencarian dengan binary search dengan mudah. Penggunaan pohon biner umum digunakan juga untuk mempercepat query order by dikarenakan sifat keterurutannya.

B. Fungsi Hash

Fungsi hash adalah semua jenis fungsi yang bisa memetakan segala ukuran data ke dalam data berukuran tetap. [4] Dikarenakan ukuran fungsi hash yang tetap, pemetaan fungsi hash menjadi banyak ke satu dan hal ini berarti tidak bisanya mengembalikan nilai hash ke nilai awal.

Tidak semua fungsi hash dapat aman digunakan sebagai algoritma kriptografi. Agar suatu fungsi hash bisa dinyatakan aman secara kriptografi, fungsi hash disyaratkan memenuhi kriteria sebagai berikut.

- Sulit untuk membalik fungsi hash dari suatu nilai hash ke suatu nilai awal (pre-image resistance)
- Sulit untuk menemukan masukan lainnya yang berbeda namun menghasilkan nilai hash yang sama (second pre-image resistance)
- Sulit menemukan masukan lainnya dengan ukuran yang tidak dibatasi yang berbeda namun menghasilkan nilai hash yang sama (collision resistance)

Fungsi hash telah banyak mengalami perkembangan dan sampai saat ini banyak juga yang dinyatakan aman secara kriptografi. Beberapa diantara hash yang aman secara kriptografi adalah MD5, SHA3, RipeMD, Whirpool, dan masih banyak lainnya.

Fungsi hash memiliki banyak kegunaan seperti pengecekan validitas file, autentikasi email, pengindeksan basis data, atau pemetaan objek seperti hashmap.

C. ORM Hibernate

ORM (Object-Relational Mapper) adalah suatu metode/teknik pemrograman yang digunakan untuk mengkonversi data dari lingkungan bahasa pemrograman berorientasi objek (OOP) dengan lingkungan database relasional. [5]

ORM tersedia dalam banyak bahasa berorientasi objek. Bahasa java, python, php, dan lainnya memiliki framework ORM yang disediakan pihak ketiga. Selain itu ORM juga biasanya terdapat pada framework OOP seperti eloquent pada laravel.

Hibernate adalah contoh ORM yang cukup banyak digunakan pada lingkungan pengembangan java. Selain itu, framework hibernate juga tersedia dalam framework spring yang merupakan framework web java.

ORM Hibernate dapat secara otomatis memetakan suatu objek pada java ke dalam suatu tabel pada basis data relasional. Pemetaan otomatis ini dimungkinkan dengan pemberian anotasi khusus pada kelas dan properti-properti. Dengan pemberian anotasi ini hibernate mengetahui hal-hal seperti di mana data tersebut harus disimpan dan apa saja yang harus dilakukan sebelum atau sesudah operasi dilakukan.

Selain itu, hibernate juga memiliki fleksibilitas yang cukup dikarenakan adanya anotasi @ColumnTransformer yang berfungsi untuk mengubah cara serialisasi dan deserialisasi tipe data pada objek menjadi suatu record pada basis data.

III. USULAN SOLUSI

Pada kasus basis data, operasi yang umum digunakan pada query select adalah operasi perbandingan yaitu pencarian nilai dan pengurutan nilai. Oleh karena itu, kolom yang dinilai akan sering menjadi basis pada kedua operasi tersebut perlu pengindeksan untuk mempercepat query select.

A. Fungsi Hash untuk Pengindeksan Database

Untuk mengoptimasi pencarian nilai yang dilakukan pada query select dapat dilakukan dengan penggunaan indeks yang kadang tidak didukung ketika kolom tersebut terenkripsi pada framework ORM.

Salah satu cara penyelesaian masalah ini adalah menuliskan algoritma hash pada anotasi @ColumnTransformer pada hibernate sehingga proses penulisan atau pembacaan data dapat

dimodifikasi. Proses yang dilakukan pada @ColumnTransformer meliputi proses enkripsi untuk mengubah nilai kolom tersebut dan proses menghitung nilai hash kolom tersebut untuk disimpan pada memori.

Cara penyelesaian masalah lainnya adalah membuat tabel atau kolom baru yang dapat meniru fungsi indeks. Tabel baru tersebut dapat digunakan untuk meniru algoritma indeks hashtable, dengan dua kolom yaitu kolom yang digunakan sebagai basis pencarian dan kolom yang digunakan sebagai foreign key menunjukkan lokasi data terenkripsi tersebut.

Gambar 3 relasi tabel terenkripsi dengan hashtable

Dikarenakan ORM hibernate tidak memungkinkan kolom yang dianotasi dengan anotasi enkripsi untuk diindeks, pembuatan hashtable ini akan dapat melakukan pencarian menggunakan indeks dengan kolom telepon_hash. Dari baris yang didapatkan pada pencarian tersebut, kolom telepon_fk akan menjadi penunjuk baris yang dimaksud pada tabel user.

Penggunaan algoritma hash yang bisa digunakan pada hashtable tidak dibatasi, oleh karena itu dapat dipilih algoritma hash yang telah ada. Selain itu metode ini cukup sederhana dan juga memungkinkan penggunaan algoritma enkripsi yang telah ada.

Sayangnya, penggunaan algoritma hash biasa hanya dapat menyelesaikan permasalah pencarian. Hal ini dikarenakan hasil hash tidak menghasilkan keterurutan yang sama dengan data awal, sehingga bisa terjadi hash(x) > hash(y) untuk x < y. Dikarenakan ketidaksesuaian perbandingan inilah, algoritma hash biasa tidak dapat menyelesaikan permasalahan sorting.

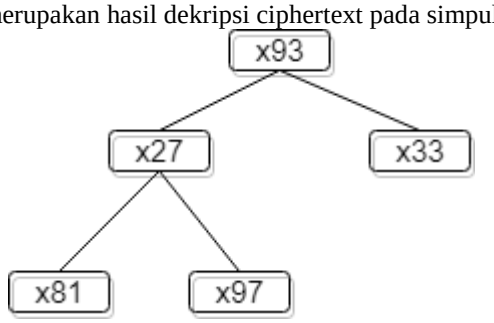
B. Pemetaan Keterurutan Ciphertext pada Kolom Sortable

Permasalahan utama kasus enkripsi pada kolom yang mengalami proses pengurutan adalah proses enkripsi dan hashtable yang tidak dapat menjaga hubungan keterurutannya. Oleh karena itu, proses enkripsi dan hash yang dilakukan tidak bisa dilakukan dengan cara konvensional dalam kriptografi.

Kriteria untuk melakukan enkripsi dan hash pada hasil enkripsi dan hash yang berarti untuk x > y, encrypt(x) > encrypt(y) dan hash(x) > hash(y) dan sebaliknya.

Algoritma enkripsi yang dapat mempertahankan keterurutan tentu akan sangat sulit dilakukan, hal ini dikarenakan untuk mengenkripsi suatu nilai harus memperhitungkan nilai seluruh data yang sebelumnya telah dienkripsi. Salah satu cara untuk mengatasi permasalahan ini adalah memetakan keterurutan ciphertext, sehingga dari suatu ciphertext dapat diketahui urutannya dan memungkinkan perbandingan pada ciphertext. Salah satu cara memetakan keterurutan adalah pohon biner yang dapat merepresentasikan keterurutan pada sebuah data relatif terhadap akarnya. Enkripsi dilakukan dengan memilih nilai akar untuk membentuk pohon yang akan dapat juga berlaku sebagai kunci untuk enkripsi dan dekripsinya.

Dalam memetakan keterurutan pada pohon biner, diperlukan suatu struktur data pohon biner yang setiap simpulnya adalah ciphertext yang disusun sesuai dengan keterurutan plaintext. Ciphertext akan disisipkan ke kanan pada pohon biner apabila plaintext yang berkesuaian lebih besar dari plaintext yang merupakan hasil dekripsi ciphertext pada simpul sebelumnya.



Gambar 4 pohon biner yang terbentuk dari plaintext [54,31,11,84,43] yang dienkripsi menjadi [93,27,81,33,97]

Dari pohon biner tersebut, nilai paling kecil akan ditunjukkan pada simpul terkiri yaitu ciphertext 81 yang plaintextnya adalah 11.

Dikarenakan pohon biner disusun berdasarkan keterurutan plaintext, keterurutan ciphertext yang terbentuk juga dapat diketahui dengan menelusuri pohon binernya.

Dengan adanya pohon biner, maka keterurutan ciphertext dapat ditelusuri dan menjadikan proses pengurutan pada basis data dapat diselesaikan.

C. Order Preserving Hash pada Kolom Sortable

Adapun untuk penyelesaian permasalahan keterurutan nilai hash juga dapat dilakukan hal yang sama, yaitu dengan membuat pohon biner yang diurutkan berdasarkan nilai plaintext yang berkesesuaian. Selain itu, pohon biner juga bisa disusun berdasarkan keterurutan ciphertext dikarenakan keterurutan ciphertext juga telah bisa ditelusuri dengan pohon biner untuk proses enkripsi.

Alternatif kedua dalam menyelesaikan masalah ini adalah memberikan nomor urutan pada tempat paling signifikan pada hash yang dihasilkan. Dikarenakan nilai hash yang ukurannya tetap, penambahan urutan yang ukurannya tetap tidak akan banyak berpengaruh. Selain itu, cara ini cukup mudah untuk diimplementasikan dibandingkan menyusun pohon biner.

Permasalahan dari penggunaan pohon biner dan penambahan nomor urutan adalah penyusunannya yang lama dibandingkan proses hash itu sendiri dan tidak efektif untuk data berukuran besar. Oleh karena itu, fungsi hash yang bisa menghasilkan keterurutan tanpa memperhitungkan nilai yang di hash sebelumnya akan sangat membantu mempercepat proses pengurutan.

Order Preserving Hash Function (OPHF) adalah jenis hash yang dapat menjaga keterurutan hasilnya. Beberapa algoritma hash yang termasuk jenis ini adalah CHM, BMZ, dan pearson.

IV. IMPLEMENTASI SOLUSI

Dalam melakukan implementasi solusi, penulis melakukan beberapa implementasi yang nantinya dapat dibandingkan performansinya yaitu sebagai berikut.

A. Pencarian dengan Fungsi Hash

Dalam mengoptimasi pencarian dengan fungsi hash dapat dilakukan dengan dua cara yaitu dengan membentuk suatu array buckets seperti pada implementasi hash table atau dengan membentuk tabel basis data baru yang dapat meniru fungsi hash table.

Metode untuk implementasi hash table dengan array buckets lebih fleksibel dikarenakan dapat menggunakan fungsi hash apapun. Sementara itu, metode dengan membuat tabel basis data lebih mudah diimplementasikan.

Dalam implementasi hash table menggunakan fungsi hash digunakan metode sebagai berikut.

- Buat array dengan panjang n di mana semakin panjang n maka satu bucket pada array akan menampung lebih sedikit alamat record yang dapat menjadikan hash table lebih efisien.
- Hitung hash nilai record yang telah terenkripsi, hash = hash (value)
- Hitung indeks bucket, idx = hash % n
- Untuk setiap item pada bucket, lakukan iterasi sekuensial untuk mencari nilai tersebut

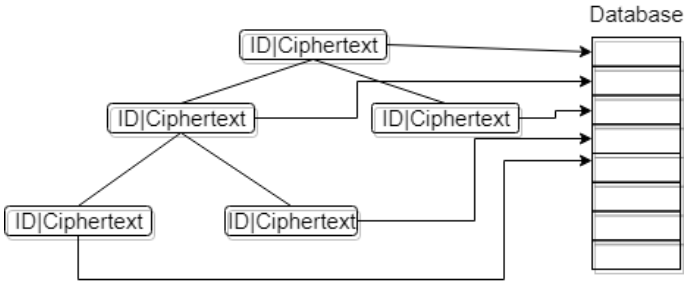
Adapun untuk implementasi hash table menggunakan tabel basis data diimplementasikan sebagai berikut.

- Buat tabel baru dengan kolom pertama adalah nilai hash dari nilai terenkripsi dan kolom kedua adalah foreign key yang menunjuk ke record nilai terenkripsi tersebut.
- Tambahkan indeks pada kolom hash
- Hitung hash nilai yang akan dicari, hash = hash(value)
- Lakukan query pencarian atas nilai hash pada kolom hash pada tabel baru SELECT `encrypted_fk` FROM `new_hashtable` WHERE `hash` = hash;
- Lakukan seleksi pada record yang diacu oleh foreign key, SELECT * FROM `original_table` WHERE `encrypted_column` = `encrypted_fk`;.

B. Pencarian dengan Pohon Biner

Pencarian dengan pohon biner memungkinkan untuk tidak mencari secara sekuensial, tetapi dengan pencarian biner yang kompleksitasnya O(log n). Perbedaan metode ini dengan pohon pencarian biner adalah pembentukan pohon binernya yang berdasarkan nilai plaintext, bukan nilai simpul tersebut.

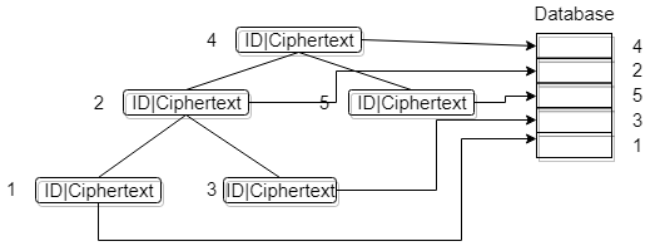
Untuk dapat memetakan simpul pohon ke suatu record pada tabel basis data dapat digunakan suatu id baru. ID tersebut disisipkan pada simpul pohon dan ditambahkan sebagai kolom tabel.



Gambar 5 pemetaan simpul pohon biner ke baris basis data

C. Penyortiran dengan Pohon Biner

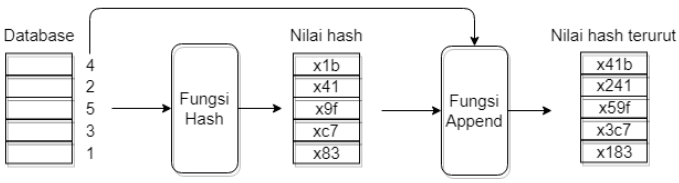
Penyortiran dengan pohon biner memiliki langkah-langkah yang sama dengan pencarian dengan pohon biner. Pembentukan pohon biner ini sama dengan pohon biner untuk pencarian. Sementara untuk proses penyortirannya dilakukan mulai dari simpul terkiri sampai simpul terkanan secara berurutan.



Gambar 6 penyortiran pohon biner

D. Penyoritran dengan Menambahkan Urutan pada Hash

Dalam menambahkan keterurutan pada nilai hash dapat dilakukan dengan menambahkan nomor urut pada awal hash. Dengan menambahkan awalan nomor urut ini, operasi perbandingan dari nilai hash dapat dilakukan sesuai dengan perbandingan nilai plaintext.



Gambar 7 penambahan urutan pada hash

Implementasi cara ini mudah, namun cara ini memiliki kelemahan yaitu harus mengetahui urutan seluruh data terlebih dahulu. Selain itu, penggunaan ini menambah ukuran data pada tiap record. Jika setiap record bertambah sebanyak satu byte, maka untuk 1.000.000 record akan bertambah sebanyak satu mega byte.

E. Penyortiran dengan Order Preserving Hash

Untuk menentukan nilai hash yang memiliki keterurutan yang sama, fungsi hash harus bisa menjamin semua operasinya tidak mengubah keterurutan nilai atau membalik semua keterurutan. Oleh karena itu, fungsi order preserving hash dibuat sederhana untuk bisa menjamin hal tersebut.

Salah satu operasi yang dapat menghasilkan order preserving hash adalah operasi xor. Operasi xor dapat membalik

keterurutan, sehingga jika semua operasi yang ada pada fungsi hash adalah fungsi xor, nilai yang akan dihasilkan akan memiliki keterurutan yang sama atau terbalik.

Salah satu algoritma yang memanfaatkan sifat keterurutan xor adalah algoritma hash pearson yang pseudocode nya sederhana seperti di bawah ini. Di mana T adalah suatu array pearson yang digunakan untuk mensubtitusi dan C adalah isi pesan.

```
h := 0
for each c in C loop
    h := T[ h xor c ]
end loop
return h
```

V. Eksperimen dan Hasil Uji

Untuk memperkirakan perbandingan performasi beberapa usulan, dilakukan beberapa pengujian untuk melihat perbandingan lama waktu untuk proses pencarian dan pengurutan beberapa jenis ukuran data. Selain itu dilakukan pengujian terhadap konsumsi memori yang digunakan pada proses dengan menggunakan kelas Instrumentation untuk mengetahui ukuran memori yang digunakan untuk satu objek.

A. Pengujian Pencarian Data

Pengujian ini dilakukan untuk jenis data integer dan varchar yang kondisi awalnya acak dan terurut yang telah terenkripsi dengan algoritma AES. Pengujian ini dilakukan untuk membandingkan metode hash table dan jenis hashnya dengan pemetaan keterurutan ciphertext yang dikombinasikan dengan binary search yang hasilnya seperti berikut.

Ukuran data (baris)	Waktu rata-rata (ms)
100.000	954
1.000.000	15293

Tabel 1 Performansi pencarian data terenkripsi tanpa pengindeksan

Fungsi hash	Ukuran data (baris)	Waktu rata-rata (ms)
SHA1	100.000	24
	1.000.000	33
MD5	100.000	29
	1.000.000	38
SHA2 (224)	100.000	41
	1.000.000	61

Tabel 2 Perbandingan performansi algoritma hash pada pencarian data acak

Fungsi hash	Ukuran data (baris)	Waktu rata-rata (ms)
SHA1	100.000	20
	1.000.000	36
MD5	100.000	25
	1.000.000	39
SHA2 (224)	100.000	64
	1.000.000	121

Tabel 3 Perbandingan performansi algoritma hash pada pencarian data terurut

Ukuran data (baris)	Waktu (ms)
100.000	55
1.000.000	761

Tabel 4 Performansi pohon biner dalam pengurutan data terenkripsi acak

Ukuran data (baris)	Waktu (ms)
100.000	24
1.000.000	142

Tabel 5 Performansi pohon biner dalam pengurutan data terenkripsi terurut

Ukuran data (baris)	Memori rata-rata (KB)
100.000	31
1.000.000	418

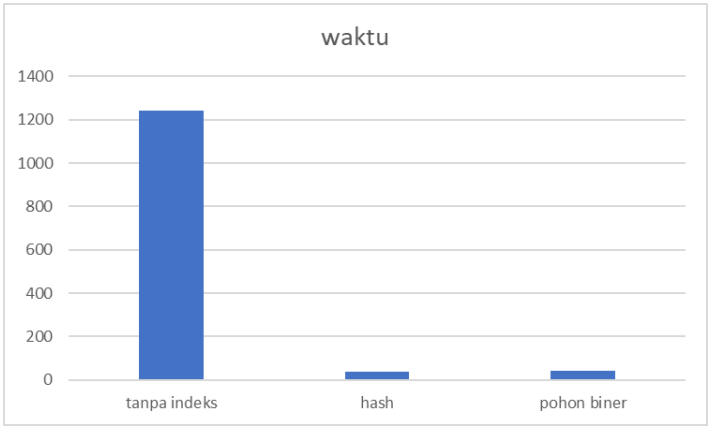
Tabel 6 Performansi penggunaan memori pada pencarian tanpa pengindeksan

Ukuran data (baris)	Memori rata-rata (KB)
100.000	561
1.000.000	1083

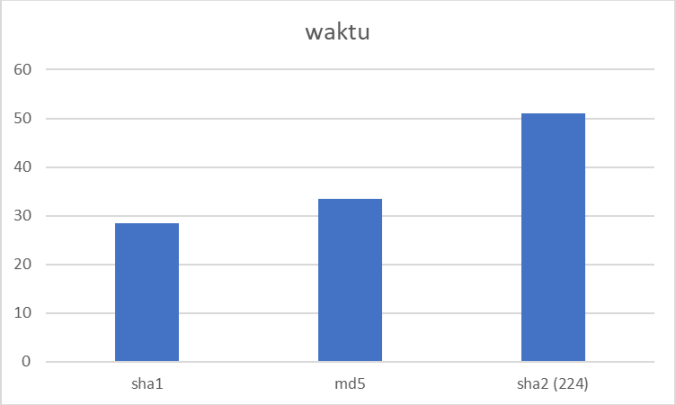
Tabel 7 Performansi penggunaan memori pada pencarian dengan pengindeksan menggunakan hash

Ukuran data (baris)	Memori rata-rata (KB)
100.000	1028
1.000.000	14180

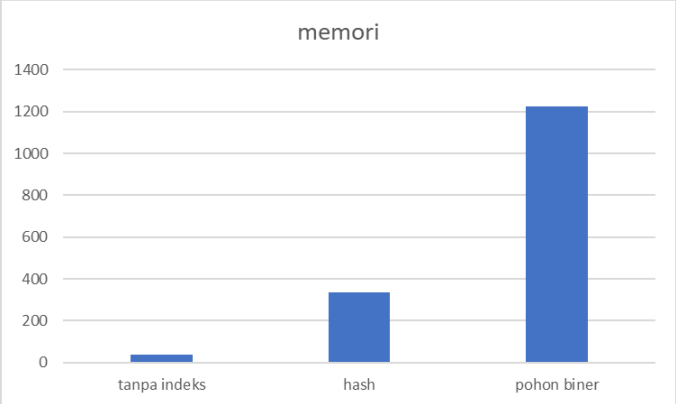
Tabel 8 Performansi penggunaan memori pada pencarian menggunakan pohon biner



Gambar 8 perbandingan performansi waktu untuk pencarian data



Gambar 9 perbandingan performansi waktu pencarian untuk fungsi hash



Gambar 10 perbandingan performansi memori untuk pencarian data

B. Pengujian Pengurutan Data

Pengujian ini dilakukan untuk jenis data sortable seperti integer, varchar, atau timestamp. Pengujian dilakukan terhadap data acak yang telah dienkripsi pada kolom yang akan digunakan sebagai basis dalam pengurutan.

Ukuran data (baris)	Waktu rata-rata (ms)
1.000	5375
10.000	61831

Tabel 9 Performansi pengurutan data terenkripsi tanpa pengindeksan

Ukuran data (baris)	Waktu rata-rata (ms)
1.000	1481
10.000	9912

Tabel 10 Performansi pengurutan menggunakan pohon biner

Fungsi hash	Ukuran data (baris)	Waktu rata-rata (ms)
SHA1	1.000	1084
	10.000	9148
MD5	1.000	814
	10.000	9147
SHA2 (224)	1.000	1183
	10.000	15801

Tabel 11 performansi pengurutan menggunakan fungsi hash yang ditambahkan nomor urut

Ukuran data (baris)	Waktu rata-rata (ms)
1.000	1729
10.000	18294

Tabel 12 Performansi pengurutan menggunakan order preserving hash function

Ukuran data (baris)	Memori rata-rata (KB)
100.000	181
1.000.000	1759

Tabel 13 Performansi penggunaan memori pada penyortiran tanpa pengindeksan

Ukuran data (baris)	Memori rata-rata (KB)
100.000	8571
1.000.000	81593

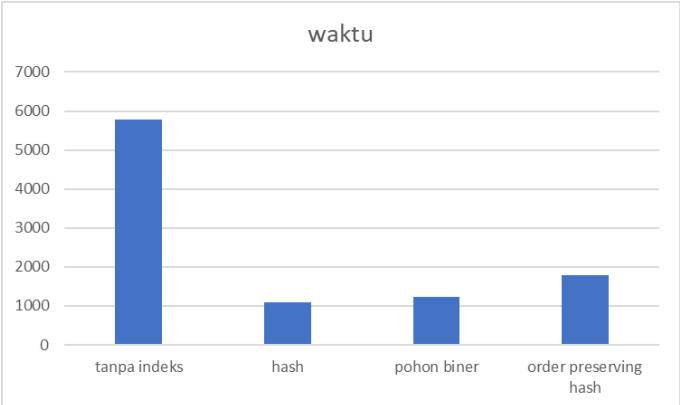
Tabel 14 Performansi penggunaan memori pada penyortiran menggunakan pohon biner

Ukuran data (baris)	Memori rata-rata (KB)
100.000	5681
1.000.000	65819

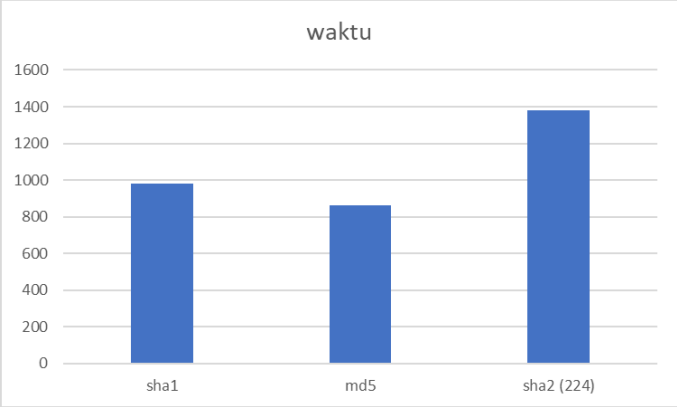
Tabel 15 Performansi penggunaan memori pada penyortiran menggunakan hash yang ditambah nomor urutan

Ukuran data (baris)	Memori rata-rata (KB)
100.000	4817
1.000.000	47194

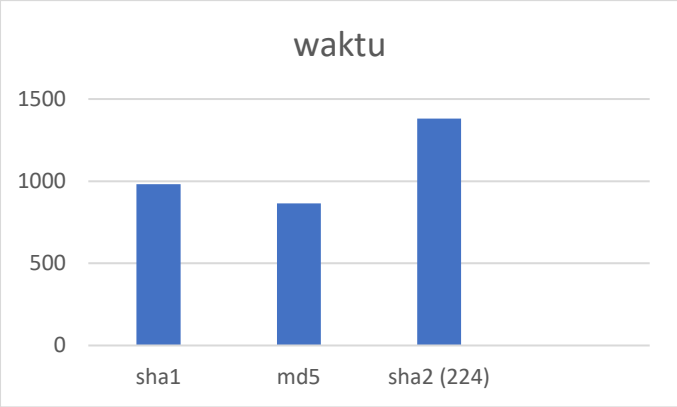
Tabel 16 Performansi penggunaan memori pada penyortiran menggunakan order preserving hash



Gambar 11 perbandingan performansi waktu untuk penyortiran data



Gambar 12 perbandingan performansi waktu penyortiran untuk fungsi hash



Gambar 13 perbandingan performansi waktu untuk penyortiran data

VI. Analisis Hasil

A. Analisis Performansi

Dalam pengujian pencarian data, penerapan hash table akan membantu optimasi pemrosesan query. Penggunaan hash table akan membuat waktu pemrosesan query mendekati nilai konstan. Sementara penggunaan pohon biner tidak terlalu konsisten dan waktu pencarian akan semakin lama jika data semakin besar.

Sementara untuk fungsi hash yang paling efisien adalah SHA1 untuk pencarian dan MD5 untuk penyortiran. Tetapi kedua fungsi hash ini memiliki perbedaan yang tidak terlampau jauh sehingga bisa dipertukarkan.

Sementara pada proses pengurutan data, metode order preserving hash dan metode pohon biner tidak terlalu berbeda hasilnya. Namun, kedua metode ini dapat beroperasi jauh lebih cepat dibandingkan dengan pengurutan tanpa menggunakan kedua metode ini.

B. Analisis Keamanan

Penggunaan metode ini telah dapat menjadikan enkripsi menjadi aman dan tidak terlalu mengorbankan performansi. Administrator basis data tidak akan dapat membaca isi basis data jika proses dekripsi dialihkan ke level API atau level aplikasi klien. Namun, pemindahan proses dekripsi ke level API akan menyebabkan performansi API turun dan pemindahan ke level aplikasi klien akan memudahkan pencurian kunci dekripsi. Oleh karena itu, pemindahan risiko ini harus disertai pertimbangan akan halangan lain yang akan ditimbulkannya.

Selain itu, penggunaan order preserving hash function yang meninggalkan urutan dapat menjadi tidak efektif pada kasus tertentu. Salah satu kasus yang paling berbahaya adalah ketika diterapkan pada nilai yang distribusinya terbatas, misalnya gaji pegawai yang ada nilai minimumnya kemudian nilainya naik sejumlah tertentu hingga ke nilai maksimum. Hal ini dapat menjadi dasar untuk menebak nilai sesungguhnya dari suatu data.

VII. Kesimpulan dan Saran

Dari hasil analisis dapat disimpulkan bahwa untuk melakukan query select pada basis data terenkripsi tidaklah efisien dalam segi waktu. Oleh karena itu, segala bentuk optimasi dapat digunakan untuk memperoleh waktu eksekusi yang lebih cepat.

Untuk menangani proses pencarian pada suatu data terenkripsi dapat menggunakan fungsi hash dan pohon biner sebagai indeks yang berfungsi mempercepat pencarian sehingga pencarian tidak harus dilakukan secara sekuensial. Penggunaan hash lebih efisien dalam hal waktu dan selain itu lebih mudah diimplementasikan. Sementara fungsi hash yang paling efisien adalah fungsi SHA-1.

Sementara itu untuk proses penyortiran, penggunaan pohon biner lebih efisien dibandingkan penggunaan fungsi hash, baik itu penambahan nomor urut pada hash ataupun order preserving hash.

Implementasi ini masih jauh dari kata sempurna dikarenakan fungsi order preserving hash yang digunakan telah banyak dinilai tidak sempurna dikarenakan fungsi hash pearson memiliki operasi yang sederhana dan tidak dapat dinilai aman secara kriptografi.

VIII. References

[1] Y. D. W. Asnar, *Database Security*, Bandung, 2014.

[2] V. Beal, "webopedia," [Online]. [Accessed 16 May 2018].

[3] "Inovasi Informatika Indonesia," [Online]. Available: http://i-3.co.id/index-pada-database/. [Accessed 17 May 2018].

[4] "Tutorials Point," [Online]. Available: https://www.tutorialspoint.com/cryptography/cryptography_hash_functions.htm. [Accessed 17 May 2018].

[5] "Mikrolima," [Online]. Available: https://mikrolima.wordpress.com/2010/02/22/orm-object-relational-mapping/. [Accessed 18 May 2018].

[6] R. Munir, "Slide Kuliah Kriptografi," [Online]. Available: http://informatika.stei.itb.ac.id/~rinaldi.munir/Kriptografi/2017-2018/. [Accessed 15 May 2018].

[7] M. Rouse, "techtarget," [Online]. Available: http://searchsecurity.techtarget.com/definition/block-cipher. [Accessed 15 May 2018].

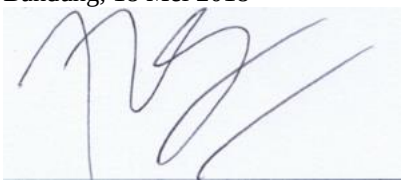
[8] M. Rouse, "What is cryptography," [Online]. Available: http://searchsoftwarequality.techtarget.com/definition/cryptography. [Accessed 15 May 2018].

[9] "techdifferences," [Online]. Available: https://techdifferences.com/difference-between-confusion-and-diffusion.html#Definition. [Accessed 15 May 2018].

IX. PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Mei 2018



Nugroho Satriyanto