PENERAPAN ALGORITMA *START END MID* UNTUK MENDETEKSI KESALAHAN LOGIKA *STRUCTURED QUERY LANGUAGE*

Jevri Tri Ardiansah[1)], Aji Prasetya Wibawa [2)], Triyanna Widiyaningtyas [3)]

[1), 2), 3)]Teknik Elektro Universitas negeri malang
Jl Semarang No. 5 Malang 65145
Email : jev.ardian@gmail.com[1)], aji.prasetya.ft@um.ac.id[2)], triyannaw@gmail.com[3)]

Abstrak

Basis data merupakan bidang keilmuan penting yang harus dikuasai oleh mahasiswa jurusan teknik informatika, mengingat setiap sistem yang dikembangkan pasti memiliki basis data untuk penyimpanan informasi. Observasi yang dilakukan kepada mahasiswa Pendidikan Teknik Informatika Universitas Negeri Malang menghasilkan kesimpulan bahwa basis data adalah matakuliah yang dianggap mudah secara teori tetapi sulit diimplementasikan menggunakan SQL. Hal tersebut dikarenakan terdapat dua jenis kesalahan yang dapat terjadi, yaitu kesalahan sintaksis dan kesalahan logika. Kesalahan sintaksis dapat dikoreksi oleh compiler, sedangkan kesalahan logika dapat diketahui dengan proses pembandingan antara pernyataan yang sedang dituliskan dengan pernyataan yang seharusnya dituliskan. Berdasarkan hal tersebut, maka perlu dikembangkan suatu sistem yang dapat mengetahui kesalahan sintaksis dan kesalahan logika. Penelitian ini bertujuan untuk menghasilkan langkah-langkah yang dapat dilakukan untuk mendeteksi kesalahan logika SQL menggunakan Algoritma Start End Mid yang selanjutnya akan dilanjutkan ke tahap desain dan pengembangan sistem yang sebenarnya.

Kata kunci: *SQL, basis data, kesalahan logika, algoritma pencocokan string, algoritma Start End Mid.*

1. Pendahuluan

1.1. Latar Belakang

Observasi yang dilakukan kepada mahasiswa Pendidikan Teknik Informatika Universitas Negeri Malang menghasilkan kesimpulan bahwa basis data adalah matakuliah yang dianggap mudah secara teori tetapi sulit diimplementasikan menggunakan *Stuctured Query Language* (*SQL)* [1]. *SQL* merupakan bahasa yang digunakan untuk mengakses informasi yang terdapat di basis data. Dalam penerapannya, sangat mungkin terjadi kesalahan atau *error*, yaitu *syntax error* dan *logic error*. *Syntax error* atau kesalahan sintaksis merupakan kesalahan penulisan pernyataan *SQL* yang dapat diatasi oleh *compiler*. Secara otomatis, *compiler* akan memberikan informasi kesalahan. Sedangkan *logic error* atau kesalahan logika terjadi apabila penulisan pernyataan *SQL* telah sesuai dengan standar tetapi tidak mengeluarkan hasil yang sesuai dengan tujuan. Dalam hal ini, *compiler* tidak memberikan informasi kesalahan. Kedua jenis *error* tersebut dapat menghambat proses yang berkaitan dengan manipulasi informasi dalam basis data, baik itu penyimpanan maupun pengambilan data [2].

Penelitian kepada programmer baru dan siswa menghasilkan data bahwa 32% responden mengalami kesalahan logika, 18% mengalami kesalahan sintaksis dan 14% mengalami kesalahan logika dan sintaksis [2]. Dari penelitian tersebut dapat disimpulkan bahwa kemungkinan terjadinya kesalahan logika mencapai 46%, lebih besar dari kemungkinan kesalahan sintaksis.

Untuk dapat mengetahui kesalahan logika yang terdapat dalam pernyataan *SQL*, dapat dilakukan proses pembandingan antara pernyataan yang sedang dituliskan dengan pernyataan yang seharusnya dituliskan [3]. Dengan kata lain, melalui mekanisme pembandingan antara jawaban pengguna dengan jawaban kunci, dapat diperoleh perbedaan antara keduanya yang mengindikasikan adanya kesalahan logika. Salah satu algoritma yang dapat digunakan untuk membandingkan antar pernyataan adalah Algoritma *Start End Mid*. Berdasarkan masalah yang terjadi yaitu besarnya variasi dan kemungkinan terjadinya kesalahan logika serta terdapatnya potensi pemecahan masalah, maka perlu dikembangkan sistem yang dapat mengetahui kesalahan logika pernyataan *SQL* melalui mekanisme pembandingan pernyataan *SQL* pengguna dengan pernyataan *SQL* kunci dengan menggunakan Algoritma *Start End Mid*.

1.2. Tinjauan Pustaka

Untuk memperjelas konsep, berikut dipaparkan beberapa tinjauan pustaka.

1. *Structured Query Language (SQL)*

Dalam suatu sistem diperlukan adanya media penyimpanan data yang berupa basis data. Basis data merupakan kumpulan data yang mana jika data tersebut diolah maka akan menghasilkan suatu informasi yang dibutuhkan [5]. Pengolahan data tersebut melibatkan

suatu bahasa yang disebut dengan *Structured Query Language (SQL)*.

Terdapat dua kategori kesalahan yang dapat terjadi pada pernyataan *SQL*, yaitu kesalahan sintaksis dan kesalahan logika [2]. Berbeda dengan kesalahan sintaksis, kesalahan logika tidak dapat diatasi oleh *SQL compiler*. Oleh karena itu, perlu adanya pemrosesan lebih lanjut mengenai pernyataan *SQL* yang dituliskan oleh pengguna hingga dapat diketahui kesalahan logika yang terjadi. Salah satu caranya yaitu dengan membandingkan antara jawaban pengguna dengan jawaban kunci [3].

Untuk memperjelas konsep kesalahan logika, diberikan contoh tabel pegawai (Tabel 1) yang mengilustrasikan penyimpanan data pegawai dalam suatu perusahaan.

Tabel 1 Ilustrasi Tabel Pegawai

id_peg	nama	status	id_dep
1	Adi	Direktur	1
2	Budi	Magang	2
3	Citra	Sekretaris	3
4	Disa	Sekretaris	2
5	Adi	Direktur	2

Salah satu jenis kesalahan logika seleksi adalah karena ketidaksesuaian kondisi. Kesalahan logika ini terjadi karena terdapat kekeliruan dalam klausa *WHERE* sehingga menghasilkan keluaran yang tidak seharusnya [4]. Sebagai contoh yaitu proses dalam medapatkan data pegawai yang menjabat sebagai direktur dan pegawai yang menjabat sebagai sekretaris (Tabel 2).

Tabel 2 Hasil Seleksi Data Direktur dan Sekretaris

id_peg	nama	status	id_dep
1	Adi	Direktur	1
3	Citra	Sekretaris	3
4	Disa	Sekretaris	2
5	Adi	Direktur	2

Hasil seleksi yang tampak pada Tabel 2 diperoleh dengan menuliskan pernyataan *SQL* seperti berikut:

```
SELECT * FROM Pegawai WHERE status='direktur' OR
status='sekretaris';
```

Jika pengguna menuliskan pernyataan *SQL* seperti berikut:

```
SELECT * FROM Pegawai WHERE status='direktur'
AND status='sekretaris';
```

maka kolom yang dihasilkan akan tampak seperti pada Tabel 3.

Tabel 3 Hasil Kesalahan Logika Seleksi Data Direktur dan Sekretaris

id_peg	nama	status	id_dep

2. Algoritma pencocokan string

Pencocokkan string merupakan permasalahan paling mendasar dari semua permasalahan string lainnya [6]. Teknik untuk menyelesaikan permasalahan pencocokkan string biasanya akan menghasilkan mekanisme lanjutan lain yang akan memproses informasi ke tahap lainnya. Seperti mekanisme pada penerjemah bahasa Indonesia ke bahasa Jawa [7] dimana pencocokan string digunakan pada tahap awal untuk mencocokkan kata-kata bahasa Indonesia dari kalimat pengguna dengan kata-kata dalam basis data. Setelah itu proses akan lanjut ke tahap restrukturisasi kalimat dan tahap-tahap selanjutnya sehingga didapatkan kalimat dalam bentuk bahasa Jawa.

Terdapat beberapa algoritma pencocokan string yang dapat digunakan. Algoritma tersebut antaralain yaitu *Brute Force, Karp Rabin, Knuth Morris Pratt dan Boyer Moore* [8]. Berikut macam-macam algoritma yang memiliki perbedaan dalam menyelesaikan suatu permasalahan.

- Algoritma *Brute Force* merupakan algoritma yang paling sederhana untuk menyelesaikan persoalan pencocokan string [9]. Algoritma ini melakukan pencarian pada setiap posisi di dalam teks dari awal hingga akhir atau dari kiri ke kanan, tidak peduli apakah terdapat pengulangan pola atau tidak.
- Algoritma *Karp Rabin* menggunakan fungsi hashing untuk menemukan suatu pola di dalam teks tertentu [8]. Fungsi hashing ini menyediakan mekanisme sederhana untuk menghindari perbandingan jumlah karakter yang kuadratik seperti algoritma *Brute Force*.
- Algoritma *Knuth Morris Pratt (KMP)* bergerak dari kiri ke kanan seperti algortima *Brute Force* tetapi algoritma ini memiliki kemampuan yang lebih baik dalam hal melakukan pergeseran pola yang dicari dalam suatu teks [9].
- Algoritma *Boyer Moore* dianggap sebagai algoritma pencocokan string yang paling efisien karena algoritma ini menggunakan pergerakan mundur (kanan ke kiri) sehingga pencarian pola akan lebih cepat [8].
- Algoritma *Start End Mid* [10] merupakan pengembangan algoritma *Brute Force*. Ciri khas dari algoritma *Brute Force* yaitu tanpa preprocessing, dan menggeser pencarian satu langkah ke kanan tetap digunakan. Peningkatannya terdapat pada tahap pencarian dimana algoritma ini akan mencocokkan karakter pertama, karakter terakhir dan karakter tengah suatu pola. Kesamaan antar data dapat diketahui dari pola awal, akhir dan tengahnya [10]. Ketika dalam dua data terdapat pola awal, akhir, tengah yang sama, maka akan didapatkan bahwa kedua data tersebut sama dan berlaku sebaliknya. Konsep ini secara otomatis akan menyingkat waktu pemrosesan data.

Meskipun terdapat algoritma lain yang seperti *Karp Rabin, Knuth Morris Pratt dan Boyer Moore,* algoritma *Start End Mid* ini tetap efektif digunakan mengingat jumlah karakter yang akan diproses dalam penelitian ini

tidak terlalu besar sehingga pengaruh perbedaan kompleksitas pencarian antar algoritma tidak terlalu besar. Dalam hal ini digunakan algoritma *Start End Mid* karena penelitian lebih ditekankan pada pemanfaatan algoritma yang mudah dan sederhana untuk menyelesaikan suatu masalah dengan tetap memperhatikan kesesuaian dan keefektifan algoritma.

3. Penelitian yang berhubungan

Terdapat penelitian yang berkaitan dengan pembuatan sumber belajar *SQL* yang dapat menunjukkan kesalahan logika pengguna dan dinamakan dengan *SQL-Tutor* [11]. Penelitian tersebut menggunakan metode *Constraint Based Modelling (CBM)* yang mengimplementasikan *pattern matching* untuk mendapatkan kesalahan yang dilakukan pengguna sehingga sistem mampu menuntun pengguna untuk melakukan pembenaran. Penelitian ini menghasilkan *Intelligent Tutoring System (ITS)* berbasis desktop dengan model pembelajaran *Guided Discovery* sebagai cara untuk memahamkan penggunanya terkait dengan kesalahan logika.

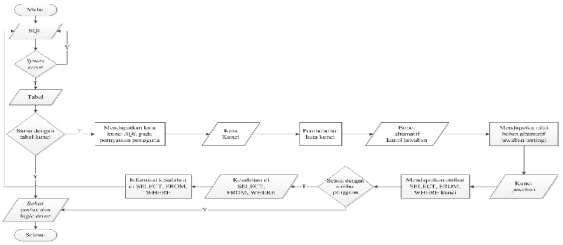
Berbeda dengan penelitian sebelumnya, *SQL Lightweight Tutoring Module* [3] dapat mendeteksi adanya kesalahan logika dengan cara mengintegrasikan sistemnya dengan *LINQ*. *LINQ* merupakan sebuah *tools* dari *Microsoft* yang dapat merepresentasikan pernyataan *SQL* pengguna menjadi bahasa *XML* yang selanjutnya akan dibandingkan dengan pola *XML* yang seharusnya. Perbedaan *XML* mengindikasikan adanya kesalahan logika.

Melalui kesamaan tujuan dengan penelitian sebelumnya yaitu mengembangkan sistem yang dapat mengetahui kesalahan logika *SQL*, *compiler* yang akan digunakan adalah *online SQL compiler* yang telah dikembangkan [12]. *Online SQL compiler* ini telah dapat menginformasikan kesalahan sintaksis dan dapat menghasilkan keluaran berupa tabel dalam format *html*. *Compiler* ini menggunakan skema basis data yang terdapat dalam server website yang digunakan sebagai acuan dalam mengeksekusi pernyataan *SQL* pengguna.

2. Pembahasan

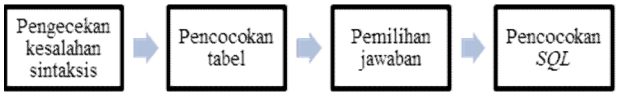
2.1. Metode

Mekanisme kerja sistem dari algoritma *Start End Mid* ditunjukkan dalam *flowchart* seperti pada Gambar 1.



Gambar 1 Mekanisme Pendeteksian Kesalahan Logika

Mekanisme sistem pada Gambar 1 dapat dikategorikan menjadi empat langkah utama dalam proses pengecekan kesalahan logika menggunakan algoritma *Start End Mid*. Langkah-langkah tersebut tampak seperti pada Gambar 6.



Gambar 2 Tahapn Pendeteksian Kesalahan Logika

2.2. Hasil dan Pembahasan

Hasil dari langkah penggunaan algoritma, dapat dijabarkan sebagai berikut:

1. Pengecekan kesalahan sintaksis

Pada tahap pertama, pernyataan *SQL* akan diperiksa oleh *SQL compiler* [12] dimana pada tahap ini sistem akan menginformasikan kesalahan hingga penulisan *SQL* dipastikan benar.

2. Pencocokan antara Tabel Hasil Eksekusi Pengguna (THEP) dengan Tabel Kunci Jawaban (TKJ)

Tahap kedua yaitu berupa pencocokan THEP dengan TKJ. Proses pencocokan ini melibatkan algoritma *Start End Mid* yang akan mencocokan setiap karakter dari tabel tersebut. Jika THEP sama dengan TKJ, maka logika pengguna dianggap benar. Jika terdapat perbedaan, maka sistem akan melakukan pengecekan pernyataan *SQL* pengguna.

Penerapan algoritma *Start End Mid* pada langkah kedua ini adalah sebagai berikut:

a. Menghitung dan membandingkan jumlah karakter *html* dari THEP dan TKJ, jika sesuai lanjut ke tahap berikutnya.
b. Memasukkan setiap karakter dari THEP dan TKJ ke dalam array. Tabel 4 dan Tabel 5 merupakan ilustrasinya.

Tabel 4 Ilustrasi THEP dalam *Array*

i	0	1	2	….	50	….	98	99	100
j	<	t	a	….	a	….	l	e	>

Keterangan:
i : indeks *array*
j : isi *array*

c. Membandingkan karakter pertama antara teks THEP dengan TKJ. Dari poin b, diperoleh bahwa karakter pertama yaitu pada indeks array ke-0 kedua-duanya berisi karakter ‘a’.
d. Membandingkan karakter terakhir yaitu indeks array ke-100. Terdapat karakter yang sama pada indeks tersebut, yaitu karakter ‘>’.
e. Memilih karakter tengah dengan melakukan segmentasi dengan rumus yang ditunjukkan oleh Persamaan (1).

$$Indeks = \frac{indeks\_maks}{2} \quad \text{.....(1)}$$

Keterangan:
Indeks : indeks *array* segmentasi
Indeks_maks : indeks *array* maksimal

Dalam hal ini, perhitungan kasus di atas adalah sebagai berikut:

$$Indeks = \frac{100}{2} = 50$$

Sehingga indeks yang digunakan dalam segmentasi adalah 50 dan terdapat karakter yang sama yaitu karakter 'a'.

f. Membandingkan setiap karakter antara teks THEP dengan TKJ, yaitu dimulai dari indeks pertama hingga indeks terakhir.

Konsep di atas merupakan penerapan dari algoritma *Start End Mid*, dimana jika terjadi perbedaan hasil di setiap tahapannya, maka didapatkan bahwa terdapat perbedaan antara jawaban pengguna dengan kunci. Hal tersebut mengindikasikan bahwa terdapat kesalahan logika pengguna dalam pernyataan *SQL* yang dituliskan.

3. Pemilihan jawaban kunci

Tahap ketiga yaitu proses pengecekan pernyataan *SQL* pengguna yang selanjutnya digunakan untuk pencarian kunci jawaban yang paling mendekati. Hal ini dilakukan karena mengingat terdapat beberapa pendekatan *SQL* untuk kasus yang sama. Pencarian jawaban kunci ini menggunakan mekanisme perangkingan berdasarkan bobot kemiripan yang dimiliki oleh setiap pendekatan kunci jawaban (PKJ) terhadap pernyataan *SQL* pengguna (PSP). Algoritma yang digunakan untuk pembobotan ini adalah Algoritma *Tf-Idf (Term Frequency – Inverse Document Frequency)* [13]. Bobot setiap PKJ kemudian akan dirangking berdasarkan nilai terbesar sehingga nilai bobot yang paling besar akan dipilih sebagai PKJ yang sesuai dengan PSP.

Misalkan terdapat struktur basis data Pegawai seperti pada Tabel 1 dan data Departemen pada Tabel 6 seperti berikut:

Tabel 6 Ilustrasi Tabel Departemen

id_dep	nama	alamat
1	Sukamaju	Jakarta
2	Sukamakmur	Malang
3	Sukabakti	Blitar

Untuk mendapatkan nama pegawai yang bekerja pada lebih dari satu departemen, maka pengguna akan menuliskan *SQL* seperti berikut :

```
SELECT DISTINCT nama
FROM Departemen D1, Pegawai P2
WHERE D1.id_dep=P2.id_dep
AND D1.id_dep<>P2.id_peg
```

Dalam *database* telah terdapat beberapa pendekatan *SQL* untuk studi kasus terkait yang tampak pada Tabel 7.

Tabel 7 Ilustrasi Perbedaan Pendekatan Pernyataan *SQL*

Pendekatan I	Pendekatan II	Pendekatan III
`SELECT DISTINCT nama FROM Pegawai P1, Pegawai P2 WHERE P1.id_dep=P2.id_dep AND P1.id_peg<>P2.id_peg`	`SELECT Peg.nama FROM Pegawai WHERE Peg.nama IN (SELECT nama FROM Pegawai WHERE id_peg<>Dep.id_peg)`	`SELECT nama FROM Pegawai GROUP BY nama HAVING COUNT(*)>=2`

Berikut adalah langkah dalam mendapatkan PKJ yang mendekati Pernyataan *SQL* Pengguna (PSP).

a. Mendapatkan kata kunci *SQL* yang digunakan di PSP, sehingga dari pernyataan

```
SELECT DISTINCT nama FROM Departemen D1,
Pegawai P2 WHERE D1.id_dep=P2.id_dep AND
D1.id_dep<>P2.id_peg
```

akan diperoleh kata kunci *SQL* seperti berikut:

```
SELECT DISTINCT FROM WHERE AND
```

b. Memberikan bobot pada setiap kata kunci *SQL* yang digunakan di PSP terhadap PKJ. Algoritma yang digunakan yaitu Algoritma *Tf-Idf (Term Frequency – Inverse Document Frequency)* yang intinya akan membobotkan setiap kata *query* dengan PKJ [14]. Rumus algoritma *Tf-Idf* ditunjukkan pada Persamaan (2).

$$W_{ij} = tf_{ij} \times log(\frac{D}{df_j}) \quad \text{.....(2)}$$

Keterangan:

W_{ij} = bobot kata j terhadap kalimat i
tf_{ij} = jumlah kemunculan kata j dalam kalimat i
D = jumlah kalimat kunci dalam *database*
df_j = jumlah kalimat kunci yang terdapat kata j

Persamaan (2) akan dapat berfungsi dengan baik jika tidak terjadi mekanisme *log 0*. Hal ini dikarenakan *log 0* akan menghasilkan nilai yang tidak terdefinisi sehingga akan mempengaruhi nilai pembobotan. Untuk menghindari kasus tersebut, maka Persamaan (2) dimodifikasi menjadi Persamaan (3) sebagai berikut.

$$W_{ij} = tf_{ij} \times log(\frac{D}{df_j} + 1) \quad \text{.....(3)}$$

Dengan keterangan dan konsep yang sama, Persamaan (3) tidak mungkin menghasilkan mekanisme *log 0* seperti Persamaan (2) karena nilai *log* selalu ditambahkan 1 sebelumnya.

Berdasarkan rumus tersebut, proses penghitungan bobot **PKJ 1** adalah sebagai berikut:

- Bobot *select*
  $W_{select} = 1 \times \log(\frac{3}{3} + 1) = 1 \times 0{,}3 = 0{,}3$
- Bobot *distict*
  $W_{distict} = 1 \times \log(\frac{3}{1} + 1) = 1 \times 0{,}6 = 0{,}6$
- Bobot *from*
  $W_{from} = 1 \times \log(\frac{3}{3} + 1) = 1 \times 0{,}3 = 0{,}3$
- Bobot *where*
  $W_{where} = 1 \times \log(\frac{3}{2} + 1) = 1 \times 0{,}4 = 0{,}4$
- Bobot *and*
  $W_{and} = 1 \times \log(\frac{3}{1} + 1) = 1 \times 0{,}6 = 0{,}6$
- Jumlah = 0,3 + 0,6 + 0,3 + 0,4 + 0,6 = 2,2

Proses penghitungan bobot **PKJ 2** adalah sebagai berikut:

- Bobot *select*
  $W_{select} = 2 \times \log(\frac{3}{3} + 1) = 2 \times 0{,}3 = 0{,}6$
- Bobot *distict*
  $W_{distict} = 0 \times \log(\frac{3}{1} + 1) = 0 \times 0{,}6 = 0$
- Bobot *from*
  $W_{from} = 2 \times \log(\frac{3}{3} + 1) = 2 \times 0{,}3 = 0{,}6$
- Bobot *where*
  $W_{where} = 2 \times \log(\frac{3}{2} + 1) = 2 \times 0{,}4 = 0{,}8$
- Bobot *and*
  $W_{and} = 0 \times \log(\frac{3}{1} + 1) = 0 \times 0{,}6 = 0$
- Jumlah = 0,6 + 0 + 0,6 + 0,8 + 0 = 2,0

Proses penghitungan bobot **PKJ 3** adalah sebagai berikut:

- Bobot *select*
  $W_{select} = 1 \times \log(\frac{3}{3} + 1) = 1 \times 0{,}3 = 0{,}3$
- Bobot *distict*
  $W_{distict} = 0 \times \log(\frac{3}{1} + 1) = 0 \times 0{,}6 = 0$
- Bobot *from*
  $W_{from} = 1 \times \log(\frac{3}{3} + 1) = 1 \times 0{,}3 = 0{,}3$
- Bobot *where*
  $W_{where} = 0 \times \log(\frac{3}{2} + 1) = 0 \times 0{,}4 = 0$
- Bobot *and*
  $W_{and} = 0 \times \log(\frac{3}{1} + 1) = 0 \times 0{,}6 = 0$
- Jumlah = 0,3 + 0 + 0,3 + 0 + 0 = 0,6

Setelah diketahui semua bobot, selanjutnya adalah perangkingan nilai setiap bobotnya yaitu bobot PKJ 1 adalah 2,2, bobot PKJ adalah 2,0 dan bobot PKJ 3 adalah 0,6. Dari ketiga nilai diperoleh bahwa bobot PKJ 1 memiliki nilai terbesar, sehingga dalam hal ini PKJ yang dianggap sesuai dengan PSP adalah PKJ 1.

4. Pencocokan PSP dengan PKJ yang dipilih

Pada tahap terakhir terjadi pencocokan antara PSP dengan PKJ yang dipilih. Tahap ini intinya menemukan perbedaan pernyataan yang dibagi menjadi tiga kategori yaitu pernyataan *SELECT*, *FROM* dan *WHERE*. Jika terdapat perbedaan, maka sistem akan memberitahukan kesalahan berdasarkan kategori *SELECT*, *FROM* dan *WHERE*. Sama halnya dengan tahap sebelumnya, proses pencocokan ini menggunakan algoritma *Start End Mid* pada setiap blok utama, yaitu blok *SELECT*, *FROM* dan *WHERE* yang bukan merupakan blok pernyataan bersarang di klausa *WHERE*.

Ilustrasi pencocokan pada blok *SELECT*, *FROM* dan *WHERE* adalah seperti berikut:

- Blok *SELECT*

PSP	PKJ
`DISTINCT nama`	`DISTINCT nama`

- Blok *FROM*

PSP	PKJ
`Departemen` `Pegawai`	`Pegawai` `Pegawai`

- Blok *WHERE*

PSP	PKJ
`id_dep=id_dep AND` `id_dep<>id_peg`	`id_dep=id_dep` `AND id_peg<>id_peg`

Dari perbandingan di atas, ditemukan bahwa terjadi perbedaan atribut pada blok *FROM* dan *WHERE*, sehingga dalam hal ini sistem akan memberikan informasi bahwa terdapat kesalahan pada blok *FROM* dan *WHERE*.

2.3. Rancangan Pengujian

Berdasarkan Gambar 2, pengujian sistem akan dilakukan pada tahap tertentu yaitu sebagai berikut.

1. Pengecekan Kesalahan Sintaksis

Pengujian tidak dilakukan pada tahap ini mengingat berdasarkan penelitian sebelumnya [12] telah dinyatakan bahwa mekanisme pendeteksian kesalahan sintaksis telah valid.

2. Pencocokan Tabel

Tahapan ini akan menghasilkan keterangan dengan dua kemungkinan output, yaitu tabel sesuai dan tabel tidak sesuai. Oleh karena itu, pengujian dilakukan secara berulang-ulang. Kemudian untuk mendapatkan prosentase tingkat ketepatan sistem dalam mencocokkan tabel, digunakan rumus *Precision* [15] seperti Persamaan (4) berikut.

$$Precision = \frac{TP}{(TP + FP)} \times 100\% \quad \text{.....(4)}$$

Dengan kata lain Persamaan (4) memiliki arti seperti Persamaan (5) berikut.

$$Precision = \frac{jumlah\ pencocokan\ sesuai\ (yang\ ditemukan\ sistem)}{jumlah\ pencocokan\ dianggap\ sesuai\ (yang\ ditemukan\ sistem)} \times 100\% \quad \text{....(5)}$$

Selain itu, untuk mendapatkan prosentase tingkat keberhasilan sistem dalam mencocokkan tabel, digunakan rumus Recall [15] seperti Persamaan (6) berikut.

$$Recall = \frac{TP}{(TP + FN)} \times 100\% \quad \text{.....(6)}$$

Persamaan (6) memiliki arti seperti Persamaan (7) berikut.

$$Recall = \frac{jumlah\ pencocokan\ sesuai\ (yang\ ditemukan\ sistem)}{jumlah\ pencocokan\ sesuai\ (yang\ sebenarnya)} \times 100\% \quad \text{.....(7)}$$

3. Pemilihan Jawaban.

Proses pemilihan jawaban akan menghasilkan satu pendekatan kunci yang sesuai dengan jawaban pengguna. Oleh karena itu, untuk mendapatkan prosentase ketepatan sistem dalam pemilihan kunci dilakukan pengujian secara berulang-ulang dan digunakan rumus *Precission* dan *Recall* [15] seperti pada Persamaan (4) dan (6) dimana masing masing persamaan diterjemahkan ke Persamaan (8) dan (9) berikut.

$$Precision = \frac{jumlah\ pendekatan\ sesuai\ (yang\ ditemukan\ sistem)}{jumlah\ pendekatan\ dianggap\ sesuai\ (yang\ ditemukan\ sistem)} \times 100\% \quad \text{.....(8)}$$

$$Recall = \frac{jumlah\ pendekatan\ sesuai\ (yang\ ditemukan\ sistem)}{jumlah\ pendekatan\ sesuai\ (yang\ sebenarnya)} \times 100\% \quad \text{.....(9)}$$

4. Pencocokan *SQL*

Pada tahap terakhir, akan didapatkan perbedaan atribut antara *query* kunci dengan *query* jawaban pengguna. Oleh karena itu, pengujian dilakukan secara berulang-ulang untuk kemudian didapatkan nilai rata-rata *Precision* dan *Recall* yang konsepnya sama dengan Persamaan (4) dan (6) dan dapat diterjemahkan sebagai Persamaan (10) dan (11) berikut.

$$Precision = \frac{jumlah\ perbedaan\ benar\ (yang\ ditemukan\ sistem)}{jumlah\ perbedaan\ ditemukan\ (yang\ ditemukan\ sistem)} \times 100\% \quad \text{.....(10)}$$

$$Recall = \frac{jumlah\ perbedaan\ benar\ (yang\ ditemukan\ sistem)}{jumlah\ perbedaan\ (yang\ sebenarnya)} \times 100\% \quad \text{.....(11)}$$

Dari keempat tahapan tersebut akan didapatkan nilai rata-rata dari *Precision* dan *Recall.* Nilai *Precision* akan menunjukkan tingkat prosentase ketepatan sistem dalam mendeteksi kesalahan logika *SQL*. Sedangkan nilai *Recall* akan menunjukkan tingkat prosentase keberhasilan sistem dalam mendeteksi kesalahan logika *SQL*.

3. Kesimpulan

Berdasarkan hasil dan pembahasan, dapat disimpulkan bahwa:

1. Kesalahan logika pada *SQL* dapat diketahui dengan menggunakan algoritma *Start End Mid.*

2. Pada tahap selanjutnya akan dibuat desain sistem secara lebih detail dan dilanjutkan pada tahap pengembangan sistem.

Daftar Pustaka

[1] Fanani, Ach. Chanifuddin. 2013. Pengembangan Sumber Belajar SQL Berbasis Web untuk Matakuliah Basis Data Prodi S1 PTI Universitas Negeri Malang. Skripsi tidak diterbitkan. Malang: Fakultas Teknik UM.

[2] Goldberg, Christian. 2009. *Do You Know SQL? About Semantic Errors In Database Queries*. Pada British National Conference on Databases, diakses 24 April 2015.

[3] Dollinger, Robert. 2013. *SQL Lightweight Tutoring Module – Semantic Analysis of SQL Queries based on XML Representation and LINQ*. Pada Proceedings of World Conference on Educational Multimedia, Hypermedia & Telecommunications, diakses 24 April 2015.

[4] Brass, Stefan & Goldberg, Christian. 2006. *Semantic Errors in SQL Queries: A Quite Complete List.* Pada International Conference on Quality Software, 197-204, diakses 24 April 2015.

[5] Elmasri, Ramez & Navathe, Shamkant. 2011. *Fundementals of Database System (6th edition)*. United States of America: Pearson Education Inc.

[6] Prasetyo, R. Edy, Suhartono, dkk. 2012 . *Alat Bantu Penelusuran String pada Materi dengan Menggunakan Algoritma Boyer-Moore.* (Online), (http://research.pps.dinus.ac.id/lib/jurnal/Vol%2008.1%20040-048.pdf), diakses 27 maret 2015.

[7] Wibawa, Aji P., dkk. 2013. *Indonesian to Javanese Machine Translation*. Pada International Journal of Innovation, Management and Technology, Vol 04 No 04, diakses 10 Juni 2015.

[8] Fernando, Hary. 2009. *Perbandingan dan Pengujian Beberapa Algoritma Pencocokan String.* (Online), (http://webmail.informatika.org/~rinaldi/Stmik/2009-2010/Makalah2009/MakalahIF3051-2009-006.pdf), diakses 20 Maret 2015.

[9] Kumara, Gozali Harda. 2008. *Visualisasi Beberapa Algoritma Pencocokan String Dengan Java*. (Online), (http://informatika.stei.itb.ac.id/~rinaldi.munir/ TA/Makalah_TA%20Gozali.pdf), diakses 20 Maret 2015.

[10] Abdeen, Rawan Ali. 2011. *An Algorithm for String Searching Based on Brute-Force Algorithm.* Pada International Journal of Computer Science and Network Security, Vol.11 No.7, diakses 23 April 2015.

[11] Mitrovic, Antonija. 1998. *A Knowledge-Based Teaching System for SQL*. (Online), (https://nats-www.informatik.uni-hamburg.de/pub/INCOM/ Literatur/*SQL*-Tutor.pdf), diakses 24 April 2015

[12] Suriawan, Tofa. 2015. Pengembangan Media Pembelajaran Berbasis Web pada Mata Pelajaran Basis Data untuk Siswa Kelas XI RPL SMK Negeri 5 Malang. Skripsi tidak diterbitkan. Malang: Fakultas Teknik UM.

[13] Intan, R., & Defeng, A. (2006). *Hard: Subject-Based Search Engine Menggunakan Tf-Idf Dan Jaccard S Coefficient*. Jurnal Teknik Industri, 8(1), pp-61.

[14] Fitri, Meisya. 2012. Perancangan Sistem Temu Balik Informasi dengan Metode Pembobotan Kombinasi Tf-Idf untuk Pencarian Dokumen Berbahasa Indonesia. (Online), (http://download.portalgaruda.org/article.php?article=112004&val=2313), diakses 12 Mei 2015.

[15] Syauqi, A'la & Aeny Nurwahdah. 2015. *Sistem Tanya Jawab dengan Web Semantik.* Seminar Nasional Aplikasi Teknologi Informasi (SNATI).

Biodata Penulis

Jevri Tri Ardiansah, mahasiswa tingkat akhir Jurusan Teknik Elektro Universitas Negeri Malang.

Aji Prasetya Wibawa, memperoleh gelar Sarjana Teknik (S.T), Jurusan Teknik Elektro Universitas Brawijaya, lulus tahun 2004. Memperoleh gelar Magister Menejemen Teknologi (M.M.T) Program Pasca Sarjana Magister Menegemen dan Teknologi Informasi Institute Teknologi Sepuluh Nopember, lulus tahun 2007. Memperoleh gelar Doctor of Philosophy (PhD) Program School of Engineering, University of South Australia, lulus tahun 2010. Saat ini menjadi Dosen di Universitas Negeri Malang sekaligus anggota dari Institute of Electrical and Electronics Engineers (IEEE).

Triyanna Widiyaningtyas, memperoleh gelar Sarjana Teknik (S.T), Jurusan Teknik Telekomunikasi STT Telkom Bandung, lulus tahun 1998. Memperoleh gelar Magister Teknik (M.T) Program Pasca Sarjana Magister Sistem Komputer dan Informatika Universitas Gadjah Mada Yogyakarta, lulus tahun 2002. Saat ini menjadi Dosen di Universitas Negeri Malang.