e-ISSN: 2549-7952
p-ISSN: 2580-3336

Optimasi Query untuk Pencarian Tempat Wisata Menggunakan Penguraian Kalimat dan *Algoritma Levenshtain Distance* di Kabupaten Rote Ndao

Risky Vridel Eduard Pandie[1], Ardi Sanjaya[2], Julian Sahertian[3]
[1,2]Teknik Informatika, Fakultas Teknik, Universitas Nusantara PGRI Kediri
[3]Teknik Mesin, Fakultas Teknik, Universitas Nusantara PGRI Kediri
E-mail: [1]*riskypandie28@gmail.com*, [2]*dersky@gmail.com*, [3]*juliansahertian@unpkediri.ac.id*

Abstrak *– Pencarian data pada suatu sistem informasi yang belum menggunakan algoritma pencarian sehingga hasil pencarian dari system tersebut belum optimal. Oleh karena itu, perlu adanya implementasi algoritma yang membantu menghasilkan hasil pencarian yang cepat dan optimal. Dalam penelitian ini bertujuan untuk mengoptimalkan query untuk pencarian tempat wisata menggunakan penguraian kalimat dan algoritma levenshtain distance. Metode penelitian yang digunakan dalam penelitian ini yaitu alur optimasi query, penyusunan query, penguraian kalimat dan algoritma levenshtain distance. Hasil pengujian menunjukan bahwa pencarian data menggunakan penguraian kalimat dan algoritma levenshtain distance mampu menemuka data yang dimaksud. Sebaliknya jika data yang dicari tidak berurutan atau diacak masih mampu menemukan.*

Kata Kunci *- Levenshtain,Optimasi, Pencarian Data, Query*

1. PENDAHULUAN

Jaringan *internet* semakin luas dengan cepat sehingga sangat mudah untuk digunakan dimana saja secara cepat dan akurat. Hal ini nampak dari semakin banyak rancangan *web* yang muncul karena pesatnya perkembangan dunia maya yang berbasis *web* dapat memberi berbagai kemudahan dalam setiap sektor kehidupan manusia. Demikian pula dalam sektor parawisata yang merupakan salah satu sektor potensial di Indonesia, termasuk Kabupaten Rote Ndao yang terletak di Provinsi Nusa Tenggara Timur. Kabupaten Rote Ndao memiliki lokasi wisata yang sangat bervariasi, seperti wisata pantai dan wisata alam. Namun penyebaran informasi mengenai pesona wisata ini masih dilakukan secara manual melalui penyebaran brosur, pemasangan poster, dan lain-lain. Hal ini cukup riskan di tengah perkembangan teknologi yang semakin pesat dan karena itu sudah saatnya sistem pengelolah informasi manual ini digantikan dengan sistem pengelolah informasi menggunakan teknologi.

Pada penelitian sebelumnya yang berjudul Optimasi Query Pada Sistem FAQ Di Suara Warga Universitas Negeri Semarang mengimplementasikan dan mengetahui model *query* dengan kinerja terbaik pada sistem FAQ di Suara Warga UNNES. Penelitian tersebut diujikan sebanyak 25 data Suara Warga UNNES dan 45 data dari inputan 9 responden mahasiswa UNNES pada 5 topik bahasan dengan 3 model *query*. Hasil pengujian yang dilakukan nilai recall 0.92, precision 0.41, fmeasure 0.56 dengan model *query* isi, nilai recall 0.86, *precision* 0.59, *fmeasure* 0.70 dengan model *query* isi dengan *feature selection*, nilai recall 0.88, *precision* 0.78, *fmeasure* 0.82 dengan model *query* judul. Dari hasil tersebut model *query* dari judul *input* memiliki nilai paling tinggi dari ke-3 pengujian tersebut [1].

Penelitian lain yang berjudul Optimasi Query Untuk Pencarian Data Menggunakan Penguraian Kalimat menggunakan 3 skenario uji coba dimana setiap scenario mengambil 5 contoh pencarian dengan beberapa kata acak dari judul yang hendak dicari. Hasilnya sebelum dilakukan optimasi indetifikasi personal membutuhkan waktu 2 dan setelah dioptimasi 26 [2]. Dari hasil penelitian sebelumnya oleh karena itu penulis disini ingin melakukan penelitian optimasi *query* untuk pencarian tempat wisata menggunakan penguraian kalimat dan algoritma *levenshtain distance*.

Berdasarkan uraian diatas, maka dapat dirumuskan permasalahan, yaitu

1) Bagaimana merancang dan membuat suatu sistem yang mampu memberikan informasi tentang obyek wisata?
2) Bagaimana menghasilkan *query* yang optimal untuk pencarian data dengan menggunakan kunci pencarian yang dipisah-pisah/diurai susunannya ?

e-ISSN: 2549-7952
p-ISSN: 2580-3336

Pada penelitian ini mencakup beberapa batasan yaitu:

1) Sistem ini di khususkan untuk memberikan informasi obyek-obyek wisata di Pulau Rote yaitu wisata pantai
2) Pada pengujian, menggunakan pembanding berupa *query* dengan kata kunci pencarian yang tidak diurai.
3) Menggunakan bahasa pemrograman PHP dan *database* MySQL.

Berdasarkan perumusan masalah diatas maka tujuan penelitian ini yaitu :

1) Untuk merancang dan membuat suatu sistem informasi tentang obyek-obyek wisata yang ada di Pulau Rote.
2) Untuk mengoptimalkan pencarian data dengan menerapkan metode Penguraian Kalimat dan Algoritme Levenshtein Distance.
3) Penelitian ini diimplementasikan pada pencarian tempat wisata.

1.1. Landasan Teori

Query

Pengertian *query* adalah suatu kemampuan untuk menampilkan data dari *database* untuk diolah lebih lanjut yang biasanya diambil dari tabel-tabel dalam *database*. Pengertian *query* yang lain adalah pertanyaan (question) atau permintaan (order) informasi tertentu daru sebuah *database* yang tertulis dalam format tertentu. *Query* dapat didefinisikan sebagai perintah-perintah untuk mengakses data pada *database* [2].

Penguraian Kalimat

Penguraian kalimat menjadi bagian kata untuk kunci pencarian menggunakan spasi sebagai ciri untuk proses pemisah an/penguraian. Mula-mula ditetapkan variable aw dan ak dengan nilai nol. Mulai dari index awal dan dengan penambahan pergeseran pergerakan sebesar 1, jika didapati spasi maka nilai variabel ak diubah menjadi nilai index yang sekarang. Lalu mengambil nilai *substring* dari nilai awal aw dan nilai akhir ak dan disimpan pada variabel kata dengan tipe *array*. Diulang seterusnya sampai mencapai index akhir dari kalimat. Berikut potongan kode program yang diambil dari peneliti sebelumnya dengan judul Optimasi Query Untuk Pencarian Data Menggunakan Penguraian Kalimat [2], untuk mengurai kalimat :

```
$aw=0; $ak=0; $in=0;
For
$i=0;$i<=strlen($cari);$i++){
if (substr($cari,$i,1)==' '){
$ak=$i;$kata[$in]=substr($cari,
   $aw,$ak-$aw);
$in=$in + 1;
$aw=$ak;}}
$kata[$in]=substr($cari,$aw+1,
strlen($cari)
($aw+1));
```

Algoritma Levenshtain Distance

Dalam teori informasi, *Levenshtein distance* dua string adalah jumlah minimal operasi yang dibutuhkan untuk mengubah suatu *string* ke *string* yang lain, di mana operasi-operasi tersebut adalah operasi penyisipan, penghapusan, atau penyubstitusian sebuah karakter. Algoritma ini dinamakan berdasarkan Vladimir Levenshtein yang ditemukannya pada tahun 1965. Pada makalah ini, *Levenshtein distance* dirujuk dengan menggunakan kata jarak saja agar lebih singkat.Algoritma dasar penentuan jarak dua string ini, dapat dibentuk melalui hubungan rekursif.

```
Basis:
levDis(“”,“”=0
levDis(s,“”)=levDis(“”,s)=
|s|
```

Di atas terdapat dua basis. Baris pertama menyatakan dengan jelas bahwa dua *string* kosong tidak memiliki jarak, berarti untuk mengubah *string* yang satu ke yang lain tidak diperlukan operasi apapun. Baris kedua menyatakan bahwa jarak antara suatu string tidak kosong dengan *string* kosong adalah sebesar panjang (jumlah karakter) di dalam *string* yang tidak kosong.

Rekurens:

```
levDis(s1+c1,s2+c2)=
min((levDis(s1,s2)+(if(c1=c2)
then 0
else 1
endif  ) ),
(levDis(s1+c1,s2)+1),
(levDis(s1,s2+c2)+1))
```

Di atas tertulis bahwa kedua *string* yang dibandingkan tidak kosong. Keduanya memiliki karakter terakhir c1 dan c2. Di sini dijelaskan bahwa terdapat tiga alternatif untuk menentukan jarak/ kedua *string*. Pertama, Jika c1 dan c2 sama, maka c1 dan c2 tidak perlu dipertukarkan berarti jaraknya levDis(s1,s2) + 0, jika berbeda berarti hanya tinggal mengubah c1 menjadi c2 saja, berarti jaraknya levDis(s1,s2) + 1. Dalam hal ini operasi yang dilakukan adalah operasi substitusi. Kedua, dapat juga dilakukan operasi penghapusan c1 dari s1 dan mengubahnya menjadi s2 + c2 sehingga jaraknya menjadi levDis(s1,s2+c2) + 1. Ketiga, mirip seperti yang kedua dapat dilakukan juga operasi penyisipan c2 pada s1 + c1 yang telah diubah menjadi s2 sehingga jaraknya adalah levDis(s1+c1,c2) + 1. Ketiga alternatif di atas adalah semua kemungkinan perubahan yang ada, dan dari antara ketiganya dicariyang mana yang paling sedikit jaraknya dengan fungsi min yang mencari nilai paling minimum di antara tiga nilai.

Dalam implementasi algoritma rekursif ini, terdapat tiga kali pemanggilan rekursif untuk setiap rekurens. Hal ini membuat algoritma ini menjadi sangat lambat dan hanya baik digunakan pada *string*

e-ISSN: 2549-7952
p-ISSN: 2580-3336

yang terdiri dari karakter-karakter yang sedikit saja. Oleh karena itu, pemeriksaan lebih lanjut menunjukkan bahwa jarak s1 dan s2 bergantung pada jarak s1' dan s2' saja di mana s1' lebih pendek dari s1, dan s2' lebih pendek dari s2. Sementara jarak s1' dan s2' bergantung pada jarak s1" dan s2" dimana keduanya lebih pendek dari yang sebelumnya. Hal ini menunjukkan bahwa teknik pemrograman dinamis dapat digunakan. Untuk menghitung jaraknya tanpa menggunakan proses rekursif, digunakan matriks (n + 1) × (m + 1) di mana n adalah panjang *string* s1 dan m adalah panjang *string* s2. Berikut dua *string* yang akan digunakan sebagai contoh:

BARU
BATU

Jika kita melihat sekilas, kedua *string* tersebut memiliki jarak 1 kata. Berarti untuk mengubah *string* BARU menjadi BATU diperlukan 1 operasi, yaitu:

Mensubstitusikan R dengan T

BARU Æ BATU

Dengan menggunakan representasi matriks dapat ditunjukkan tabel berikut:

		B	A	R	U
	0	1	2	3	4
B	1				
A	2				
T	3				
U	4				

Pada tabel ini, elemen baris 1 kolom 1 (M[1,1])adalah jumlah operasi yang diperlukan untuk mengubah *substring* dari kata BARU yang diambil mulai dari karakter awal sebanyak 1 (B) ke *substring* dari kata BATU yang diambil mulai dari karakter awal sebanyak 1 (B). Sementara elemen M[3,3] adalah jumlah operasi antara BAR(*substring* yang diambil mulai dari karakter awal sebanyak 3) dengan BAT(*substring* yang diambil mulai dari karakter awal sebanyak 3). Berarti elemen M[p,q] adalah jumlah operasi antara *substring* kata pertama yang diambil mulai dari awal sebanyak p dengan *substring* kata kedua yang diambil dari awal sebanyak q. Sehingga dengan peraturan ini matriks dapat diisi, menghasilkan:

		B	A	R	U
	0	1	2	3	4
B	1	0	1	2	3
A	2	1	0	1	2
T	3	2	1	1	2
U	4	3	2	2	1

Elemen terakhir (yang paling kanan bawah) adalah elemen yang nilainya menyatakan jarak kedua *string* yang dibandingkan. Sehingga algoritma untuk mengisi matriks sesuai dengan yang di atas adalah:

```
function levDis(s1:string,s2:
string):integer kamus
i,j,cost:integer
m:array[0..s1.length,0..s2.le
ngth] of integer algoritma
for i Å 0 to s1.length do
for j Å 0 to s2.length do
if i = 0 then
m[i,j]Åj{perbandingan  dengan
kosong}
else if j = 0 then
m[i,j]Åi{perbandingan  dengan
kosong}
else{implementeasi
pemrograman dinamis}
if s1[i] = s2[j] then
cost Å 0
else
cost Å 1
m[i,j] = minimum (
m[i-1,j-1]+cost,{substitusi}
m[i-1,j]+1,{penghapusan}
m[i,j-
1]+1,{penambahan})return
m[s1.length, s2.length]
```

Algoritma ini memiliki kompleksitas waktu O(mn), dengan m dan n adalah panjang masing-masing *string* yang diperbandingkan. Dengan kata lain, jika kedua *string* memiliki panjang yang sama kompleksitas waktunya adalah O(n2). Kompleksitas ruang untuk algoritma ini adalah juga O(mn) atau O(n2) jika kedua *string* memiliki panjang sama. Namun, jika kita melihat bagian implementasi rekurens pada algoritma ini, sebenarnya suatu baris matriks hanya membutuhkan data dari baris sebelumnya saja, berarti hanya dibutuhkan 2 × n ruang memori untuk menyimpannya.

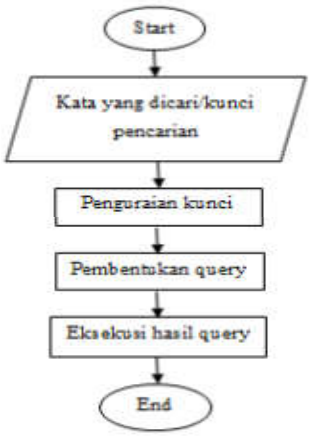
2. METODE PENELITIAN

2.1. Alur

Adapun alur optimasi *query* untuk pencarian tempat wisata sebagai berikut:

1) Memasukan kunci pencarian atau kata yang dicari berupa susunan kata ataupun kalimat.
2) Kemudian kata kunci pencarian di urai menjadi kata yang terpisah.
3) Selanjutnya menyusun *query* lalu *query* yang sudah disusun ditampilkan sebagai hasil pencarian.

e-ISSN: 2549-7952
p-ISSN: 2580-3336



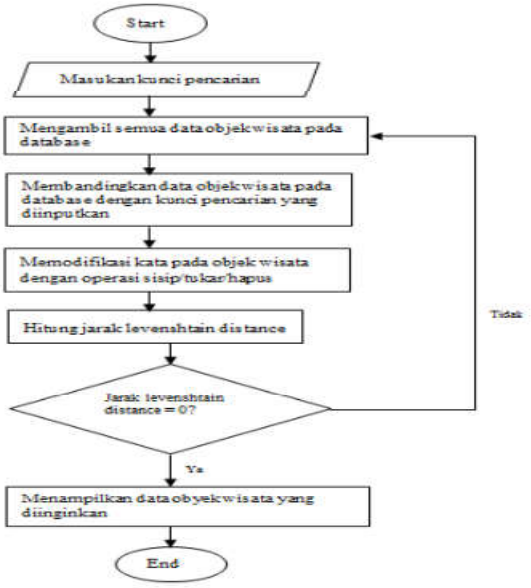
Gambar 1. Alur proses optimasi query

2.2. Penyusunan Query

Gambar 2. Struktur query untuk optimasi

Perintah *select* nama digunakan untuk memilih kolom/field dengan nama obyek wisata dari table obyek yang ada di *database*. Kemudian dikombinasikan dengan operator *OR* untuk menggabungkan syarat pencarian/kemiripan teks berdasarkan hasil penguraian kalimat[2].

2.3. Flowchart Sistem

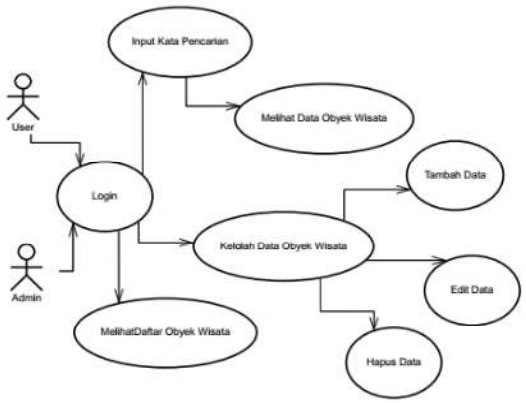


Gambar 3. Flowchart System

Langkah-langkah yang terdapat pada *flowchart* sistem dapat menjelaskan bagaimana alur kerja yang terdapat di dalam sistem, proses kerja algoritma *Levenshtein Distance* untuk mendapatkan nilai jarak *Levenshtein*,yaitu:

1) Memberikan *input* berupa data obyek wisata pada kotak pencarian.
2) Selanjutnya data obyek wisata yang *diinputkan* akan dibandingkan dengan semua data obyek wisata yang terdapat di dalam *database*.
3) Kemudian akan dilakukan modifikasi kata pada data obyek wisata dengan menggunakan operasi sisip, tukar dan hapus, hal ini dilakukan jika kata yang dicocokkan tidak sama.
4) Setelah modifikasi dilakukan, selanjutnya yaitu menghitung nilai jarak *levenshtein*, nilai *levenshtein* diperoleh dari proses modifikasi.
5) Jika nilai jarak *levenshtein* sama dengan 0 maka data obyek wisata yang diinputkan dengan data obyek wisata yang terdapat di dalam *database* dianggap sama persis sehingga proses perulangan akan berhenti karena dianggap cocok, dan jika tidak maka perulangan akan terus berlanjut.
6) Kemudian untuk data obyek wisata yang dianggap sudah cocok atau mendekati maka akan ditampilkan sebagai hasil pencaarian.

2.4. Use Case Diagram



Gambar 4. Use Case Diagram

Di dalam *use case* diagram dapat digambarkan bahwa terdapat 2 orang aktor yang akan berperan, yaitu *user* dan admin. Untuk memperoleh informasi mengenai obyek wisata yangdicari maka aktor *user* harus melakukan *login*, dengan menginputkan *username* dan *password,* jika berhasil di validasi maka selanjutnya *user* akan masuk ke halaman beranda. kemudian pada menu obyek terdapat kolom pencarian/*search box*, *user* memasukkan *input* berupa data ke dalam *searchbox.* Selanjutnya jika pencarian yang dilakukan berhasil maka sistem akan menampilkan sebuah halaman yang berisi kumpulan dari obyek wisata yang berkaitan dengan data yang diinputkan, kemudian *user* harus mengklik salah satu dari data yang ditampilkan untuk dapat melihat informasi obyek wisata secara lengkap.

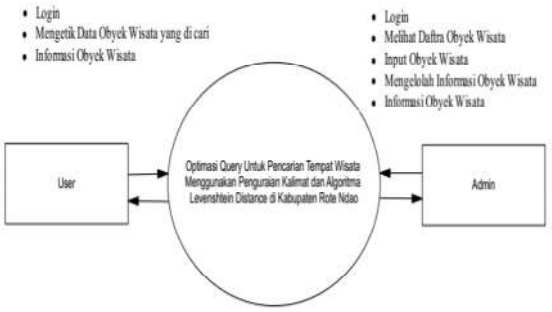
Selanjutnya untuk bagian admin yaitu terlebih dahulu melakukan *login*, dengan menginputkan

e-ISSN: 2549-7952
p-ISSN: 2580-3336

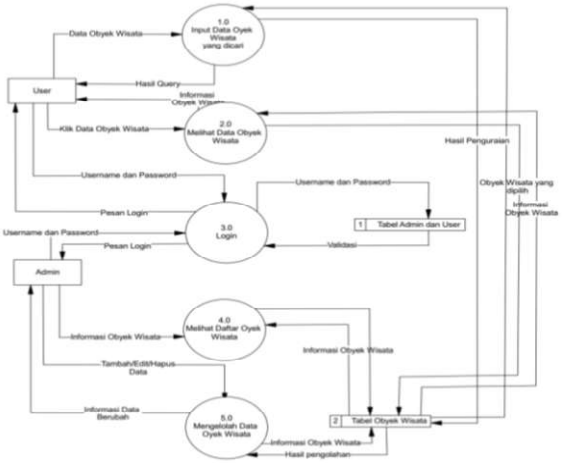
username dan *password*, jika *username* dan *password* berhasil divalidasi, maka selanjutnya, *admin* akan masuk ke halaman beranda. Untuk melakukan pengelolahan admin dapat melakukan penambahan, pengeditan dan penghapusan obyek wisata.

2.5. Data Flow Diagram

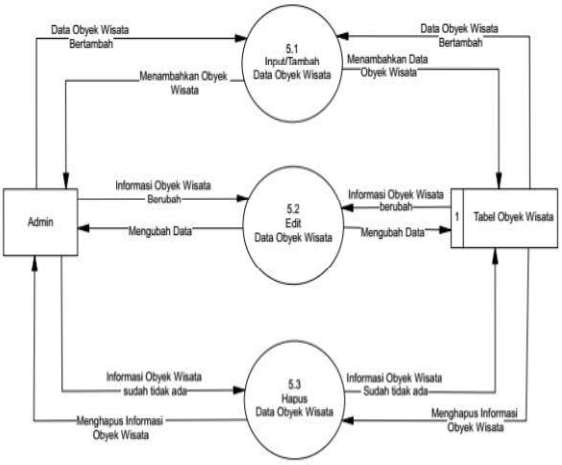
1) DFD Level 0



Gambar 5. DFD level 0



Gambar 6. DFD level 1



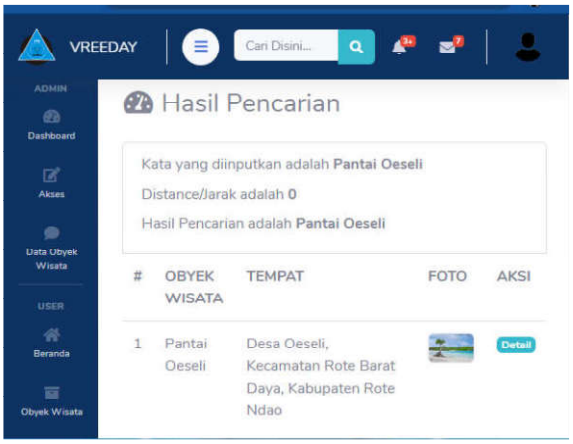
Gambar 7. DFD level 2

3. HASIL DAN PEMBAHASAN

Dalam melakukan pengujian, penulis membuat 2 skenario uji coba dan mempunyai panjang dan isi yang berbeda dimana tiap scenario dilakukan 3 contoh pencarian dengan melakukan beberapa kata acak dari data yang hendak dicari. Pada scenario uji coba 1 dilakukan dengan 3 kunci pencarian tanpa optimasi atau berurutan masih bisa menemukan data. Selanjutnya pada skenario uji coba 2 dengan data kunci pencarian yang sama tetapi dilakukan dengan acak atau tidak berurutan masih bisa menemukan data. Hasil uji coba dapat dilihat pada tabel berikut :

Tabel 1. Hasil uji coba

No	Uji coba	Data yang dicari	Hasil sistem
1	Scenario uji coba 1	- pantai oeseli - labirin pantai mulut seribu - pantai nemberala	- berhasil - berhasil - behasil
2	Scenario uji coba 2	- oeseli pantai - labirin pantai mulut - nemberala	-berhasil -berhasil -berhasil



Gambar 8. Uji coba skenario 1

Gambar 9. Uji coba skenario 2

e-ISSN: 2549-7952
p-ISSN: 2580-3336

4. SIMPULAN

Dari hasil penelitian yang diperoleh maka dapat disimpulkan bahwa optimasi pada *query* mampu menemukan hasil dengan kata kunci pencarian yang diacak atau tidak berurutan. Dengan menggunakan algoritma *levenshtein distance* dapat memudahkan menentukan pencarian berdasarkan *string* secara efektif dan efesien. Algoritma *levenshtain distance* sangat membantu dalam pencocokan *string.*

Saran untuk penelitian selanjutnya adalah melengkapi proses pencarian dengan seleksi tingkat kemiripan dari kata pencarian terhadap hasil data yang ditemukan dan menampilkannya dengan batasan hasil pencarian yang memiliki kemiripan terbesar sehingga dapat diperoleh hasil yang lebih optimal.

DAFTAR PUSTAKA

[1] Tama, G A, *Optimasi Model Query Pada Sistem FAQ Di Suara Warga Universitas Negeri Semarang.* http://lib.unnes.ac.id/20570/1/5302411232-S.pdf, diakses 29 Januari 2020

[2] A. Sanjaya, "*Optimasi Query Untuk PencarianData Menggunakan Penguraian Kalimat,*"no. November, pp. 6–7, 2016

[3] Aripin, *Meningkatkan Efektifitas Pengelolaan Database Dengan Optimasi Query.* http://dinus.ac.id/wbsc/assets/dokumen/majalah/MENINGKATKAN_EFEKTIFITAS_PENGELOLAAN_DATABASE_DENGAN_OPTIMASI_SQL.pdf, diakses 29 Januari 2020

[4] Tama, G A, *Optimasi FAQ Di Suara Warga Universitas Negeri Semarang.* http://lib.unnes.ac.id/20570/1/5302411232-S.pdf, diakses 29 Januari 2020 and A. D. Kurnawan, "*Optimasi Query Untuk Pencarian Data Menggunakan Penguraian Kalimat*," no. November, pp. 6–7, 2016

optimal.

optimal.

[2] A. Sanjaya, "*Optimasi Q*
Menggunakan Penguraic
pp. 6–7, 2016