BAB III

ANALISA DAN PERANCANGAN SISTEM

3.1 Analisis Sistem

Tahap analisis sistem dilakukan sebelum tahap perancangan sistem. Tahap analisis sistem merupakan tahap yang kritis dan sangat penting karena kesalahan di dalam tahap ini akan menyebabkan kesalahan pada tahap selanjutnya. Proses analisis sistem merupakan suatu prosedur yang dilakukan untuk pemeriksaan masalah dan penyusunan alternatif pemecahan masalah yang timbul serta membuat spesifikasi sistem yang baru atau sistem yang akan diusulkan dan dimodifikasi. Hasil akhir dari tahap analisis sistem adalah suatu laporan yang dapat menggambarkan sistem yang telah dipelajari dan diketahui bentuk permasalahannya serta rancangan sistem baru yang akan dikembangkan.

3.1.1 Analisis Masalah

Salah satu kendala dalam proses pengolahan data dari *database* yang tersebar adalah proses penggabungan data dari *database* yang berbeda. Tidak semua DBMS memiliki fitur *database link* seperti pada DBMS Oracle sehingga dalam proses penggabungan data akan mengalami kendala dalam hal akses data antar DBMS yang berbeda.

Proses penggabungan data akan lebih mudah jika data yang akan digabungkan mempunyai format data yang sama. Karena hal ini merupakan syarat dilakukan proses penggabungan data.

Yang dimaksud dengan format data yang sama ini adalah sebagai berikut :

1. Memiliki jumlah *field* atau kolom yang sama
2. Memiliki nama-nama *field* atau kolom yang sama.
3. Memiliki tipe data dan ukuran yang sama di setiap *field* atau kolomnya.

Dalam kenyataannya tidak semua data yang akan digabungkan pasti memiliki format data yang sama. Informasi yang akan diambil tidak hanya berasal dari satu tabel, tetapi dapat juga berasal dari banyak tabel, maka harus dilakukan *query* dahulu ke dalam masing-masing *database* yang digunakan. Hasil dari *query* tersebut juga harus memiliki format data yang sama dengan data (hasil *query*) yang lain jika akan dilakukan penggabungan data.

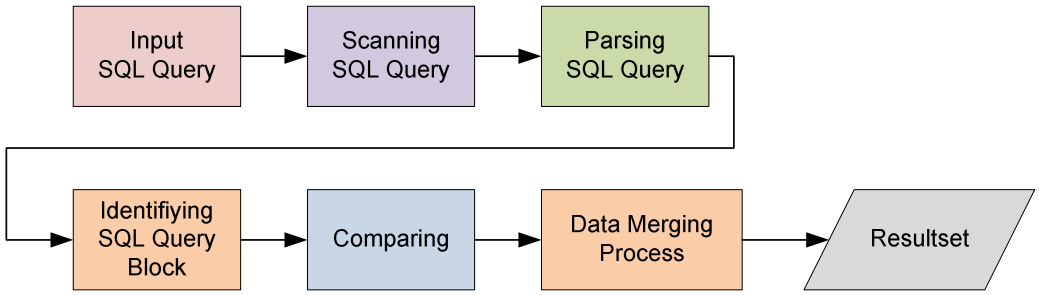
Pada umumnya cara yang sering dipakai adalah dengan membuat *spool* data tersebut ke dalam *text file* dengan *delimiter* tertentu. Setelah itu semua *text file* yang ada digabungkan dengan menggunakan *text editor* atau menggunakan *command prompt* menjadi satu *text file*. Kemudian data gabungan di dalam *file* tersebut diolah menjadi sebuah informasi.

Dari permasalahan yang sudah dijelaskan sebelumnya maka dapat disimpulkan beberapa kendala yang ada, diantaranya sebagai berikut:

1. Tidak ada aplikasi yang mempunyai fungsi untuk melakukan koneksi dan menghubungkan *database* dengan DBMS yang berbeda.
2. Data yang akan digabungkan tidak selalu memiliki format data yang sama.
3. Proses penggabungan data dilakukan secara manual dengan mengolah data *textfile* hasil dari data yang diambil dari *database* yang berbeda.

3.1.2 Blok Diagram Model Aplikasi

Aplikasi yang akan dibangun dimaksudkan untuk menyelesaikan permasalahan dalam penggabungan data dari *database* yang berbeda. Secara umum dapat dimodelkan dalam *block diagram* berikut ini.



Gambar 3.1 *Block diagram* model aplikasi

Block Diagram pada gambar 3.1 di atas adalah gambaran umum model aplikasi. Aplikasi berbasis web yang akan dibangun memiliki fungsi utama yaitu melakukan penggabungan data antar *database* dengan DBMS yang berbeda. Pengabungan data yang dimaksud adalah dengan menggunakan fungsi set operasi atau *operation set* (*Union, Intersect, Minus)* antar tabel dalam *database* dengan DBMS yang berbeda dalam satu perintah atau sintaks SQL *query*.

Penjelasan dari *block diagram* di atas adalah sebagai berikut :

a. *Input* SQL *Query*

- SQL *query* dipakai sebagai masukan aplikasi.
- *Query* data dipakai sebagai metode untuk mengambil data dari masing-masing *database*.
- Format *query* yang diterima oleh aplikasi ini sesuai dengan aturan produksi yang ada dalam aplikasi ini.

- Sintaks blok SQL *query* yang diterima adalah SQL *query* yang diterima oleh masing-masing DBMS yang diakses.

b. *Scanning* SQL *Query*

Proses *scanning* (analisis leksikal) melakukan pemeriksaan terhadap SQL *query* dengan cara membaca satu per satu karakter yang ada pada kode sumber tersebut, kemudian dikelompokkan menjadi token/leksik yang mempunyai arti tertentu.

c. *Parsing* SQL *Query*

Proses *parsing* (analisis sintaksis) memeriksa kebenaran sintaks dari SQL *query* dan menangani kesalahan sintaks berdasarkan tata bahasa (*grammar*) SQL *query*. Proses ini dilakukan oleh masing-masing DBMS yang diakses.

d. *Identifying* SQL *Query Block*

Proses ini mengidentifikasi elemen-elemen pada SQL *query*, seperti jumlah kolom, nama kolom, alias kolom, nama *table*, dan alias *table*. Kolom-kolom yang didefinisikan dapat juga berupa fungsi yang memiliki tipe data keluaran tertentu sesuai dengan daftar fungsi-fungsi yang ada dalam SQL *Query*.

e. *Comparing* (Pembandingan format data).

- Validasi data dilakukan untuk mengecek nama kolom dan tipe data kolom yang didefinisikan dalam SQL *query*. Nama kolom yang dipakai adalah nama-nama kolom yang didefinisikan dalam blok SQL *query* yang pertama.
- Jika format data tidak sama, aplikasi akan menghasilkan pesan error.

- Jika format data sama, aplikasi akan melakukan penggabungan data berdasarkan fungsi yang didefinisikan oleh pengguna (*Union, Intersect*, atau *Minus*) dalam SQL *Query*.

f. *Data Merging Process* (*Union, Intersect,* atau *Minus*).

- Fungsi ini dapat dieksekusi jika format data yang akan digabungkan memiliki format yang sama.
- Hasil dari penggabungan data dapat disimpan dalam teks file atau disimpan dalam sebuah tabel di dalam *database*.

3.1.3 Analisis Proses *Parsing* pada Masukan SQL *Query*

Proses *parsing* terhadap inputan SQL *query* ini terdari dari dua bagian. Bagian pertama adalah yang menggabungkan karakter demi karakter untuk membuat token (biasanya dilakukan oleh bagian yang disebut *scanner* atau *lexer*), dan bagian kedua adalah yang menentukan apakah token-token tersebut memenuhi *grammar* (dilakukan oleh bagian yang disebut *parser*).

Analisis leksikal (*scanner*) melakukan pemeriksaan terhadap SQL *query* dengan cara membaca satu per satu karakter yang ada pada kode sumber tersebut, kemudian dikelompokkan menjadi token atau leksik yang mempunyai arti tertentu. *Scanner* berperan sebagai antar muka antara *source code* dengan proses analisis sintaksis (*parser*).

Analisis sintaksis (*parser*) menerima masukan dari *scanner* (dalam bentuk token) dan membentuk *parse tree* sesuai dengan sintaks dan tata bahasanya. Dengan kata lain *parser* memeriksa kebenaran sintaks dari SQL *query* dan menangani kesalahan sintaks.

3.1.3.1 Aturan produksi sintaks dan Diagram sintaks

Aturan produksi sintaks pada SQL *query* ditulis dalam format BNF (*Backus Nour Form*). Aturan produksi ini mengacu pada *Final Committee Draft* (FCD) *of* ISO/IEC 9075-2:2003. Dalam penelitian ini, aturan produksi yang dipakai dapat dilihat pada table 3.1 di halaman ini. Aturan produksi ini dipakai untuk memeriksa kebenaran sintaks. Untuk sintaks *query* ke *database* akan ditangani oleh masing-masing DBMS yang diakses.

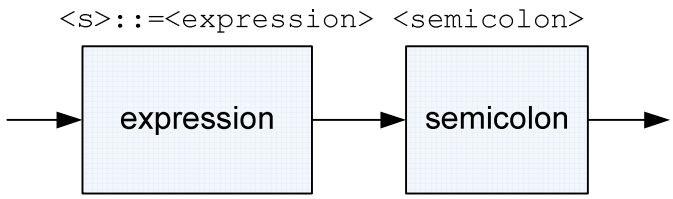
Aturan produksi yang dimaksud dapat dilihat pada table 3.1 berikut ini:

Tabel 3.1 Tabel BNF SQL *Query*

1	<s>	::=	<expression><semicolon>
2	<expression>	::=	<term> | <expression> <set_operator> <term>
3	<term>	::=	<lroundbrackets> <expression> <rroundbrackets> | <expression> | <*squery*>
4	<*squery*>	::=	<dbidentifier> <lsquarebrackets> <rsquarebrackets>
5	<dbidentifier>	::=	**'db-'**<alphabet> { <alphabet> | <numeral> }
6	<set_operator>	::=	**union'** [**'all'**] | **'intersect'** | **'minus'**
7	<alphabet>	::=	**'a..z'**
8	<numeral>	::=	**'0..9'**
9	<lsquarebrackets>	::=	**'['**
10	<rsquarebrackets>	::=	**']'**
11	<lroundbrackets>	::=	**'('**
12	<rroundbrackets>	::=	**')'**
13	<semicolon>	::=	**';'**

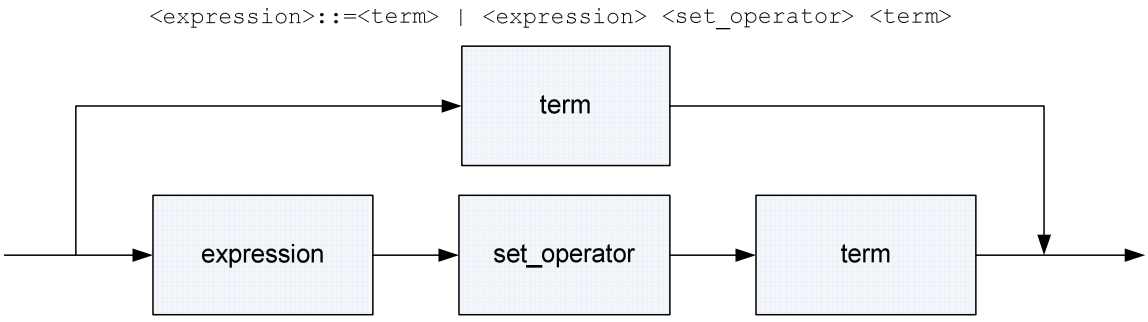
Diagram sintaks memberikan gambaran yang jelas mengenai BNF yang telah dirancang dalam bentuk grafis. Masing-masing diagram sintaks memudahkan pembaca untuk melihat masing-masing aturan produksi yang ada. Diagram-diagram sintaks pada gambar 3.2 sampai dengan gambar 3.14 berikut ini menggambarkan BNF (13 aturan produksi) yang telah dirancang pada tabel 3.1 di halaman 42.

Simbol <s> pada gambar 3.2 adalah sebuah simbol variabel (simbol non terminal) yang menghasilkan urutan variable <expression><semicolon>.



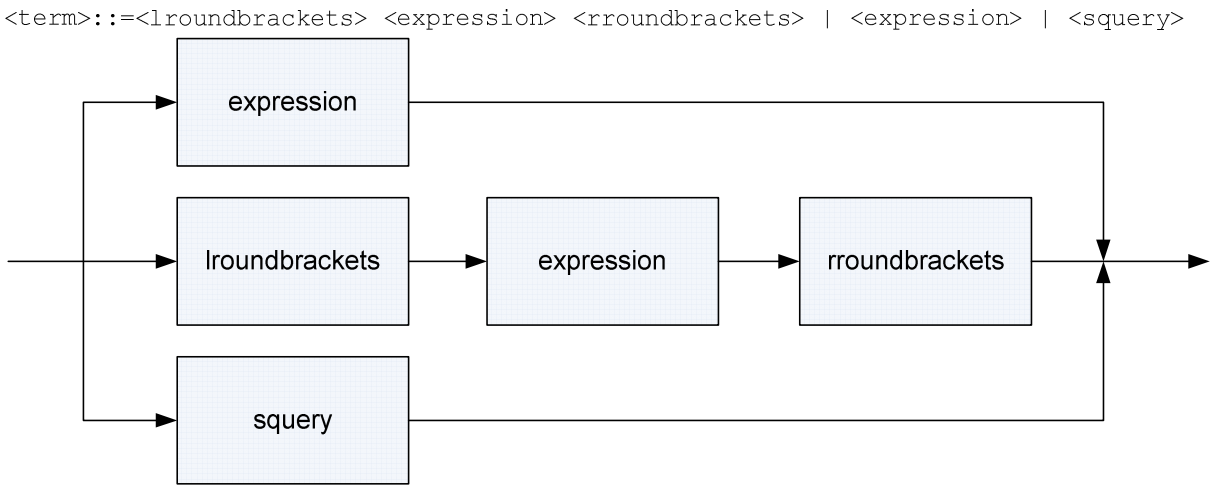
Gambar 3.2 Diagram sintaks *S*

Simbol <expression> pada gambar 3.3 adalah sebuah simbol variabel (simbol non terminal) yang menghasilkan variable <term>. Variabel <expression> juga menghasilkan urutan variable <expression><set_operator><term>.



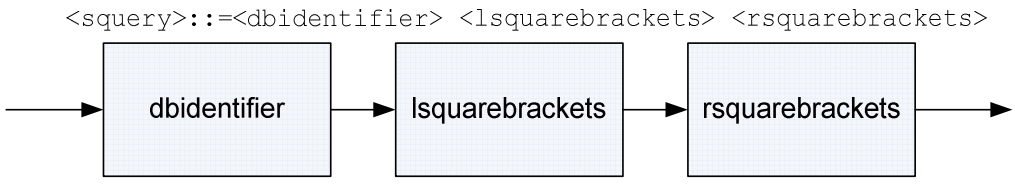
Gambar 3.3 Diagram sintaks *EXPRESSION*

Simbol <term> pada gambar 3.4 adalah sebuah simbol variabel (simbol non terminal) yang menghasilkan variabel <expression> atau <s*query*> atau urutan variable <lroundbrackets><expression><rroundbrackets>.



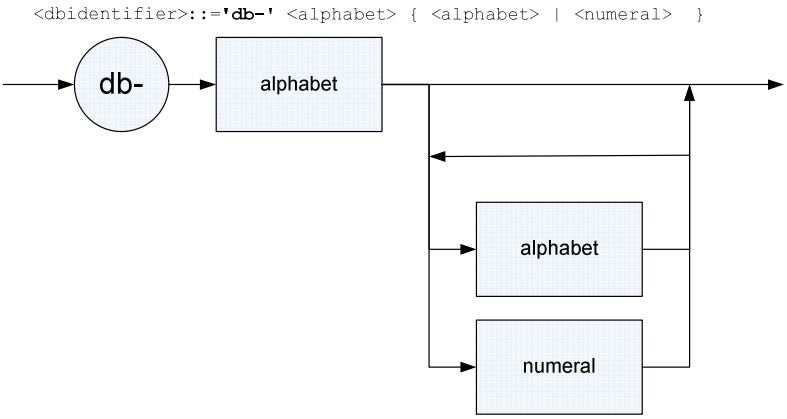
Gambar 3.4 Diagram sintaks *TERM*

Simbol <s*query*> pada gambar 3.5 adalah sebuah simbol variabel (simbol non terminal) yang menghasilkan urutan variabel <dbidentifier> <lsquarebrackets> <rsquarebrackets>. SQL *query* ke masing-masing DBMS diletakkan diantara variable <lsquarebrackets> dan <rsquarebrackets>. SQL *query* tidak diparsing oleh aplikasi ini, akan tetapi diparsing oleh masing-masing DBMS yang dipanggil sesuai dengan variabel <dbidentifier> yang didefinisikan.



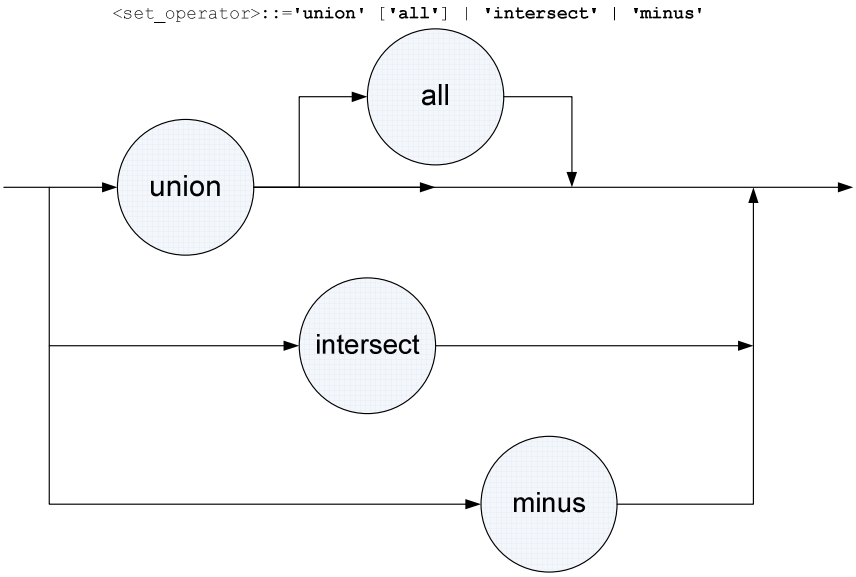
Gambar 3.5 Diagram sintaks *SQUERY*

Simbol <dbidentifier> pada gambar 3.6 adalah sebuah simbol variabel (simbol non terminal) yang menghasilkan urutan terminal ‘db-’ (urutan karakter d, b, dan -) variabel <alphabet> diikuti iterasi variable <alphabet> atau <numeral>. Contoh *string* yang diterima oleh variabel <dbidentifier> : db-best, db-ora90, db-mssql2011, dsb.



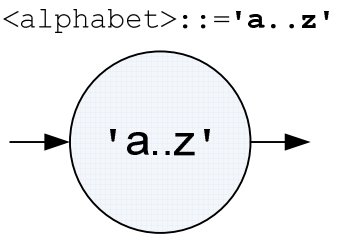
Gambar 3.6 Diagram sintaks *DBIDENTIFIER*

Simbol <set_operator> pada gambar 3.7 adalah sebuah simbol variabel (simbol non terminal) yang menghasilkan terminal ‘union’ atau ‘union all’ atau ‘intersect’ atau ‘minus’.



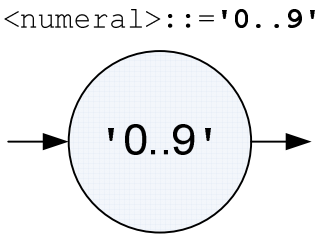
Gambar 3.7 Diagram sintaks *SET_OPERATOR*

Simbol <alphabet> pada gambar 3.8 adalah sebuah simbol variabel (simbol non terminal) yang menghasilkan terminal karakter alphabet ‘a’ sampai dengan ‘z’.



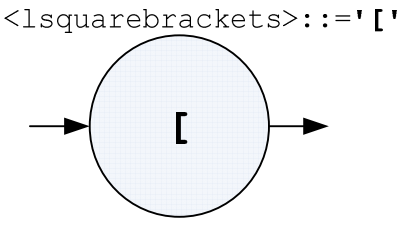
Gambar 3.8 Diagram sintaks *ALPHABET*

Simbol <numeral> pada gambar 3.9 adalah sebuah simbol variabel (simbol non terminal) yang menghasilkan terminal karakter number ‘0’ sampai dengan ‘9’.



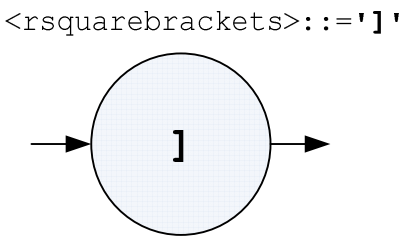
Gambar 3.9 Diagram sintaks *NUMERAL*

Simbol <lsquarebrackets> pada gambar 3.10 adalah sebuah simbol variabel (simbol non terminal) yang menghasilkan terminal karakter ‘[’ (kurung siku buka).



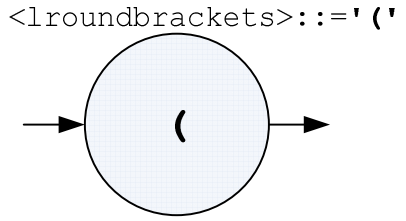
Gambar 3.10 Diagram sintaks *LSQUAREBRACKETS*

Simbol <rsquarebrackets> pada gambar 3.11 adalah sebuah simbol variabel (simbol non terminal) yang menghasilkan terminal karakter ‘]’ (kurung siku tutup).



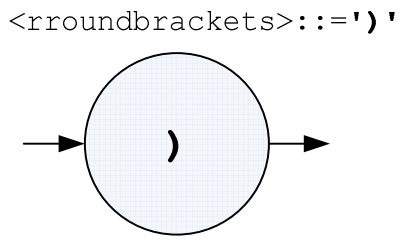
Gambar 3.11 Diagram sintaks *RSQUAREBRACKETS*

Simbol <lroundbrackets> pada gambar 3.12 adalah sebuah simbol variabel (simbol non terminal) yang menghasilkan terminal karakter ‘(’ (kurung buka).



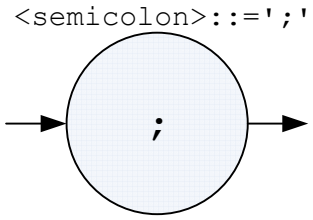
Gambar 3.12 Diagram sintaks *LROUNDBRACKETS*

Simbol <rroundbrackets> pada gambar 3.13 adalah sebuah simbol variabel (simbol non terminal) yang menghasilkan terminal karakter ‘(’ (kurung tutup).



Gambar 3.13 Diagram sintaks *RROUNDBRACKETS*

Simbol <semicolon> pada gambar 3.14 adalah sebuah simbol variabel (simbol non terminal) yang menghasilkan terminal karakter ‘;’ (titik koma). Variable ini menandakan akhir dari *string* SQL *query* yang diterima oleh aplikasi ini.



Gambar 3.14 Diagram sintaks *SEMICOLON*

3.1.3.2 Diagram FSA (*Finite State Automata*) untuk analisis leksikal (*scanner*)

Pada sub bab sebelumnya sudah dijelaskan mengenai aturan produksi dan diagram sintaks pada SQL *query*. Aturan produksi tersebut digunakan sebagai pedoman dalam proses *parsing* pada saat proses penggabungan data dilakukan. Proses penggabungan data dapat dilakukan jika SQL *query* yang diberikan oleh pengguna sudah sesuai dengan aturan produksi tersebut. Aturan produksi tersebut menggunakan tata bahasa bebas konteks. Proses *parsing* merupakan tahapan yang berfungsi untuk memeriksa urutan kemunculan token.

Token dihasilkan oleh *parser* (yang melakukan proses *scanning*) dengan cara menguraikan kode sumber (*source code*) menjadi unit-unit kecil yang mempunyai arti. Proses *parsing* (analisis leksikal) lebih mudah diimplementasikan pada FSM (*Finite State Machine*) atau FSA (*Finite State Automata*). FSA bukan merupakan sebuah mesin fisik tetapi merupakan model

matematika dari suatu sistem yang menerima masukan dan mengeluarkan keluaran dalam bentuk diskrit. FSA terdiri dari sejumlah *state* berhingga.

Pada gambar 3.15 dapat dilihat sebuah *state* diagram dalam bentuk DFA (Non-*Deterministic State Automata*) yang menggambarkan proses *scanning* dalam penelitian ini. FSA dalam bentuk DFA akan memudahkan pembuatan algoritma pada tahap implementasi nantinya karena dalam DFA dari suatu *state* ada tepat satu *state* berikutnya untuk setiap simbol masukan.

Tuple yang ada dalam gambar 3.15 adalah sebagai berikut :

Q = {q0,q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,q11,q12,q13,q14,q16,q17,q18,q19,q20,
q21,q22,q23,q24,q25,q26,q27,q28,q29,q30,q31,q32,q33,q34,q35,q36,q37,q3
8,q39,q40,q41,q42,q43,q44,q45,q46,q47,q48,q49,q50,q51,q52,q53,q54,q55,
q56,q57,q58}

∑ = { '-','(',')',';','[',']',0,1,2,3,4,5,6,7,8,9,
a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z}

S = q0

F = q15

Keterangan simbol yang dipakai sebagai simbol masukan dapat dilihat pada tabel 3.2 berikut ini.

Tabel 3.2 Keterangan simbol masukan

simbol input	**keterangan**
;	titik koma
(	Tanda kurung buka
)	Tanda kurung tutup
-	Tanda kurang
[	Tanda kurung siku buka
]	Tanda kurung siku tutup

Tabel transisi untuk DFA pada gambar 3.15 dapat dilihat pada tabel 3.3 dan tabel 3.4 berikut ini.

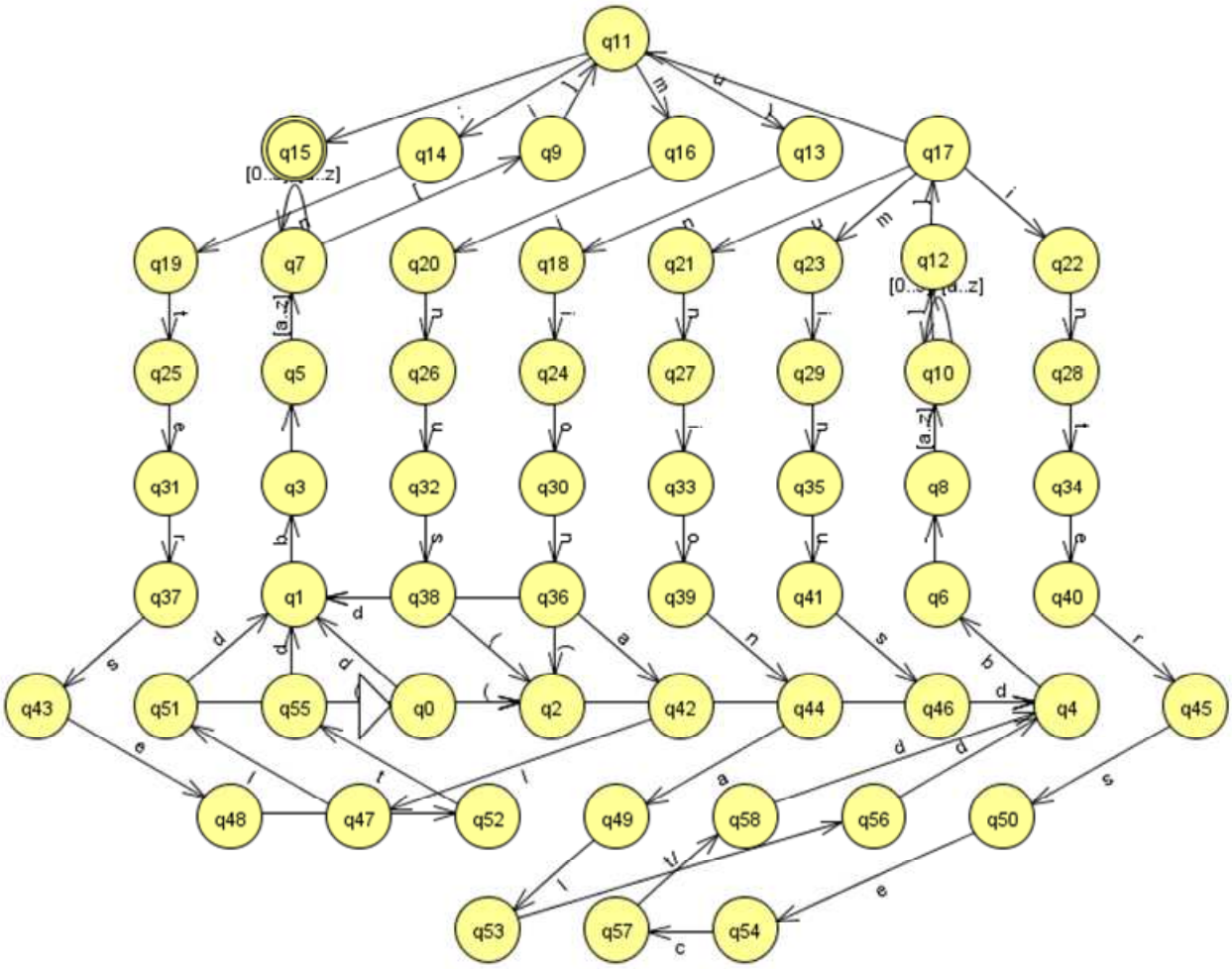
Tabel 3.3 Tabel transisi DFA (bagian 1)

δ	-	(	)	;	[	]	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e
q0		q2																		q1	
q1																		q3			
q2																				q4	
q3	q5																				
q4																		q6			
q5																	q7	q7	q7	q7	q7
q6	q8																				
q7					q9		q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7
q8																	q10	q10	q10	q10	q10
q9						q11															
q10					q12		q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10
q11				q15																	
q12						q17															
q13																					
q14																					
q16																					
q17			q11																		
q18																					
q19																					
q20																					
q21																					
q22																					
q23																					
q24																					
q25																					q31
q26																					
q27																					
q28																					
q29																					
q30																					
q31																					
q32																					
q33																					
q34																					q40
q35																					
q36		q2															q42			q1	
q37																					
q38		q2																		q1	
q39																					
q40																					
q41																					
q42																					
q43																					q48
q44																	q49			q4	
q45																					
q46																				q4	
q47																					
q48																			q52		
q49																					
q50																					q54
q51		q2																		q1	
q52																					
q53																					
q54																			q57		
q55		q2																		q1	
q56																				q4	
q57																					
q58																				q4	

Tabel 3.4 Tabel transisi DFA (bagian 2)

δ	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
q0																					
q1																					
q2																					
q3																					
q4																					
q5	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7
q6																					
q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7	q7
q8	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10
q9																					
q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10	q10
q11				q14				q16								q13					
q12																					
q13									q18												
q14									q19												
q16				q20																	
q17				q22				q23								q21					
q18				q24																	
q19															q25						
q20									q26												
q21									q27												
q22									q28												
q23				q29																	
q24										q30											
q25																					
q26																q32					
q27				q33																	
q28															q34						
q29									q35												
q30									q36												
q31													q37								
q32														q38							
q33										q39											
q34																					
q35																q41					
q36																					
q37														q43							
q38																					
q39									q44												
q40													q45								
q41														q46							
q42							q47														
q43																					
q44																					
q45														q50							
q46																					
q47							q51														
q48																					
q49							q53														
q50																					
q51																					
q52															q55						
q53							q56														
q54																					
q55																					
q56																					
q57															q58						
q58																					

Gambar berikut ini adalah sebuah diagram DFA yang dipakai sebagai acuan untuk melakukan proses *scanning*.



Gambar 3.15 DFA untuk *scanner* pada aplikasi *SODA Data Merger*

3.1.3.3 Metode *parsing*

Terdapat dua buah metode *parsing* yaitu *Top-Down Parsing* dan *Bottom-Up Parsing*. Metode *Top-Down Parsing* melakukan penelusuran dari puncak menuju daun. LL *parser* merupakan *top-down parser*, yang akan membaca masukan dari kiri ke kanan dan melakukan *parsing* secara penurunan terkiri (*left-most derivation*). Kelas *grammar* yang bisa diturunkan dengan menggunakan LL *parser* dinamakan LL *grammar*. LL *grammar* tidak boleh mengandung rekursif

kiri (*left recursive*), contoh : A → Abc. LL *grammar* tidak boleh mengandung faktorisasi kiri (*left factorization*), yaitu dalam sebuah aturan produksi yang memiliki ruas kiri yang sama tidak boleh ada alternatif penurunan (ruas kanan) yang diawali simbol yang sama, contoh : B → a | aC.

Metode *Bottom-Up Parsing* melakukan penelusuran dari daun menuju puncak. Metode ini lebih *general* dari metode *Top-Down Parsing* dan dapat menangani aturan produksi yang mengandung rekursif kiri. Hampir semua kelas bahasa CFG (*Context Free Grammar*) dapat diselesaikan oleh metode ini. Metode ini akan mereduksi *string* masukan menjadi simbol awal dengan menggunakan aturan produksi yang ada. *Parser* yang digunakan dalam metode ini adalah LR *Parser*. LR *Parser* akan membaca input dari kiri ke kanan dan melakukan *parsing* secara penurunan terkanan (*right-most derivation*). Kelas *grammar* yang bisa diturunkan dengan menggunakan LR *parser* dinamakan LR *grammar.* Aturan produksi yang memenuhi bentuk CFG merupakan LR *grammar*. Metode ini akan mereduksi *string* masukan (*input*) menjadi simbol awal dengan menggunakan aturan produksi yang ada. Dalam metode *Bottom-Up Parsing* terdapat dua buah aksi yang dilakukan dalam proses *parsing*, yaitu *Shift* (membaca *string* masukan berikutnya) dan *Reduce* (mengubah *string* menjadi simbol variabel yang dapat menurunkan *string* tersebut). Hasil akhir dari metode ini dapat berupa *accepted* (jika *string* masukan sudah habis dibaca dan *state* mampu mencapai *state* awal) atau *error* (jika *string* masukan sudah habis dibaca tetapi tidak dapat mencapai *state* awal, atau *state* awal mampu dicapai tapi masih ada *string* masukan yang tersisa).

3.1.3.4 Proses *scanning* dan *parsing* yang dilakukan dalam aplikasi ini

State diagram DFA dan aturan produksi BNF yang digunakan dalam proses *scanning* dan *parsing* mempunyai tujuan yang sama yaitu memeriksa kebenaran sintaks pada *string* masukan. Dalam penelitian ini, kebenaran sintaks ditangani di dalam proses *scanning* yaitu dengan memeriksa apakah *string* sudah sesuai dengan state diagram DFA atau tidak. Hasil proses *scanning* berupa token *dbidentifier* dan *block* SQL *query* akan dibaca, diidentifikasi dan dieksekusi.

Pada Tabel BNF SQL Query pada tabel 3.1 di halaman 45, variabel <set_operator> dapat kita sebut sebagai sebuah *operator* dan variabel <expression> dapat kita sebut sebagai sebuah *operand*. *Scanner* akan mengidentifikasi *string* masukan SQL *query* ke dalam variabel <set_operator> dan <expression>. Sehingga menghasilkan suatu format dengan urutan tertentu yang akan dijalankan oleh aplikasi ini.

Diagram DFA yang digunakan sebagai acuan dalam proses *scanning* dapat dilihat pada gambar 3.15 di halaman 56. Untuk memastikan bahwa DFA tersebut sudah benar sesuai dengan kebutuhan untuk melakukan *scanning* pada SQL *query*, maka berikut ini adalah pembuktian dengan melakukan penelusuran *string* SQL *query* ke dalam DFA tersebut.

SQL *Query* :

db-alias1[select x.kolom1as col1 from tablex x]
union
(db-alias2[select y.kolom2 from tabley y]
minus
db-alias3[select z.kolom3 as col1 from tablez z]);

Hasil *tracing* dari SQL Query di atas dapat dilihat pada table 3.5 berikut ini. Sebelum aplikasi melakukan proses *scanning*, *string* masukan SQL *query* tersebut harus diubah menjadi huruf kecil (menggunakan fungsi *strtolower* pada php) dan membagi *string* masukan per baris menjadi *string-string* dan disimpan ke dalam sebuah *array variable*. Sehingga *string* masukan tersebut (dalam *array variable*) menjadi:

Array[0]: **db-alias1[select x.kolom1 as col1 from tablex x]**
Array[1]: **union**
Array[2]: **(db-alias2[select y.kolom2 from tabley y]**
Array[3]: **minus**
Array[4]**: db-alias3[select z.kolom3 as col1 from tablez z]);**

Variabel tersebut digunakan agar setiap ada kesalahan sintaks, aplikasi dapat menunjukkan letak kesalahan dengan jelas yaitu pada baris tertentu yang terjadi kesalahan.

Proses *scanning* dilakukan pada tiap-tiap karakter mulai dari baris pertama sampai dengan baris terakhir. Ada beberapa kondisi dimana karakter digolongkan ke dalam golongan sintaks dan ke dalam golongan *block* SQL *query*.

Kriteria masing-masing golongan tersebut adalah sebagai berikut:

1. Golongan sintaks.
    - Karakter-karakter pada golongan ini adalah karakter yang akan diperiksa apakah sesuai dengan DFA atau tidak.
    - Karakter-karakter ini berada di posisi sebelum karakter ‘[‘ dan berada di posisi setelah karakter ‘]’. Dengan catatan bahwa karakter ‘[’ dan ‘]’ bukan merupakan bagian dari sebuah *string* yang diapit oleh dua buah tanda kutip satu (‘).

- Menghasilkan sebuah token yang disebut *dbidentifier* yang berarti sebuah alias *database* yang didefinisikan dalam inputan SQL *query*. Setiap menemukan karakter ‘]’ (dengan catatan bahwa karakter tersebut bukan bagian dari sebuah *string* yang berada di antara simbol kutip), proses *scanning* akan menghasilkan sebuah token *dbidentifier* dan sebuah *block* SQL *query*.
- Setiap token *dbidentifier* yang ditemukan akan dicek apakah sudah didefinisikan dalam aplikasi atau belum. Jika belum ada maka proses *scanning* akan menghasilkan pesan *error* dan menunjukkan letak kesalahan tersebut.

2. Golongan *block* SQL *query*.
    - Karakter-karakter pada golongan ini adalah serangkaian karakter yang menjadi sebuah SQL *query*. Karakter pada golongan ini tidak diperiksa atau dilakukan proses *parsing* oleh aplikasi. Proses *parsing* dilakukan oleh masing-masing DBMS yang didefinisikan pada sintaks (*dbidentifier*) sebelumnya.
    - Karakter-karakter pada golongan ini berada pada posisi diantara karakter ‘[‘ dan karakter ‘]’. Dengan catatan bahwa karakter ‘[’ dan ‘]’ bukan merupakan bagian dari sebuah *string* yang diapit oleh dua buah tanda kutip satu (‘).

Selama proses *scanning* ini, aplikasi akan mengkategorikan atau menggolongkan tiap-tiap karakter apakah termasuk ke dalam golongan sintaks atau ke dalam golongan *block* SQL *query*. Pada contoh *string* masukan di atas, karakter-karakter yang termasuk ke dalam golongan sintaks adalah karakter yang cetak tebal dan yang termasuk ke dalam golongan *block* SQL *query* adalah karakter yang cetak dengan warna abu-abu. Untuk lebih jelasnya dapat dilihat pada kalimat di bawah ini:

Array[0]: **db-alias1[**select x.kolom1 as col1 from tablex x**]**
Array[1]: **union**
Array[2]: **(db-alias2[**select y.kolom2 from tabley y**]**
Array[3]: **minus**
Array[4]**: db-alias3[**select z.kolom3 as col1 from tablez z**]);**

Urutan karakter-karakter yang termasuk ke dalam golongan sintaks tersebut diperiksa dan disesuaikan dengan DFA. Hasil *tracing* tiap-tiap karakter ke DFA dapat dilihat pada tabel 3.5. Dari hasil tracing SQL *query* pada tabel 3.5 menunjukkan bahwa kalimat masukan pada SQL tersebut sudah sesuai dengan DFA pada gambar 3.15 karena *tracing* diawali dari *initial state* q0 dan berakhir di *final state* q15.

Tabel 3.5 *Tracing string* masukan

STEP	TRANSISI
1	d(q0, 'd') = q1
2	d(q1, 'b') = q3
3	d(q3, '-') = q5
4	d(q5, 'a') = q7
5	d(q7, 'l') = q7
6	d(q7, 'i') = q7
7	d(q7, 'a') = q7
8	d(q7, 's') = q7
9	d(q7, '1') = q7
10	d(q7, '[') = q9
11	d(q9, ']') = q11
12	d(q11, 'u') = q13
13	d(q13, 'n') = q18
14	d(q18, 'i') = q24
15	d(q24, 'o') = q30
16	d(q30, 'n') = q36
17	d(q36, '(') = q2
18	d(q2, 'd') = q4
19	d(q4, 'b') = q6
20	d(q6, '-') = q8
21	d(q8, 'a') = q10
22	d(q10, 'l') = q10
23	d(q10, 'i') = q10
24	d(q10, 'a') = q10
25	d(q10, 's') = q10
26	d(q10, '2') = q10
27	d(q10, '[') = q12
28	d(q12, ']') = q17
29	d(q17, 'm') = q23
30	d(q23, 'i') = q29
31	d(q29, 'n') = q35
32	d(q35, 'u') = q41
33	d(q41, 's') = q46
34	d(q46, 'd') = q4
35	d(q4, 'b') = q6
36	d(q6, '-') = q8
37	d(q8, 'a') = q10
38	d(q10, 'l') = q10
39	d(q10, 'i') = q10
40	d(q10, 'a') = q10
41	d(q10, 's') = q10
42	d(q10, '3') = q10
43	d(q10, '[') = q12
44	d(q12, ']') = q17
45	d(q17, ')') = q11
46	d(q11, ';') = q15

Pada akhir proses *scanning* ini, akan dihasilkan sebuah variabel *array* dua dimensi yang berisi *dbidentifier* dan *block* SQL *query*. Pada masing-masing elemen *array* terdapat sebuah *array* yang berisi sebagai berikut :

Array[index][0]: berisi nama atau alias *operand*

Array[index][1]: berisi *string* yang dimaksud oleh *operand*

Array[index][2]: berisi alias *database*

Array[index][3]: berisi kode atau *key* dari tabel jenis *database*

Array[index][4]: berisi keterangan jenis *database*

Array[index][5]: berisi letak baris ke-berapa terdapat alias *database*

Array[index][6]: berisi *block* SQL *query* yang dimaksud oleh *operand*

Pada contoh masukan SQL *query* di atas *array variable* akan berisi sebagai berikut:

Array[0][0]: **operand1**

Array[0][1]: **db-alias1[select x.kolom1 as col1 from tablex x]**

Array[0][2]: **alias1**

Array[0][3]: **1** (misalnya)

Array[0][4]: **oracle** (misalnya)

Array[0][5]: **baris ke-1**

Array[0][6]: **select x.kolom1 as col1 from tablex x**

Array[1][0]: **operand2**

Array[1][1]: **db-alias2[select y.kolom2 from tabley y]**

Array[1][2]: **alias2**

Array[1][3]: **2** (misalnya)

Array[1][4]: **mysql** (misalnya)

Array[1][5]: **baris ke-3**

Array[1][6]: **select y.kolom2 from tabley y**

Array[2][0]: **operand3**

Array[2][1]: **db-alias3[select z.kolom3 as col1 from tablez z])**

Array[2][2]: **alias3**

Array[2][3]: **2** (misalnya)

Array[2][4]: **mysql** (misalnya)

Array[2][5]: **baris ke-5**

Array[2][6]: **select z.kolom3 as col1 from tablez z**

Setelah mendapatkan *array variable* di atas, selanjutnya akan dihasilkan serangkaian sintaks dalam bentuk urutan *operand* dan *operator* dengan mengidentifikasi *string* mana saja yang termasuk *operand* atau *operator* atau simbol asosiasi. Pada contoh *string* SQL *query* sebelumnya akan didapat sebuah *string* berikut ini:

operand1 operator1 (operand2 operator2 operand3);

dimana :

operand1 : **db-alias1[select x.kolom1 as col1 from tablex x]**

operator1 : **union**

operand2 : **db-alias2[select y.kolom2 from tabley y]**

operator2 : **minus**

operand3 : **db-alias3[select z.kolom3 as col1 from tablez z]**

dengan urutan eksekusinya adalah

1. **operand2 operator2 operand3** dikerjakan
2. misalkan menghasilkan sebuah *resulset* atau misal kita sebut dengan **operand4**
3. **operand1 operator1 operand4** dikerjakan
4. menghasilkan sebuah *resultset* (hasil akhir)

Hasil akhir dari SQL *query* tersebut adalah sebuah *resultset* dalam bentuk *array variable* yang berisi data hasil penggabungan data.

3.1.4 Analisis Metode SODA pada Kasus Penggabungan Data

Untuk membangun aplikasi dalam kasus penggabungan data dengan *multiple* DBMS, diperlukan adanya koneksi ke masing-masing *database* yang digunakan. Untuk mengambil data dari *database*, diperlukan adanya *query* dan cara akses ke masing-masing *database*. Penggabungan data bisa dilakukan jika ada dua buah atau lebih sumber data dengan format data yang sama. Secara umum, kita dapat mengambil data dari *table* atau *view* dengan cara melakukan *query* ke *database*, lalu melakukan penggabungan data dari hasil *query* tersebut.

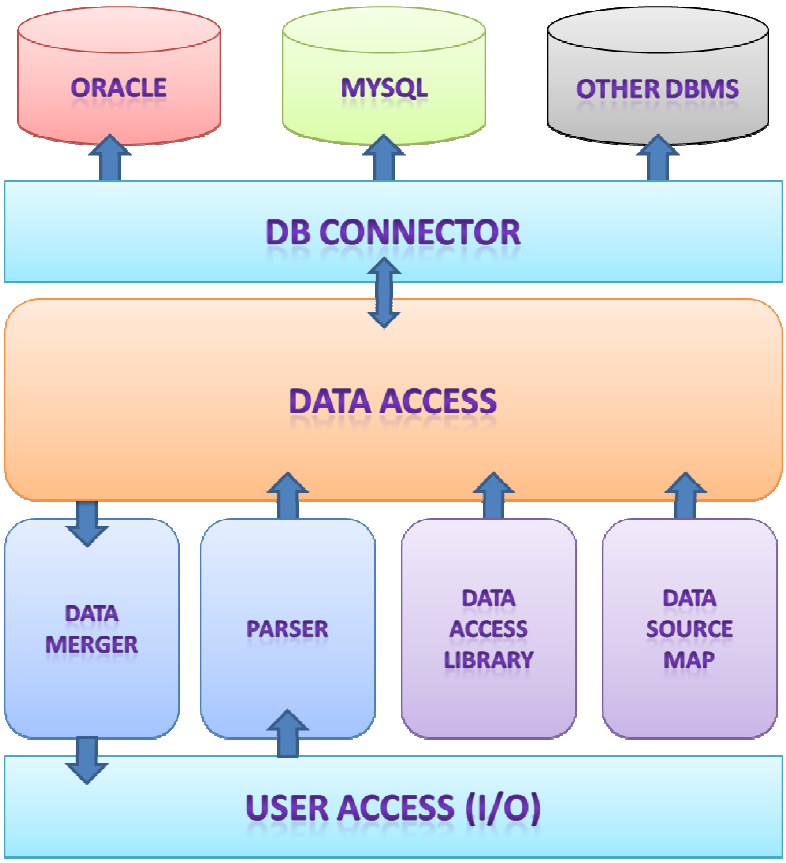
Dari penjelasan sebelumnya, aplikasi dituntut agar dapat melakukan koneksi dan akses ke *database* dengan *multiple* DBMS. Untuk memudahkan penulis dalam tahap implementasi aplikasi nanti, penulis menggunakan metode SODA untuk membantu menangani kasus penggabungan data dengan *multiple* DBMS ini.

Metode SODA adalah sebuah metode pengaksesan *database* dengan cara menyamakan dan menyederhanakan cara akses ke semua *database*. Dengan menggunakan metode ini, *programmer* akan menggunakan cara akses *database* yang sama meskipun *database* yang digunakan berbeda DBMS. Metode SODA ini akan diimplementasikan ke dalam sebuah kelas SODA. Di dalam kelas SODA terdapat satu *method* yang sama untuk melakukan akses data ke *database* meskipun berbeda-beda DBMS. Dengan demikian sintaks untuk mengeksekusi *query* ke DBMS satu dan yang lainnya adalah sama.

Untuk melakukan penggabungan data antar *database* dibutuhkan suatu komponen yang dapat berkomunikasi dengan banyak *database*. Dengan keuntungan (SODA memiliki *method* yang sama untuk akses data ke *database*)

yang dimiliki oleh metode SODA tersebut, penulis menyimpulkan bahwa metode SODA dapat diimplementasikan ke dalam aplikasi untuk melakukan penggabungan data dari *database* dengan *multiple* DBMS. Sebelum melakukan penggabungan data aplikasi harus dapat melakukan komunikasi dan akses data ke semua *database*.

Komponen-komponen dalam kelas SODA dapat dilihat dalam gambar 3.16. Pada gambar tersebut sudah ditambahkan komponen *parser* dan penggabungan data.



Gambar 3.16 Komponen-komponen dalam kelas SODA

Detail dari komponen-komponen dalam kelas SODA tersebut adalah sebagai berikut:

1. *User Access (Input/Output)*

   Komponen ini merupakan suatu *method* dalam kelas SODA yang berfungsi sebagai penerima masukan dari pengguna sekaligus sebagai keluaran data untuk pengguna. Masukan *method* ini berupa *string* SQL *query*. Keluaran *method* ini berupa data dalam variabel *array*.

2. *Parser*

   Komponen ini bertugas untuk memeriksa *string* masukan SQL *query*. Komponen ini melakukan proses *scanning dan parsing*. Komponen ini juga memeriksa apakah data yang akan digabungkan memenuhi syarat untuk digabungkan atau tidak.

3. *DB (database) Connector*

   Komponen ini bertugas untuk melakukan koneksi ke *database* sesuai dengan parameter *database* dari masukan.

4. *Data Access*

   Komponen ini bertugas untuk melakukan akses data (mengeksekusi *query*) ke dalam *database*.

5. *Data Source Map*

   Data source map ini berupa sebuah file yang berisi parameter *database* yang didefinisikan oleh pengguna.

6. *Data Access Library*

   Komponen ini berisi cara akses *database* sesuai dengan DBMS masing-masing.

7. *Data Merger*

   Komponen ini bertugas untuk melakukan penggabungan data setelah komponen *data access* menghasilkan data dalam bentuk *array*.

3.1.5 Analisis Aturan Operasi Set dalam Proses Penggabungan Data

Proses penggabungan data dalam skripsi ini menggunakan salah satu fungsi atau perintah dalam sintaks SQL *query* yaitu *set operation* atau operasi set yang dalam penelitian ini menggunakan fungsi *union/union all*, *minus* dan *intersect*. *Set operation* membutuhkan minimal dua buah relasi (hasil seleksi/*select* SQL *query*). Tidak semua DBMS mempunyai fungsi *set operation* tersebut, seperti pada DBMS MySQL yang tidak mempunyai fungsi *minus*. Informasi selengkapnya dapat dilihat pada tabel 3.6 di bawah ini.

Tabel 3.6 Daftar dukungan fungsi *set operation* pada beberapa DBMS

DBMS	UNION	UNION ALL	MINUS	INTERSECT
ORACLE	Ada	Ada	Ada	Ada
MYSQL	Ada	Ada	-	-
SQL SERVER	Ada	Ada	Ada	-

Perangkat lunak yang akan dibangun mendukung semua fungsi *set operation* (*union/union all, minus,* dan *intersect*) untuk semua DBMS. Aturan fungsi *set operation* yang dipakai dalam proses penggabungan data dalam

penelitian ini mengacu pada fungsi *set operation* DBMS Oracle. Hasil dari tiap-tiap perintah SELECT dapat dianggap sebagai sebuah SET, SQL *set operation* dapat diterapkan pada SET-SET tersebut untuk mencapai hasil akhir dari proses penggabungan data. DBMS Oracle mendukung empat buah *set operation*: *UNION ALL*. *UNION*, *MINUS*, dan *INTERSECT*.

Dua buah perintah *SELECT* dapat digabungkan ke dalam sebuah *query* gabungan dengan menggunakan *set operation*, jika memenuhi kondisi berikut ini:

1. Hasil SET dari masing-masing *query* harus memiliki jumlah kolom yang sama
2. Tipe data dari setiap kolom dalam hasil SET kedua harus cocok dengan tipe data kolom yang sesuai di hasil SET pertama. Tipe data tidak harus sama persis, minimal tipe datanya masih satu tipe atau satu jenis.

Keywords UNION, *UNION ALL*, *MINUS*, dan *INTERSECT* adalah *set operators*. Pada bagian ini penulis akan membahas tentang sintaks, contoh, aturan dan batasan pada empat buah *set operation* tersebut.

3.1.5.1 Set Operator

Berikut ini dijelaskan secara singkat tentang empat buah *set operation* yang didukung oleh DBMS Oracle. Penjelasan tentang *set operation* dapat dilihat pada tabel 3.7.

Tabel 3.7 Perbandingan empat buah *set operation*

OPERATOR	DESKRIPSI
UNION ALL	- Menggabungkan masing-masing hasil dari perintah *SELECT* menjadi sebuah SET hasil (*result set*) - Hanya ada proses penggabungan saja, *result set* tidak dilakukan proses pengurutan data. - Urutan *SELECT* tidak diperhatikan
UNION	- Menggabungkan masing-masing hasil dari perintah *SELECT* menjadi sebuah SET hasil - Mengeliminasi setiap baris-baris yang duplikat (baris-baris yang sama persis) dari SET hasil tersebut. - Operator *UNION* memperhatikan *Case Sensitive Data*. - dilakukan proses pengurutan data pada *result set* - Urutan *SELECT* tidak diperhatikan
MINUS	- Menghasilkan *result set* dari perintah *SELECT* yang pertama - menghilangkan baris-baris yang juga dihasilkan oleh perintah *SELECT* yang kedua - dilakukan proses pengurutan data pada *result set* - Urutan *SELECT* diperhatikan
INTERSECT	- Menghasilkan *result set* hanya baris-baris yang sama dihasilkan oleh kedua perintah *SELECT* - dilakukan proses pengurutan data pada *result set* - Urutan *SELECT* tidak diperhatikan

Sebelum membahas masing-masing *set operator* berikut ini ada dua buah *query* perintah *SELECT* beserta hasilnya sebagai contoh untuk memperlihatkan perbedaan dari masing-masing *set operator*. *Query* pertama menghasilkan semua pelanggan di *region* 5.

```
SELECT CUST_NBR, NAME
FROM CUSTOMER
WHERE REGION_ID = 5;

 CUST_NBR NAME
---------- ------------------------------
         1 Cooper Industries
         2 Emblazon Corp.
         3 Ditech Corp.
         4 Flowtech Inc.
         5 Gentech Industries
```

Query kedua menghasilkan semua pelanggan yang ditangani oleh *sale representative* yang bernama MARTIN.

```
SELECT C.CUST_NBR, C.NAME
FROM CUSTOMER C
WHERE C.CUST_NBR IN (SELECT O.CUST_NBR
                     FROM CUST_ORDER O, EMPLOYEE E
                     WHERE O.SALES_EMP_ID = E.EMP_ID
                     AND E.LNAME = 'MARTIN');

 CUST_NBR NAME
---------- ------------------------------
         4 Flowtech Inc.
         8 Zantech Inc.
```

Jika kita mengamati kedua *result set* tersebut, kita akan menemukan baris yang sama (baris CUST_NBR=4, NAME=Flowtech Inc.). Berikutnya akan dibahas efek dari empat buah *set operation* dari dua buah *result set* di atas.

3.1.5.2 UNION ALL

Operator *UNION ALL* menggabungkan masing-masing hasil dari perintah *SELECT* menjadi sebuah SET hasil (*result set*). Contoh di bawah ini menggambarkan operasi *UNION ALL*:

```
SELECT CUST_NBR, NAME
FROM CUSTOMER
WHERE REGION_ID = 5
UNION ALL
SELECT C.CUST_NBR, C.NAME
FROM CUSTOMER C
WHERE C.CUST_NBR IN (SELECT O.CUST_NBR
                     FROM CUST_ORDER O, EMPLOYEE E
                     WHERE O.SALES_EMP_ID = E.EMP_ID
                     AND E.LNAME = 'MARTIN');

 CUST_NBR NAME
---------- ------------------------------
         1 Cooper Industries
         2 Emblazon Corp.
         3 Ditech Corp.
         4 Flowtech Inc.
         5 Gentech Industries
         4 Flowtech Inc.
         8 Zantech Inc.
```

Seperti yang kita lihat pada hasil *query* di atas, terdapat satu *customer* yang dihasilkan oleh dua *query SELECT*, muncul sebanyak dua kali (Flowtech Inc.). Operator *UNION ALL* menggabungkan semua output dari kedua *query SELECT* tanpa peduli tentang duplikasi apapun di *result set*.

3.1.5.3 UNION

Operator *UNION* mengembalikan semua baris yang berbeda (*all distinct rows*) diambil oleh dua *query SELECT*. Operator *UNION* mengeliminasi duplikasi yang terjadi pada *result set* hasil penggabungan dua *query SELECT*. Contoh di bawah ini menggambarkan operasi *UNION*:

```
SELECT CUST_NBR, NAME
FROM CUSTOMER
WHERE REGION_ID = 5
UNION
SELECT C.CUST_NBR, C.NAME
FROM CUSTOMER C
WHERE C.CUST_NBR IN (SELECT O.CUST_NBR
                     FROM CUST_ORDER O, EMPLOYEE E
                     WHERE O.SALES_EMP_ID = E.EMP_ID
                     AND E.LNAME = 'MARTIN');

 CUST_NBR NAME
---------- ------------------------------
         1 Cooper Industries
         2 Emblazon Corp.
         3 Ditech Corp.
         4 Flowtech Inc.
         5 Gentech Industries
         8 Zantech Inc.
```

Query di atas adalah modifikasi dari *query* sebelumnya, yaitu dengan mengganti operator *UNION ALL* dengan operator *UNION*. Perhatikan bahwa hasil dari *query* di atas hanya berisi semua baris yang berbeda (*all distinct rows*) dan tidak terdapat duplikasi baris. Untuk mengeliminasi duplikasi baris, operasi *UNION* memerlukan tambahan proses jika dibandingkan dengan operasi *UNION ALL*. Tambahan proses yang dimaksud meliputi proses *sorting* dan *filtering* dari *result set*. Jika kita perhatikan baik-baik, kita akan menemukan bahwa *result set* dari operasi *UNION ALL* adalah tidak urut, sedangkan *result set* dari operasi *UNION* adalah urut. Tambahan proses (pada operasi *UNION*) ini tentu saja memerlukan waktu tambahan jika dibandingkan dengan operasi *UNION ALL* meskipun tidak ada duplikasi yang harus dihilangkan. Jika kita sudah yakin tidak akan ada duplikasi di dua buah *query* SELECT, maka lebih baik kita gunakan operasi *UNION ALL* untuk menggabungkan data.

3.1.5.4 INTERSECT

Operator ini hanya menghasilkan baris-baris yang sama dari kedua *query SELECT*. Jika dibandingkan dengan operator *UNION*, *UNION* bertindak seperti *'OR'*, sedangkan *INTERSECT* bertindak sebagai *'AND'*. Sebagai contoh:

```
SELECT CUST_NBR, NAME
FROM CUSTOMER
WHERE REGION_ID = 5
INTERSECT
SELECT C.CUST_NBR, C.NAME
FROM CUSTOMER C
WHERE C.CUST_NBR IN (SELECT O.CUST_NBR
                     FROM CUST_ORDER O, EMPLOYEE E
                     WHERE O.SALES_EMP_ID = E.EMP_ID
                     AND E.LNAME = 'MARTIN');

 CUST_NBR NAME
---------- ------------------------------
         4 Flowtech Inc.
```

Seperti yang sudah kita lihat pada awal pembahasan *set operation* ini, "Flowtech Inc." adalah satu-satunya *customer* yang dihasilkan oleh kedua *query SELECT*. Maka dari itu operator *INTERSECT* pada *query* di atas hanya menghasilkan satu baris saja.

3.1.5.5 MINUS

MINUS menghasilkan semua baris dari *query SELECT* pertama yang tidak juga dihasilkan oleh *SELECT* kedua. Sebagai contoh:

```
SELECT CUST_NBR, NAME
FROM CUSTOMER
WHERE REGION_ID = 5
MINUS
SELECT C.CUST_NBR, C.NAME
FROM CUSTOMER C
WHERE C.CUST_NBR IN (SELECT O.CUST_NBR
                     FROM CUST_ORDER O, EMPLOYEE E
                     WHERE O.SALES_EMP_ID = E.EMP_ID
                     AND E.LNAME = 'MARTIN');

 CUST_NBR NAME
---------- ------------------------------
         1 Cooper Industries
         2 Emblazon Corp.
         3 Ditech Corp.
         5 Gentech Industries
```

Pada *result set query* di atas, kita tidak melihat "Zantech Inc.". Hal penting yang harus dicatat di sini adalah urutan eksekusi dari *query* SELECT pada *set operation* yaitu dari atas ke bawah. Hasil dari operasi UNION, UNION ALL, and INTERSECT tidak akan berubah jika kita mengubah urutan dari *query* SELECT. Namun hasil untuk operasi MINUS akan berbeda jika kita mengubah urutan dari *query* SELECT. Jika kita menuliskan kembali *query* sebelumnya dengan menukar posisi dari kedua *query* SELECT, maka kita akan mendapatkan *result set* yang berbeda.

```
SELECT C.CUST_NBR, C.NAME
FROM CUSTOMER C
WHERE C.CUST_NBR IN (SELECT O.CUST_NBR
                     FROM CUST_ORDER O, EMPLOYEE E
                     WHERE O.SALES_EMP_ID = E.EMP_ID
                     AND E.LNAME = 'MARTIN')
MINUS
SELECT CUST_NBR, NAME
FROM CUSTOMER
WHERE REGION_ID = 5;

 CUST_NBR NAME
---------- ------------------------------
         8 Zantech Inc.
```

Berikut ini adalah rangkuman beberapa aturan sederhana, batasan, dan catatan yang tidak memerlukan contoh:

- *Set operations* tidak diijinkan pada tipe data BLOB, CLOB, BFILE, dan VARRAY, tidak juga diijinkan pada kolom tabel bersarang (*nested table columns*).
- Karena operator *UNION*, *INTERSECT*, dan *MINUS* melibatkan proses operasi pengurutan (*sorting*), maka tidak dibolehkan menggunakan tipe data *LONG*. Tetapi untuk operator *UNION ALL* dibolehkan menggunakan tipe data *LONG*.
- *Set operations* tidak dibolehkan pada *query SELECT* yang mengandung *TABLE collection expressions*.
- *Query SELECT* yang terlibat di *set operation* tidak dapat digunakan untuk klausa *UPDATE*.
- Jumlah dan ukuran kolom dalam daftar *SELECT* dibatasi oleh ukuran *database block*. Total *bytes* dari kolom yang di-*SELECT* tidak dapat melebihi satu *database block*.

3.1.6 Analisis Kebutuhan Non Fungsional

Analisis non fungsional dalam pembangunan aplikasi dilakukan untuk menghasilkan spesifikasi kebutuhan non fungsional. Spesifikasi kebutuhan non fungsional adalah spesifikasi yang rinci tentang hal-hal yang akan dilakukan sistem ketika diimplementasikan. Analisis kebutuhan ini diperlukan untuk menentukan keluaran yang akan dihasilkan sistem, masukan yang diperlukan sistem, lingkup proses yang digunakan untuk mengolah masukan menjadi keluaran, serta kontrol terhadap sistem.

3.1.6.1 Analisis Perangkat Keras

Perangkat keras adalah seluruh komponen atau unsur peralatan yang digunakan untuk menunjang pembangunan perangkat lunak atau aplikasi. Ada dua spesifikasi perangkat keras yang dijelaskan di bab ini, yaitu perangkat keras yang digunakan untuk mengembangkan aplikasi dan perangkat keras yang digunakan untuk implementasi aplikasi ini.

Adapun perangkat keras yang digunakan untuk mengembangkan aplikasi ini antara lain:

a. *Processor* Intel ® Core ™ i5 CPU M 520 @2.40 GHz
b. *Memory* berkapasitas 2 GB
c. *Display adapter* : Mobile Intel ® HD Graphics
d. *Harddisk* berkapasitas 250 GB dan DVD-ROM
e. LAN *card onboard*
f. Monitor SVGA ukuran 12 inci dengan resolusi maksimal 1280 x 800
g. *Keyboard* dan *Mouse/Mousepad*

Perangkat keras yang dibutuhkan untuk implementasi aplikasi yang akan dibangun minimal memiliki spesifikasi sebagai berikut :

a. *Processor* sejenis *dual core* dengan kecepatan 2 GHz
b. *Hard Disk* 40 GB.
c. *Memory* 2 GB.
d. Perangkat input standar seperti *keyboard* dan *mouse.*
e. Perangkat *output* standar seperti monitor dengan resolusi minimal 1024 x 768.
f. LAN *card onboard.*
g. Jaringan yang menghubungkan komputer *client* dengan *server*.

3.1.6.2 Analisis Perangkat Lunak

Perangkat lunak yang umum digunakan untuk mengolah *database* adalah SQL *command line*. SQL *command line* ini dapat dijalankan melalui *command line* seperti *cmd* atau *command prompt* sistem operasi Windows, *shell* pada sistem operasi unix, sqlplus pada DBMS oracle, dan lain sebagainya. *User* dengan tingkat kemampuan *database* yang tinggi seperti *Database Programmer* dan DBA (*Database Administrator*) biasanya lebih menyukai menggunakan SQL *command line*.

Tetapi banyak juga *user* yang menggunakan perangkat lunak pengolah *database* yang sudah mempunyai *user interface*, antara lain: phpmyadmin, SQLyog, TOAD for MySQL dan MySQL Front untuk *database* MySQL, dan TOAD for Oracle untuk *database* Oracle. Penggunaan perangkat lunak pengolah

database yang memiliki *user interface* lebih disukai karena lebih interaktif dan mudah digunakan jika dibandingkan dengan SQL *command line*. Tetapi pada dasarnya untuk mengelola data dalam suatu *database*, *user* harus mengetahui macam-macam SQL *query* diantaranya DDL (*Data Definition Language*) dan DML (*Data Manipulation Language*).

Hasil dari *query* yang dijalankan oleh *user* dapat disimpan ke dalam tabel di *database* dan bisa juga disimpan atau diekspor ke dalam *text file* dengan alasan tertentu. Perangkat lunak yang digunakan untuk mengolah data hasil *query* yang berupa *text file* dari masing-masing *database* yaitu teks editor (*notepad, wordpad, notepad*++, ultraedit, editplus), microsoft excel dan access, dan sebagainya. Masing-masing perangkat lunak tersebut memiliki spesifikasi dan fitur yang berbeda sehingga *user* dapat memilih sesuai dengan keinginan *user*.

Bahasa pemrograman yang digunakan dalam membangun aplikasi yang menggunakan metoda SODA ini adalah PHP versi 5.3.5. Sedangkan perangkat lunak yang digunakan sebagai *text editor*, *html script editor*, *javascript editor* dan *php script editor* adalah notepad++ dan Adobe Dreamweaver CS3. Perangkat lunak yang digunakan untuk mendesain FSA (*Finite State Automata*) adalah JFLAP Version 7.

3.1.6.3 Analisis Pengguna atau *User*

Karakteristik dari pengguna aplikasi yang akan dibangun ini adalah memiliki jenjang pendidikan sarjana atau diploma, dan memiliki kemampuan di bidang *database* dan pemrograman dengan bahasa pemrograman PHP. Aplikasi yang akan dibangun berhubungan dengan *database* dan dibuat ke dalam aplikasi web dengan menggunakan bahasa pemrograman PHP.

3.1.6.4 Analisis Jaringan

Jaringan komputer yang dipakai sebagai penghubung antar komputer-komputer dengan *server* disesuaikan dengan perusahaan atau instansi yang mengimplementasikan aplikasi ini. Dalam penelitian ini studi kasus dilakukan di Telkom, maka jaringan yang dipakai untuk implementasi aplikasi ini adalah jaringan yang dipakai di internal Telkom. Gambaran jaringan di Telkom tidak dijelaskan di penelitian ini dengan alasan keamanan data Telkom.

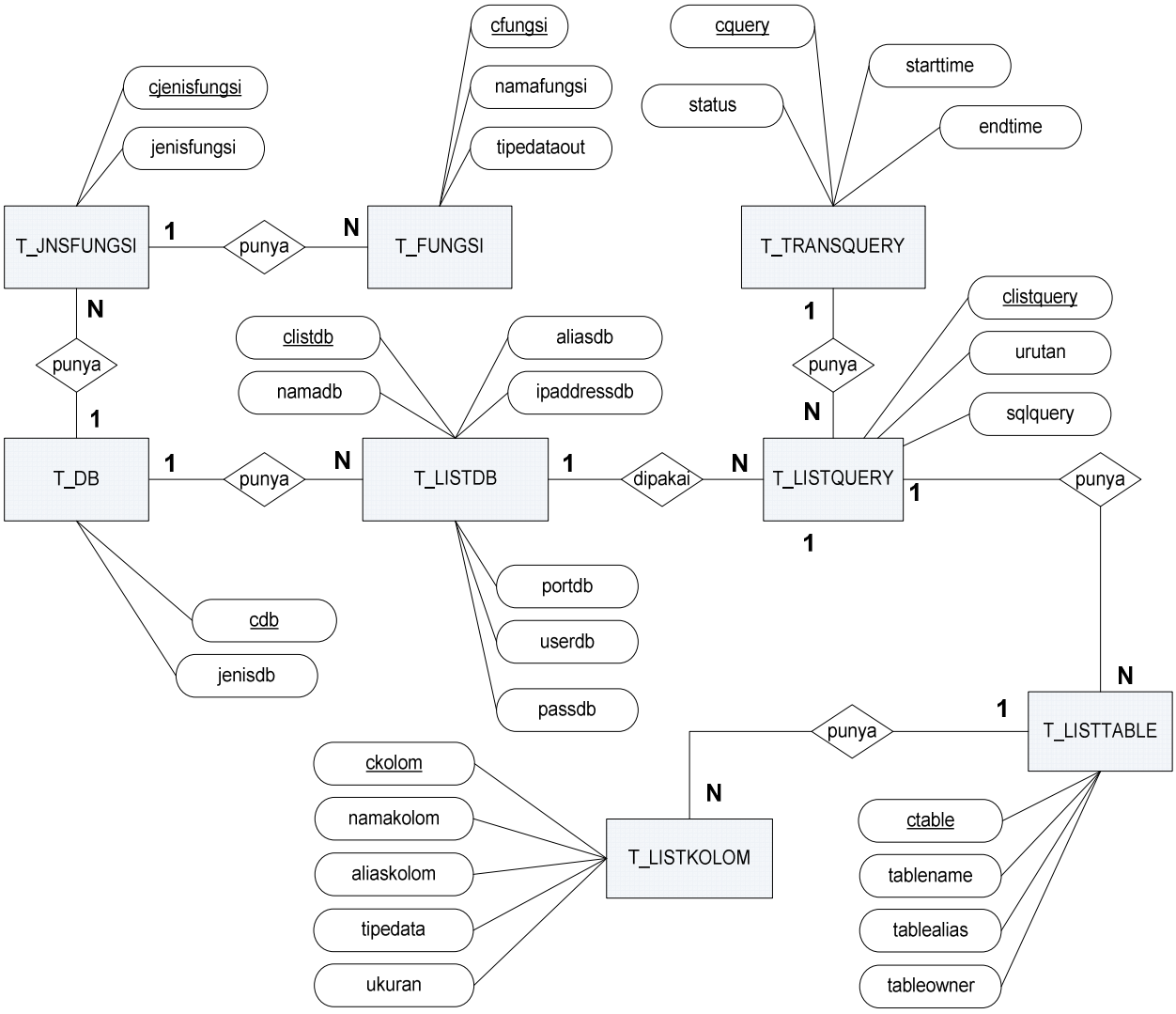
3.1.6.5 Analisis Basis Data

Basis data atau *database* dipakai sebagai tempat penyimpanan data. Dalam penelitian ini studi kasus dilakukan di Telkom, maka *database* yang dipakai untuk sumber data dalam implementasi SODA ini adalah *database* yang dipakai di Telkom. Dalam penelitian ini *database* yang dipakai adalah *database* yang digunakan pada aplikasi *web portal faabula Telkom*. Data yang digunakan diantaranya adalah data pelanggan (flexi, speedy dan telepon rumah), data tagihan pelanggan, data tunggakan pelanggan, dan data pembayaran pelanggan. Sumber datanya berasal dari *database* di aplikasi yang berbeda. *Database* yang menyimpan data pelanggan dan data tagihan berbeda dengan *database* yang menyimpan data pembayaran.

Dari hasil analisa, diperlukan sebuah *database* untuk menyimpan data atau informasi sebagai berikut :

1. Data tentang *keyword* yang dapat dipakai dalam menuliskan SQL *query* dan fungsi-fungsi yang ada dalam SQL *query*.
2. Data tentang *history* transaksi SQL *query* yang pernah dijalankan.
3. Data struktur kolom dan tipe data transaksi SQL *query*.

Kemudian dari data yang telah diperoleh, dibangun sebuah desain *database* dan desain fitur-fitur lainnya. Penulis menggunakan *Entity Relational Diagram* (ERD) untuk merancang *database*. ERD yang dibuat adalah sebagai berikut :



Gambar 3.17 ERD Aplikasi SODA *Data Merger*

Untuk mengaplikasikan aturan produksi yang dipakai sebagai acuan untuk proses *parsing*, tabel transisi pada aturan produksi tersebut dikonversikan ke dalam *array data format.* Sehingga ada dua macam penyimpanan data, yaitu dalam *database* dan dalam *text file*.

3.1.7 Analisis Kebutuhan Fungsional

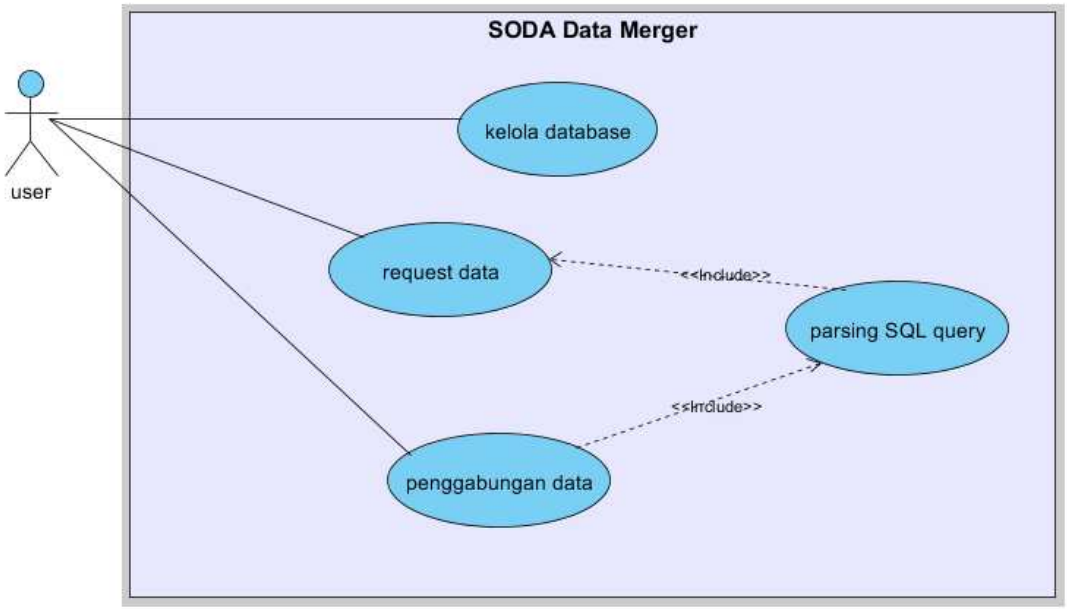
Analisis kebutuhan fungsional dilakukan untuk memberikan gambaran aliran data yang ada pada program aplikasi yang akan dibangun. Aplikasi yang akan dibangun menggunakan analisa yang berorientasi objek. Analisa dan desain aplikasi yang berorientasi objek ini dibuat dengan menggunakan UML (*Unified Modelling Language*). UML merupakan bahasa yang digunakan untuk memvisualisasikan dan mendokumentasikan hasil analisa dan desain aplikasi yang berorientasi objek.

Kebutuhan fungsional pada aplikasi yang menggunakan metode SODA ini dimodelkan ke dalam UML Diagrams, meliputi *use case*, *activity diagram*, *sequence diagram*, *class diagram*, dan *deployment diagram*.

3.1.7.1 *Use Case Diagram*

Use Case adalah teknik untuk merekam persyaratan fungsional sebuah sistem. *Use Case* mendeskripsikan interaksi tipikal antara para pengguna sistem dengan sistem itu sendiri, dengan memberi sebuah narasi tentang bagaimana sistem tersebut digunakan.

Berikut merupakan diagram *use case* dari aplikasi *SODA Data Merger* yang akan dibangun.



Gambar 3.18 Diagram *Use Case* Aplikasi SODA *Data Merger*

3.1.7.2 Skenario *Use Case*

Skenario *use case* adalah rangkaian langkah-langkah yang menjabarkan sebuah interaksi antara seorang pengguna dengan sebuah sistem di dalam masing-masing *use case*. Dalam skenario *use case* terdapat informasi seperti :

1. nama *use case*,
2. *use case* ID,
3. deskripsi *use case*,
4. aktor *use case*,
5. *pre condition*,
6. *post condition*,
7. *include use case*,
8. *extend use case*, dan
9. interaksi antara *user* dengan sistem yang ditulis dalam *user action* dan *system response*.

Skenario *use case* aplikasi *SODA Data Merger* dapat dijelaskan pada tabel 3.8 sampai dengan tabel 3.11. Skenario *use case* berhubungan dengan *sequence diagram* karena skenario masing-masing *use case* ini dipakai sebagai acuan untuk membuat *sequence diagram*.

Tabel 3.8 Skenario *use case* Kelola *Database*

Use Case		Kelola *Database*
Use Case ID (UCID)		1
Description		Menambah, mengubah dan menghapus *database setting*
Actor		*User*
Precondition		*User* memilih menu kelola data *database*.
Post condition		Data yang berisi *Setting database* disimpan dalam *database*.
Include Use Case		-
Extend Use Case		-
Actor Actions		*System Response*
1. *User* memilih menu kelola data *database*		2. Sistem menampilkan *form* untuk memasukkan data parameter *database setting.* (parameter : jenis *database*, nama *database*, alias *database*, *ip address*, *port*, *username* dan *password database*) Sistem menampilkan daftar data parameter *database setting* yang ada di *database*
3. *User* menginput data parameter *database setting* sesuai form masukan yang tersedia.		4. Sistem melakukan tes koneksi ke *database* yang diinputkan oleh *user*

		5. [Jika koneksi gagal] Sistem menampilkan pesan *error* yang berisi informasi bahwa koneksi ke *database* gagal
6. [Jika koneksi gagal] *User* menginput kembali data parameter *database setting*		7. [Jika koneksi berhasil] Sistem menyimpan parameter *database setting* ke dalam *database*. Sistem menampilkan pesan yang berisi informasi bahwa koneksi ke *database* berhasil.

Tabel 3.9 Skenario *use case Request Data*

Use Case		Request Data
Use Case ID (UCID)		2
Description		*User* melakukan *request data* dengan cara memasukkan *script* SQL *query*
Actor		*User*
Precondition		*User* membuka menu penggabungan data
Post condition		*User* dapat memasukkan *script* SQL *query*
Include Use Case		-
Extend Use Case		-
Actor Actions		*System Response*
1. *User* memilih menu *Data Merger* (menu penggabungan data) 3. *User* memasukkan *script* SQL *query* (sesuai format sintaks yang diterima aplikasi)		2. Sistem menampilkan *form textarea* sebagai masukan *script* SQL *query*. 4. Sistem membaca masukan *script* SQL *query* ke dalam satu variabel. 5. Sistem menyimpan masukan tersebut ke *database*.

		6. Sistem meneruskan variabel tersebut untuk dilakukan proses *parsing* SQL *query*.

Tabel 3.10 Skenario *use case Parsing* SQL *Query*

Use Case		*Parsing* SQL *Query*
Use Case ID (UCID)		3
Description		Sistem melakukan proses *parsing* untuk memeriksa kebenaran SQL *query* sesuai dengan aturan produksi.
Actor		*Sistem*
Precondition		Terdapat sebuah *string* SQL *query* yang didefinisikan sebelumnya
Post condition		Terdapat kondisi bahwa *string* SQL *query* diterima atau ditolak.
Include Use Case		*Request Data*
Extend Use Case		-
Actor Actions		*System Response*
		1. Sistem mengambil inputan atau masukan berupa *string* SQL *query*. 2. Sistem mengambil aturan produksi berupa DFA ke dalam suatu *array variabel* 3. Sistem melakukan proses *scanning* untuk mengidentifikasi karakter-karakter pada *string* SQL *query* menjadi token-token tertentu. 4. Sistem memeriksa format masukan, apakah sesuai dengan format pada DFA yang ditentukan atau tidak.

		5. Sistem melakukan pengecekan karakter kurung siku dan kurung buka
7. [Jika ada kesalahan] *User* memperbaiki *string* SQL *query*. Lalu kembali ke langkah pertama.		6. [Jika ada kesalahan] Sistem memberikan pesan bahwa ada kesalahan sintaks terutama pada karakter kurung siku. 7. [Jika tidak ada] Sistem menentukan operand-operand yang berisi *format block* SQL *body* 8. Sistem menentukan variabel *dbalias* 9. Sistem melakukan pengecekan *dbalias* ke *database.*
10. [Jika ada kesalahan] *User* memperbaiki *string* SQL *query*. Lalu kembali ke langkah pertama.		10. [Jika ada kesalahan] Sistem memberikan pesan bahwa ada kesalahan *dbalias*. 11. Sistem melakukan pengecekan *set operator*
12. [Jika ada kesalahan] *User* memperbaiki *string* SQL *query*. Lalu kembali ke langkah pertama.		12. [Jika ada kesalahan] Sistem memberikan pesan bahwa ada kesalahan *setoperator*. 13. Sistem melakukan pembentukan rumus berupa *array variabel* yang berisi urutan operand dan operator. 14. Sistem mengeksekusi *block* SQL *body* per masing-masing operand. 15. Sistem menyimpan data operand dan data yang berhubungan dengan operand ke dalam *database* 16. Sistem menampilkan *preview* dan *highlight sintaks* SQL *query*

Tabel 3.11 Skenario *use case Penggabungan Data*

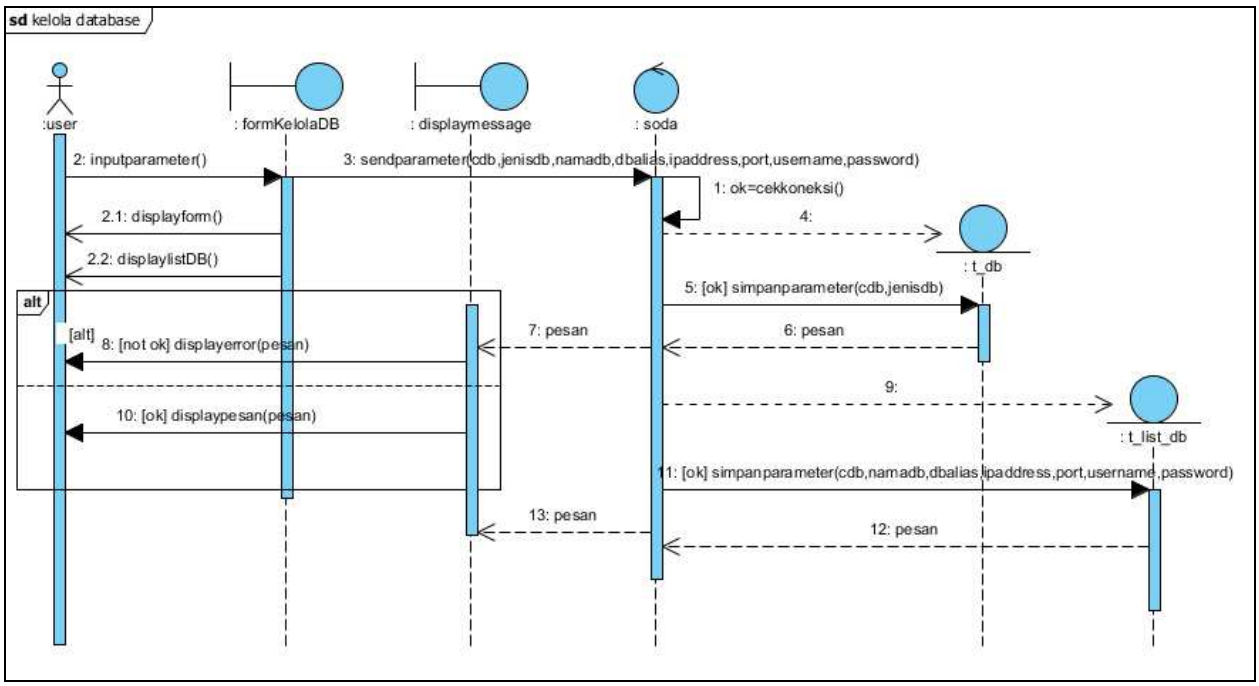
Use Case		*Penggabungan Data*
Use Case ID (UCID)		4
Description		Sistem melakukan proses penggabungan data.
Actor		*User*
Precondition		Sistem menerima masukan berupa *string* SQL *query* dan *array variable operand operator.*
Post condition		Menampilkan data hasil penggabungan data ke dalam *data grid view*.
Include Use Case		*Parsing* SQL *Query*
Extend Use Case		-
Actor Actions		*System Response*
		1. Sistem menerima masukan *string* SQL *query* dan *array variable operand operator*. 2. Sistem mengeksekusi *block* SQL *body* per masing-masing operand. Untuk mendapatkan data per operand. Sistem menyimpan data tersebut ke *database*. 3. Sistem mengidentifikasi rumus jika terdapat pengelompokon rumus. 4. Sistem menentukan rumus mana yang harus dieksekusi. (menentukan derajat operand) 5. Sistem melakukan penggabungan data sesuai dengan urutan rumus dan *set operator* yang didefinisikan. 6. Sistem melakukan proses *data sorting* jika *set operator* yang dipakai adalah

		selain UNION ALL. 7. Sistem mengupdate data transaksi *query* ke *database* 8. Sistem menyimpan data hasil penggabungan data ke *database* 9. Sistem menampilkan data hasil penggabungan data.

3.1.6.3 *Sequence Diagram*

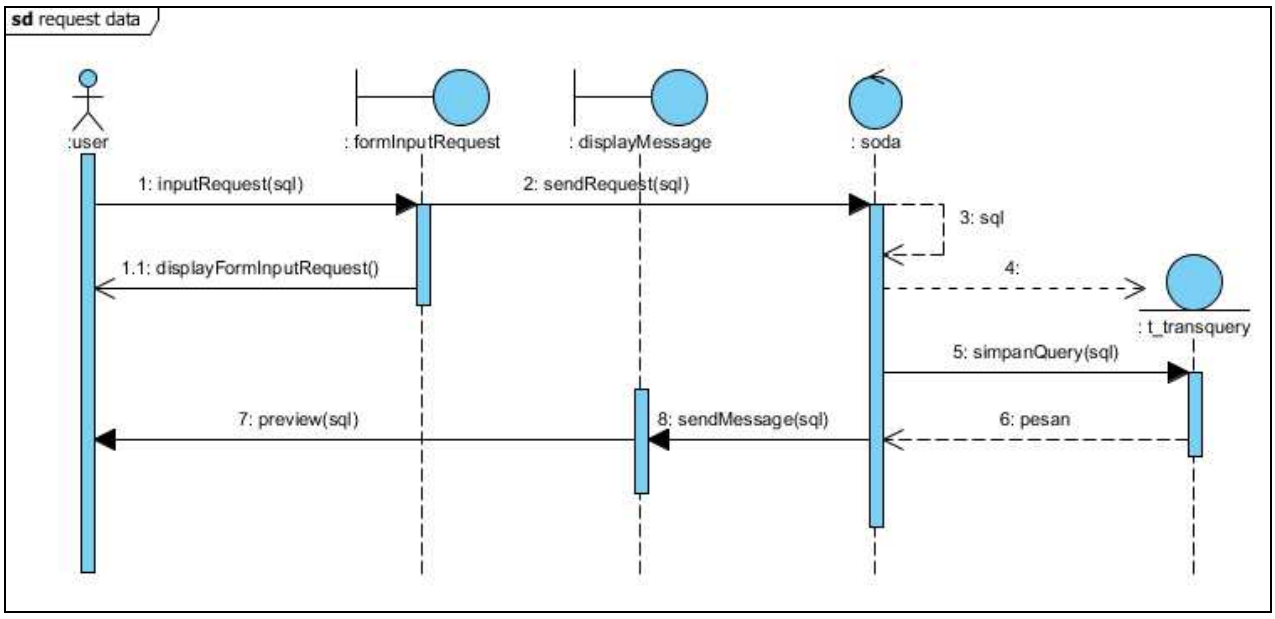
Sebelum melanjutkan ke proses desain logik bagaimana aplikasi perangkat lunak akan bekerja, diperlukan adanya investigasi dan pendefinisian tingkah laku sistem sebagai sebuah "*black box*". Tingkah laku sistem adalah deskripsi tentang apa yang dilakukan oleh sistem tanpa menjelaskan bagaimana sistem tersebut melakukannya. Salah satu cara mendeskripsikannya adalah dengan menggunakan *sequence diagram* [2:137].

Sebuah *sequence diagram* adalah diagram yang menunjukkan skenario dari *use case*, kejadian yang ada antara aktor dan sistem. Sistem yang dimaksud dianggap sebagai sebuah *black-box*, penekanan dari diagram ini adalah kejadian yang terjadi antara aktor dan sistem dalam ruang lingkup sistem.



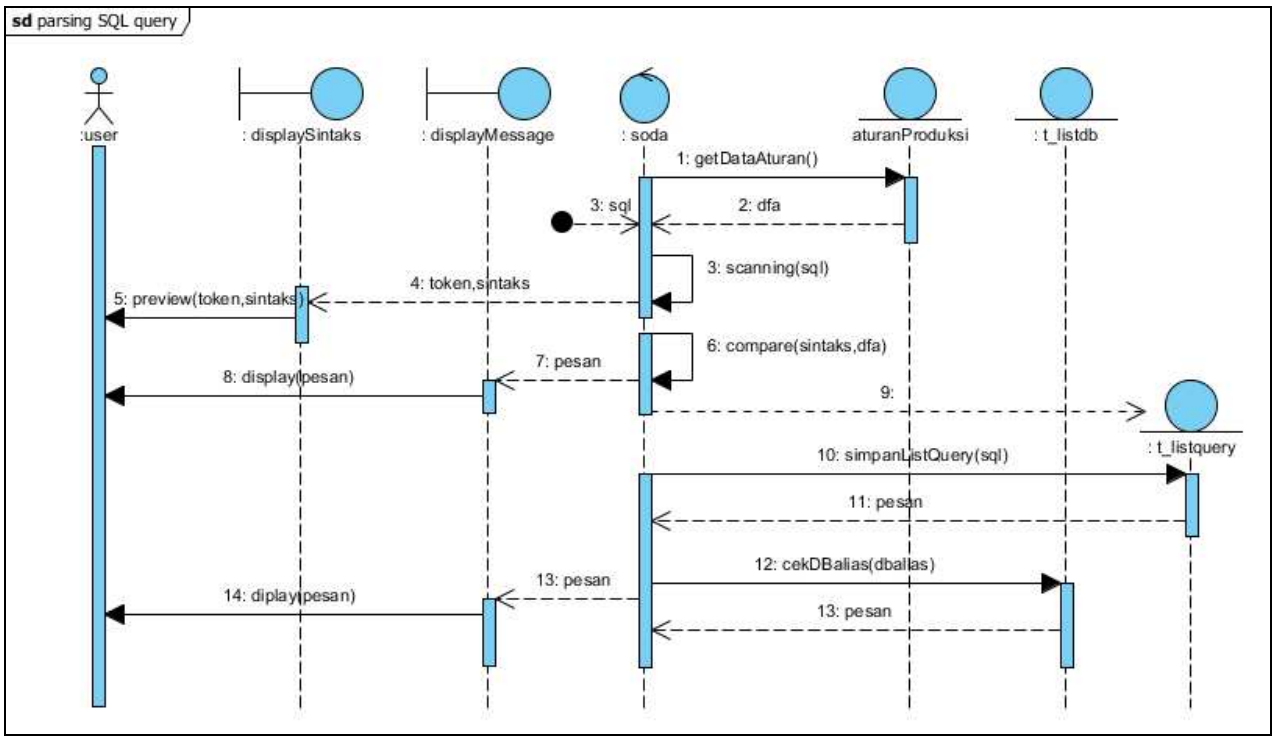
Gambar 3.19 *Sequence Diagram* skenario *Kelola Database*

Pada gambar 3.19 *sequence diagram* dari *use case kelola database* menggunakan *interface lifeline* :formKelolaDB dan :displaymessage, menggunakan *control lifeline* :soda dan *entity lifeline* :t_list_db dan :t_db. *Interface lifeline* :formKelolaDB untuk menampilkan form masukan data parameter *database*. *Interface lifeline* :displaymessage untuk menampilkan pesan kesalahan maupun pesan berhasil. *Control lifeline* :soda menyimpan data parameter *database* ke *Entity lifeline* :t_list_db dan :t_db.



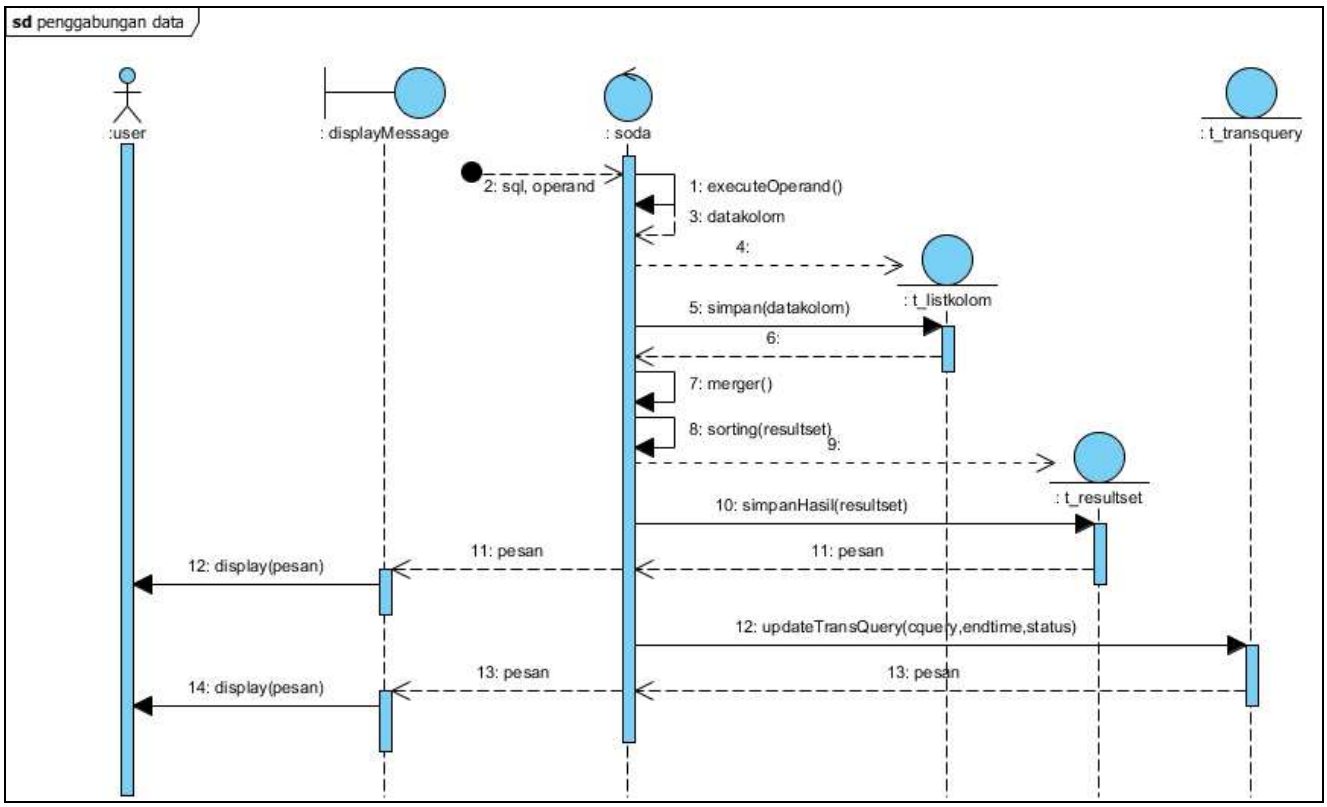
Gambar 3.20 *Sequence Diagram* skenario *Request Data*

Use case Request Data mempunyai fungsional untuk memasukkan *inputan* berupa sintaks SQL query ke *database* untuk melakukan penggabungan data dari bermacam-macam *database*. Di dalam *Sequence Diagram* pada gambar 3.20 terdapat dua buah *interface lifeline*, yang pertama adalah :formInputRequest() yang digunakan untuk menampilkan *form* masukan berupa *textarea*. Sedangkan yang kedua adalah :displaymessage yang digunakan untuk menampilkan preview sintaks sql yang sudah dimasukkan oleh pengguna. *Control lifeline* :soda digunakan untuk menerima pesan *variable* SQL *query* untuk selanjutnya akan dilakukan proses *parsing* dan penggabungan data.



Gambar 3.21 *Sequence Diagram* skenario *Parsing* SQL *Query*

Use case Parsing SQL *Query* mempunyai fungsional untuk melakukan proses *parsing* pada masukan SQL *Query*. Kondisi awal adalah menerima masukan berupa *string* SQL *Query*. Kondisi akhir adalah terbentuk variabel operand dalam suatu *array*. Proses parsing juga menghasilkan format operand dan operator untuk memudahkan sistem dalam melakukan penggabungan data. Di dalam *sequence diagram* pada gambar 3.21 menggunakan *interface lifeline* :displaysintaks untuk menampilkan sintaks masukan dari pengguna dan *interface lifeline* :displaymessage untuk menampilan suatu pesan (kesalahan atau keberhasilan). *Control lifeline* :soda mengambil aturan produksi dari *entity lifeline* :aturanproduksi. :Soda juga melakukan pengecekan *db alias* ke *entity lifeline* :t_listdb dan menyimpan daftar SQL *query* hasil proses parsing ke *entity lifeline* :t_listquery.

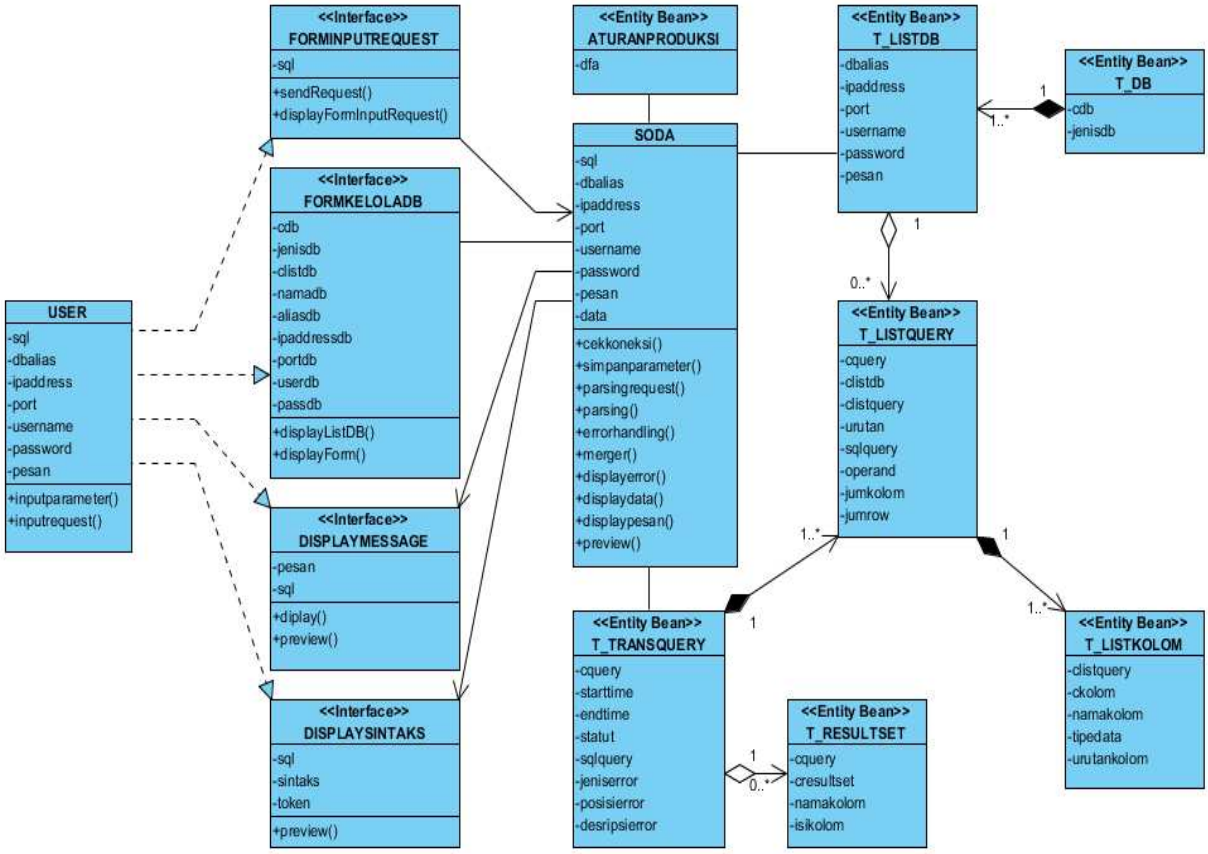


Gambar 3.22 *Sequence Diagram* skenario Penggabungan Data

Use case Penggabungan Data mempunyai fungsional untuk melakukan proses penggabungan data. Kondisi awal menerima variabel operand dalam bentuk *array* yang berisi data masing-masing SQL *query* dan parameter *database*-nya. *Control lifeline* :soda mengeksekusi semua SQL *query* yang ada dalam *array* operand kemudian akan menyimpan data daftar kolom hasil *query* ke dalam *entity lifeline* :t_listkolom. *Control lifeline* :soda melakukan penggabungan data (melalui *method merger*) dan melakukan proses *sorting* hasil penggabungan data jika *set operator* yang ada adalah selain *UNION ALL*. Hasil akhir berupa *result set* di simpan ke dalam *entity lifeline* :t_resultset dan akan melakukan *update log status* ke dalam *entity lifeline* :t_transquery. *Resultset* tersebut ditampilkan ke pengguna melalui *interface lifeline* :displayMessage.

3.1.6.4 *Class Diagram*

Class Diagram mendeskripsikan jenis-jenis objek dalam sistem dan berbagai macam hubungan statis yang terdapat di antara objek-objek tersebut. *Class Diagram* juga menunjukkan properti dan operasi sebuah *class* dan batasan-batasan yang terdapat dalam hubungan-hubungan objek-objek tersebut.



Gambar 3.23 *Class Diagram* pada aplikasi *SODA Data Merger*

3.1.6.5 *Activity Diagram*

Activity Diagram adalah teknik untuk menggambarkan logika procedural, proses bisnis, dan jalur kerja. Dalam beberapa hal, diagram ini memainkan peran mirip dengan diagram alir, tetapi perbedaan prinsip yaitu *activity diagram* mendukung *behavior parallel.*

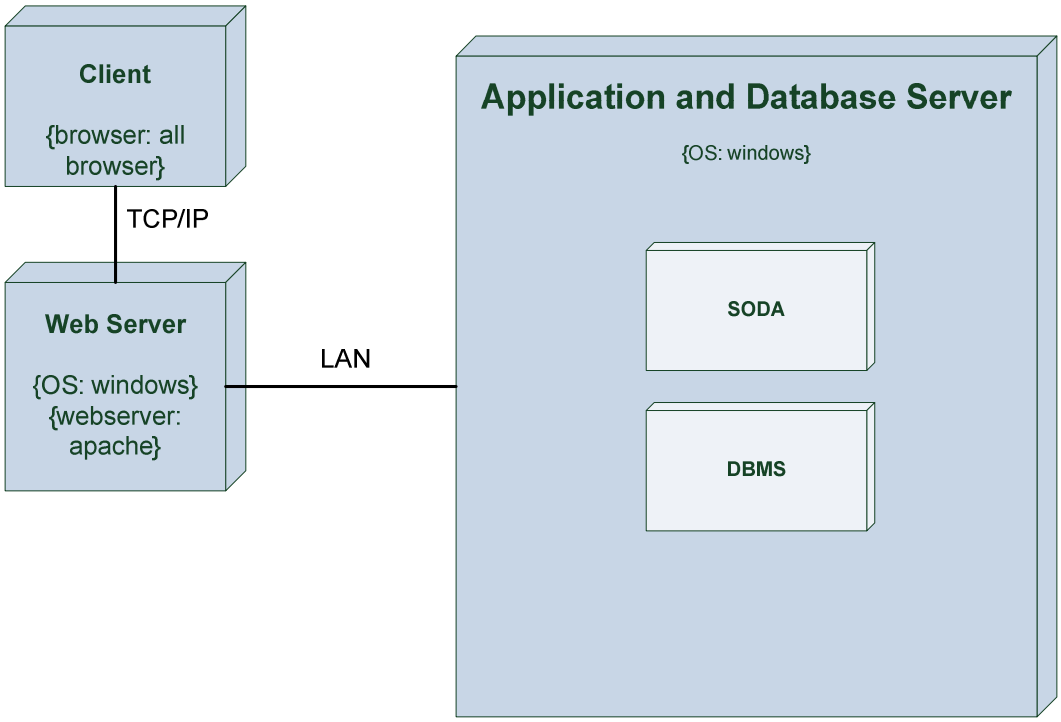
Gambar 3.24 menunjukkan *activity* pada saat proses penggabungan data.



Gambar 3.24 *Activity Diagram* pada SODA Data Merger

3.1.6.6 *Deployment Diagram*

Deployment diagram menggambarkan bagaimana komponen di-*deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, server atau perangkat keras), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi *server*, dan hal-hal lain yang bersifat fisikal. *Deployment diagram* pada penelitian ini dapat dilihat pada gambar di bawah ini.



Gambar 3.25 *Deployment Diagram SODA Agregator*

3.2 Perancangan Sistem

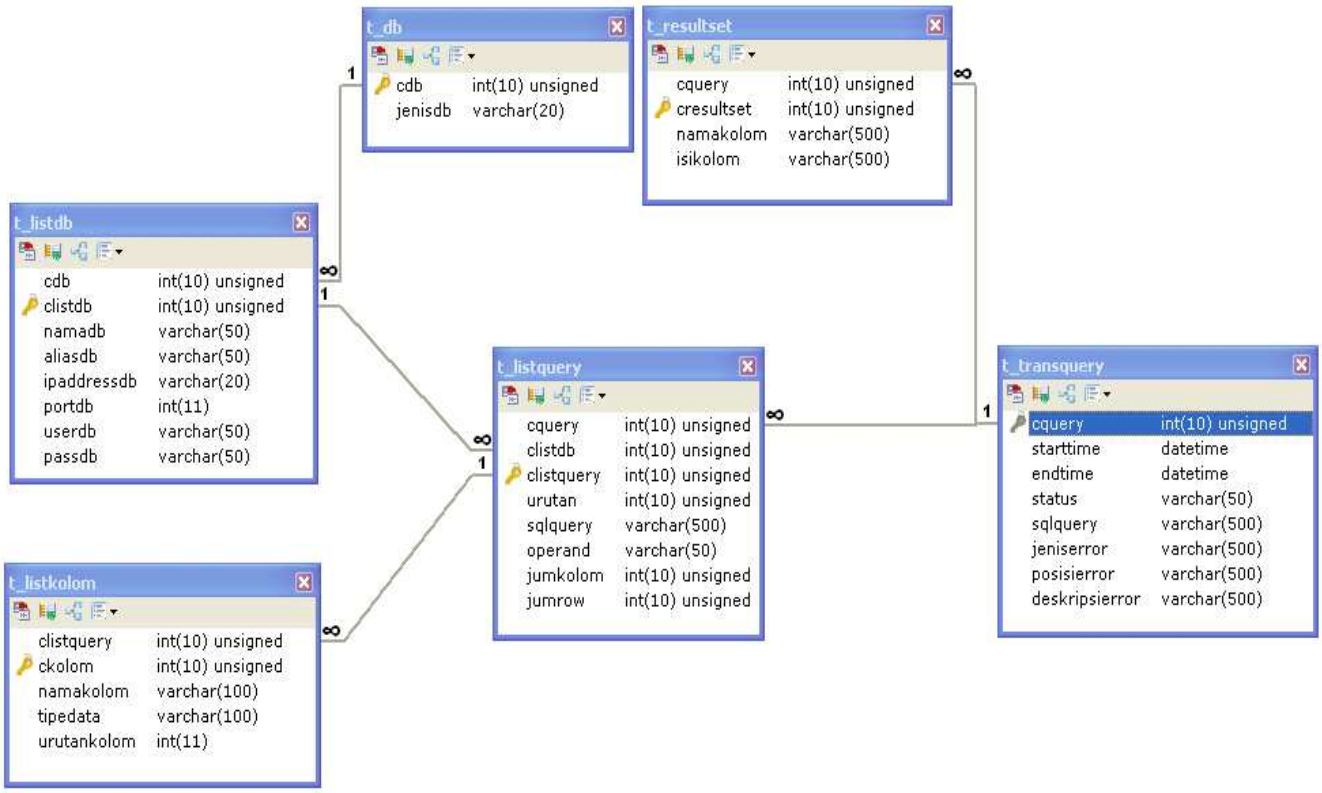
Pada tahap perancangan sistem dalam bab ini, dilakukan perancangan basis data, perancangan struktur menu, perancangan antar muka dan perancangan prosedural.

3.2.1 Perancangan Data

Perancangan data terdiri dari skema relasi, struktur data dan table. Penyimpanan data dalam aplikasi yang akan dibangun terdiri dari dua macam penyimpanan data. Yang pertama, untuk mengaplikasikan aturan produksi yang dipakai sebagai acuan untuk proses *parsing*, tabel transisi pada aturan produksi tersebut dikonversikan ke dalam *array data format*. Sehingga diperlukan suatu file untuk menyimpan variabel *array* statis tersebut. Yang kedua, untuk penyimpanan data parameter *database* dan data transaksi penggabungan data pada aplikasi yang akan dibangun dibuat dalam bentuk tabel.

3.2.1.1 Skema Relasi

Relasi antar tabel merupakan gabungan antar *file* yang mempunyai kunci utama yang sama, sehingga *file-file* tersebut menjadi satu kesatuan yang dihubungkan oleh *field* kunci (*Primary Key*). Pada proses ini elemen-elemen data dikelompokkan menjadi satu *file database* beserta *entitas* dan hubungannya. Skema relasi aplikasi yang akan dibangun dapat dilihat pada gambar 3.26 berikut ini.



Gambar 3.26 Skema Relasi Basis Data Aplikasi

3.2.1.2 Struktur Data dan Struktur Tabel

Tabel-tabel yang terdapat dalam basis data yang digunakan dalam sistem yang akan dibangun dapat dilihat pada tabel 3.12 sampai dengan tabel 3.18 berikut ini.

Tabel T_DB digunakan untuk merekam data parameter jenis *database*, seperti mysql, mssql, oracle, dsb. Tabel ini memiliki *primary key* pada *field* CDB.

Tabel 3.12 Tabel T_DB

ATRIBUT	TIPE DATA	PANJANG	NULL	KETERANGAN
cdb	int	10	NO	Primary Key
jenisdb	Varchar	20	NO	Unique Index

Tabel T_LISTDB digunakan untuk merekam data parameter *database*. Data ini dipakai untuk mengenali *database* mana yang akan diakses oleh pengguna melalui alias *database*. Tabel ini memiliki *primary key* pada *field* CLISTDB. Tabel ini memiliki *foreign key* pada kolom CDB yang berelasi dengan tabel T_DB.

Tabel 3.13 Tabel T_LISTDB

ATRIBUT	TIPE DATA	PANJANG	NULL	KETERANGAN
Cdb	int	10	NO	Foreign Key
clistdb	int	10	NO	Primary Key
namadb	varchar	50	NO	
aliasdb	varchar	50	NO	Unique Index
ipaddressdb	varchar	20	NO	
portdb	int	11	YES	
userdb	varchar	50	NO	
passdb	varchar	50	YES	

Tabel T_LISTKOLOM digunakan untuk merekam data kolom beserta tipe data dan urutannya. Kolom-kolom ini berasal dari hasil proses *parsing* masukan SQL *query* sebelum melakukan proses penggabungan data.

Tabel 3.14 Tabel T_LISTKOLOM

ATRIBUT	TIPE DATA	PANJANG	NULL	KETERANGAN
clistquery	int	10	NO	Foreign Key
ckolom	int	10	NO	Primary Key
namakolom	varchar	100	NO	
tipedata	varchar	100	NO	
urutankolom	varchar	50	NO	

Tabel T_LISTQUERY digunakan untuk merekam data masukan SQL *query* yang sudah melalui proses *parsing*. Setiap SQL *query* memiliki informasi tentang *database* yang menyimpan tabel-tabel dalam SQL *query* tersebut.

Tabel 3.15 Tabel T_LISTQUERY

ATRIBUT	TIPE DATA	PANJANG	NULL	KETERANGAN
cquery	int	10	NO	Foreign Key
clistdb	int	10	NO	Foreign Key
clistquery	int	10	NO	Primary Key
urutan	int	10	YES	
sqlquery	varchar	500	YES	
operand	varchar	50	YES	
jumkolom	Int	10	YES	
jumrow	int	10	YES	

Tabel T_RESULTSET digunakan untuk merekam data hasil proses penggabungan data.

Tabel 3.16 Tabel T_RESULTSET

ATRIBUT	TIPE DATA	PANJANG	NULL	KETERANGAN
cquery	int	10	NO	Foreign Key
cresultset	int	10	NO	Primary Key
namakolom	varchar	500	YES	
isikolom	varchar	500	YES	

Tabel T_RESULTSET digunakan untuk merekam data hasil proses penggabungan data.

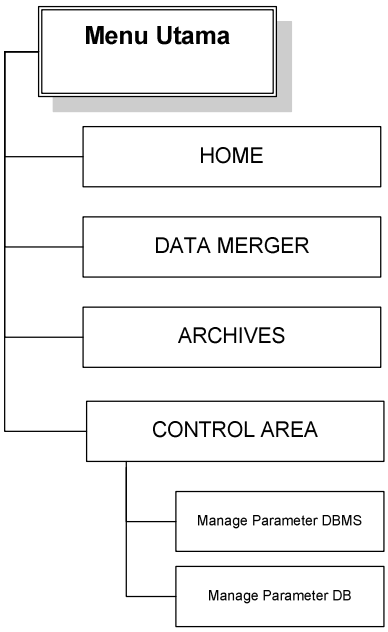
Struktur data lain yang dipakai adalah struktur data *array*, yaitu deretan variabel yang memiliki tipe data sama, struktur data ini digunakan untuk menyimpan tabel transisi pada aturan produksi. Struktur data ini dapat dilihat pada tabel 3.17 di bawah ini.

Tabel 3.17 Struktur Data Array Transisi pada Aturan Produksi

STRUKTUR DATA ARRAY	TIPE DATA
Current State	Text
Input	Text
Next State	Text
keterangan	Text

3.2.2 Perancangan Struktur Menu

Perancangan struktur menu berisikan menu yang berfungsi memudahkan *user* dalam menggunakan aplikasi. Gambar 3.27 menggambarkan struktur menu pada aplikasi yang akan dibangun. Tidak ada proses *login* dan pembagian hak akses dalam aplikasi yang akan dibuat.



Gambar 3.27 Struktur menu aplikasi

Keterangan gambar 3.27 :

- *Home :* Halaman utama aplikasi.
- *Data Merger* (proses penggabungan data) : Menu untuk melakukan penggabungan data.
- *Archives* (*view history* penggabungan data) : Menu untuk melihat data *history* penggabungan data yang sudah dilakukan sebelumnya.
- *Control Area :* Menu untuk melakukan management data dan parameter DBMS dan parameter *database*.

3.2.3 Perancangan Antar Muka

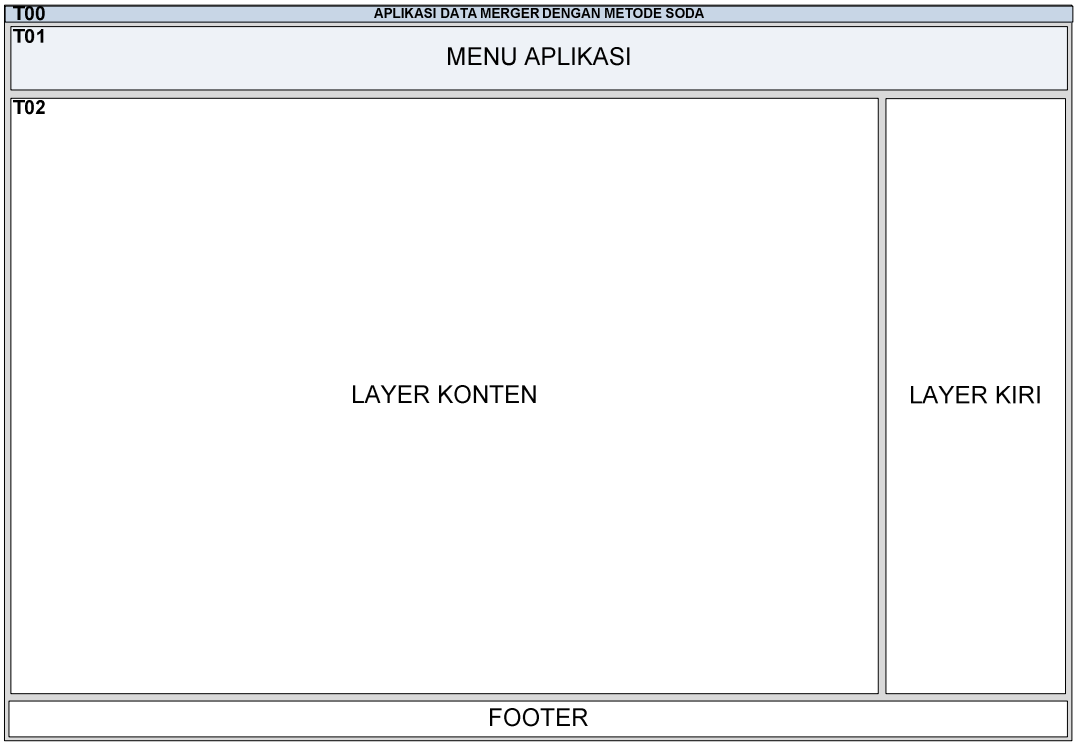
Perancangan antar muka dibuat untuk memberikan gambaran visual tentang aplikasi yang akan dibangun. Sehingga akan mempermudah dalam mengimplementasikan tampilan aplikasi.

3.2.2.1 Perancangan Antar Muka Program Aplikasi

Berikut ini adalah tampilan antar muka yang dirancang pada aplikasi yang akan dibangun.

1. **(T00) Tampilan utama aplikasi**

Gambar berikut ini adalah desain tampilan utama aplikasi.



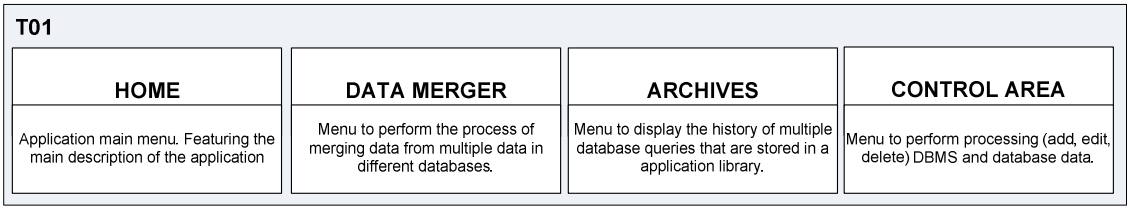
Gambar 3.28 Tampilan Utama

Keterangan gambar 3.28

- **T01** adalah tampilan menu aplikasi.
- **T02** adalah tampilan untuk konten aplikasi berdasarkan menu yang dipilih.

2. (T01) Tampilan menu aplikasi

Gambar berikut ini adalah desain tampilan menu aplikasi.



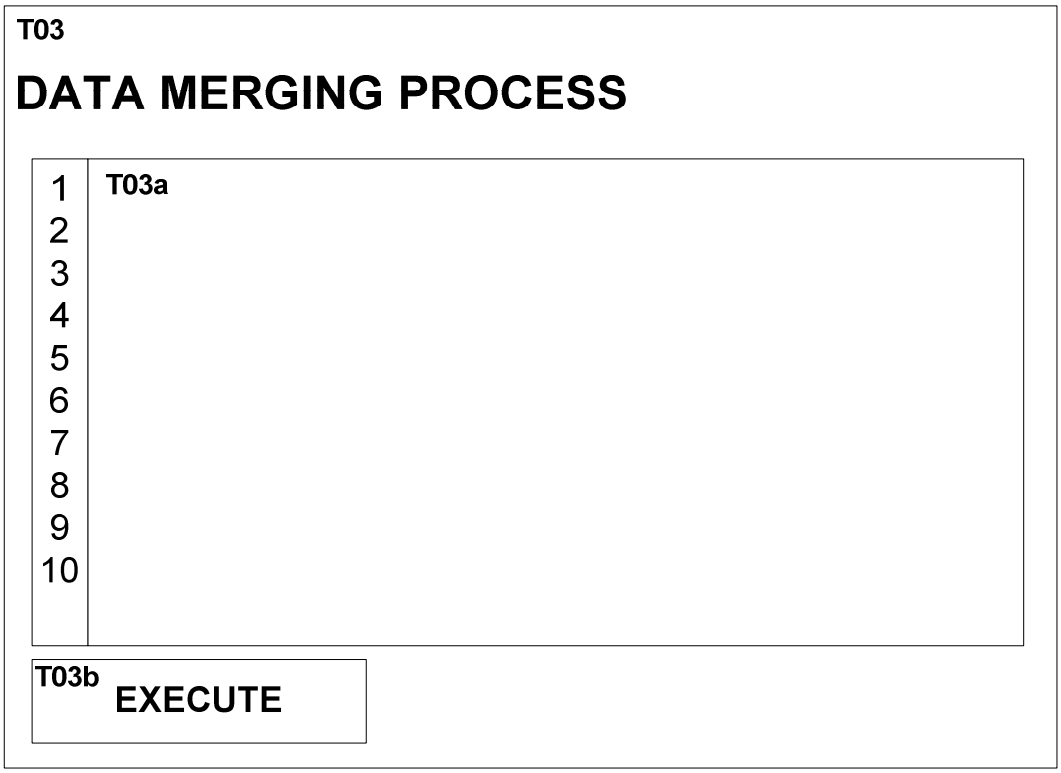
Gambar 3.29 Tampilan Menu Aplikasi

Keterangan gambar 3.29:

- Klik '*HOME'* maka akan muncul tampilan T00 (tampilan utama)
- Klik '*DATA MERGER'* maka akan muncul tampilan T03 (tampilan menu penggabungan data)
- Klik '*ARCHIVES'* maka akan muncul tampilan T04 (tampilan menu daftar arsip penggabungan data)
- Klik '*CONTROL AREA'* maka akan muncul tampilan T05 (tampilan menu *control area* atau manajemen data parameter)

3. **(T03) Tampilan menu penggabungan data (DATA MERGER)**

Gambar berikut ini adalah desain tampilan menu penggabungan data.

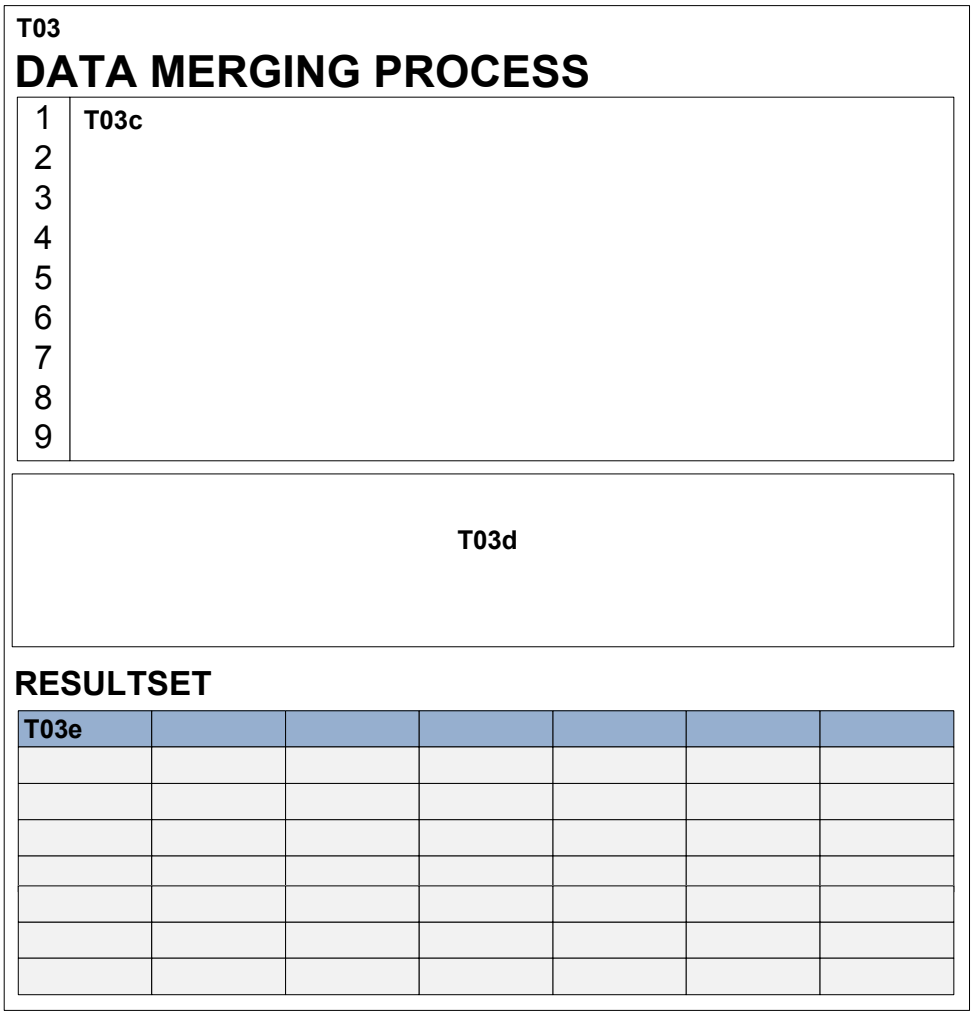


Gambar 3.30 Tampilan menu *Data Merger* (T03)

Keterangan gambar 3.30:

- **T03a** adalah sebuah masukan berupa *textarea* yang digunakan untuk memasukkan *script* SQL *query*.
- **T03b** adalah sebuah tombol (*button*) untuk mengeksekusi *script* SQL *query* di *textarea* T03a. Jika diklik maka akan muncul tampilan T03c dan T03d pada gambar 3.31.

Gambar berikut ini adalah desain tampilan hasil penggabungan data.



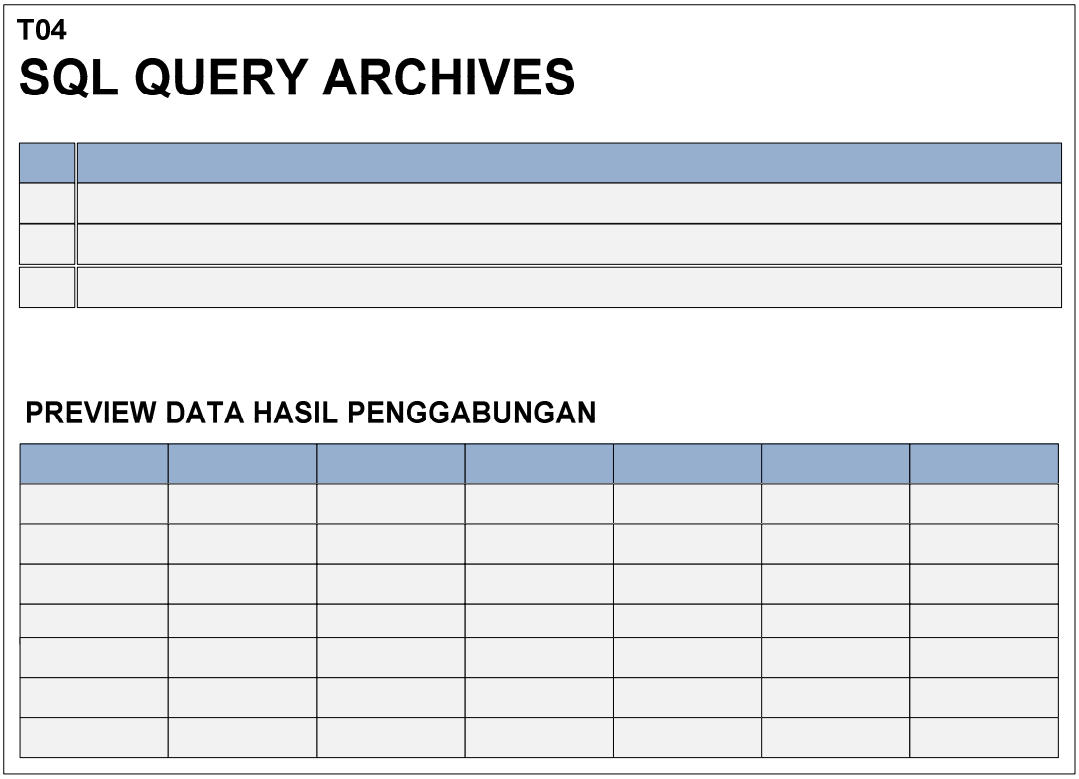
Gambar 3.31 Tampilan Menu *Data Merger* (T03c, T03d, dan T03e)

Keterangan gambar 3.31:

- **T03c** adalah preview dari *script* SQL *query* masukan dari pengguna, dilengkapi dengan *script highlighting* untuk membedakan token-token pada *script* SQL *query*.
- **T03d** adalah tampilan untuk menampilkan pesan *error* (kesalahan) sintaks (jika ada)
- **T03e** adalah sebuah tabel yang menampilkan data hasil penggabungan data.

4. **(T04) Tampilan menu *archives***

Gambar berikut ini adalah desain tampilan menu *archives*.



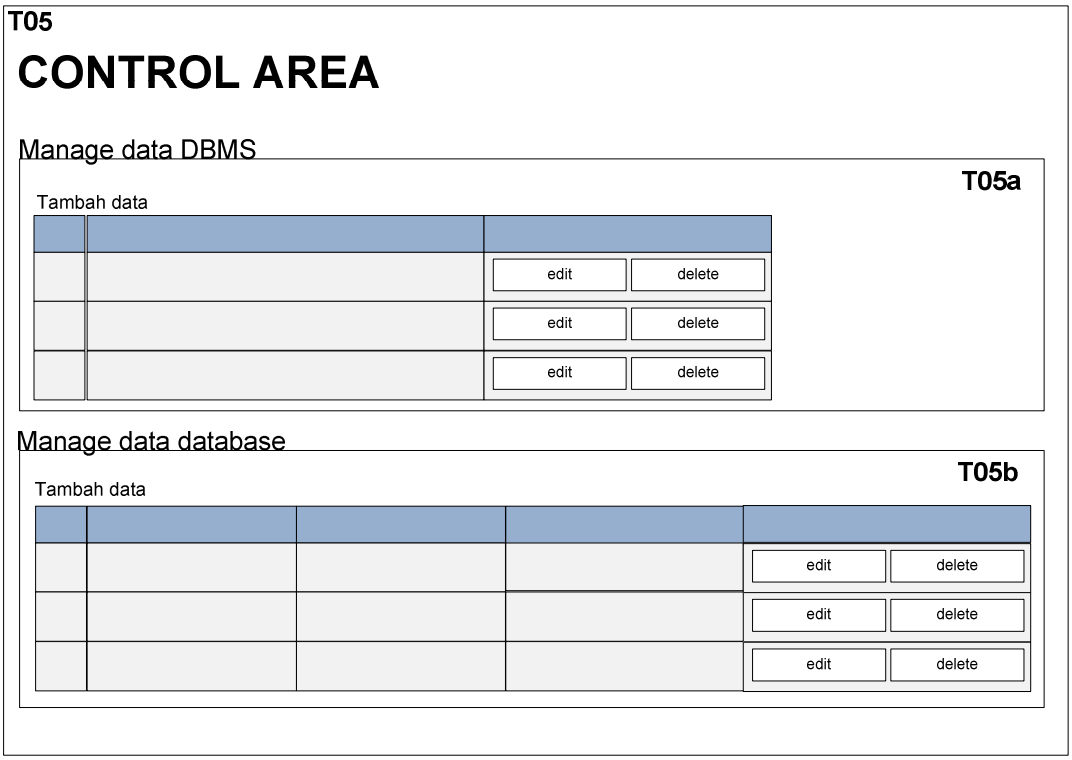
Gambar 3.32 Tampilan menu *Archives* (T04)

Keterangan gambar 3.32:

- **T04a** adalah daftar masukan SQL query yang pernah dimasukkan oleh pengguna aplikasi.
- **T04b** adalah daftar preview dari data hasil penggabungan data

5. **(T05) Tampilan menu *control area***

Gambar berikut ini adalah desain tampilan menu *control area*.



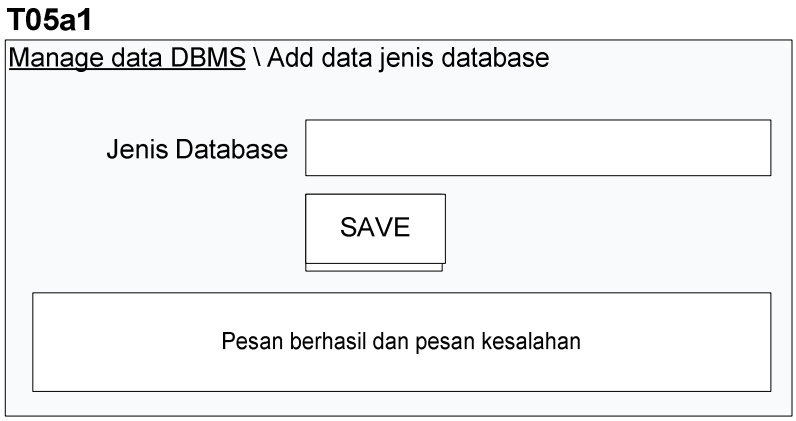
Gambar 3.33 Tampilan menu *Control Area* (T05)

Di dalam menu *control area* terdapat dua macam sub menu yaitu *manage data DBMS* (T05a) dan *manage data database* (T05b).

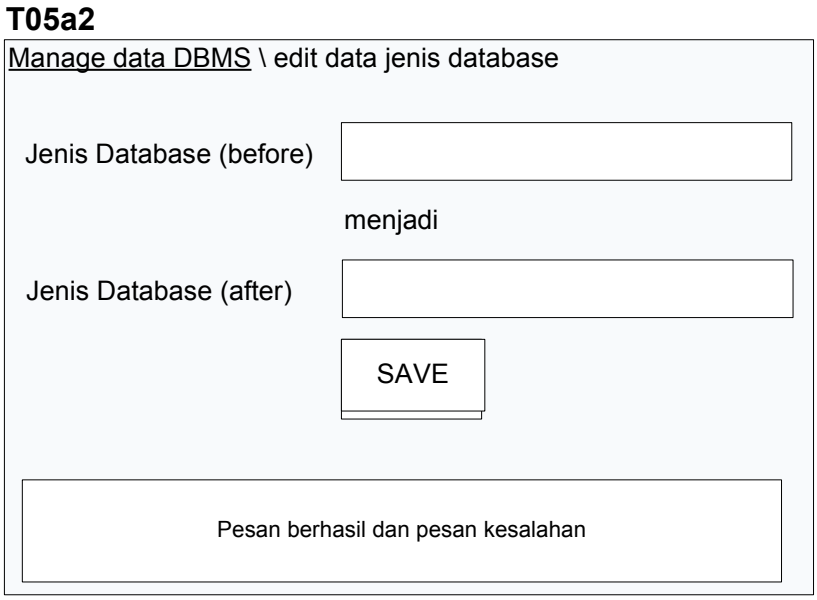
Hyperlink yang ada di dalam T05a meliputi :

- Klik **Tambah data** untuk menuju menu tambah data parameter DBMS yang ada di tampilan **T05a1**
- Klik **edit** untuk menuju menu *edit* data parameter DBMS yang ada di tampilan **T05a2**
- Klik **delete** untuk menuju menu *delete* data parameter DBMS yang ada di tampilan **T05a3**

Desain tampilan T05a1, T05a2, dan T05a3 dapat dilihat pada gambar berikut ini.

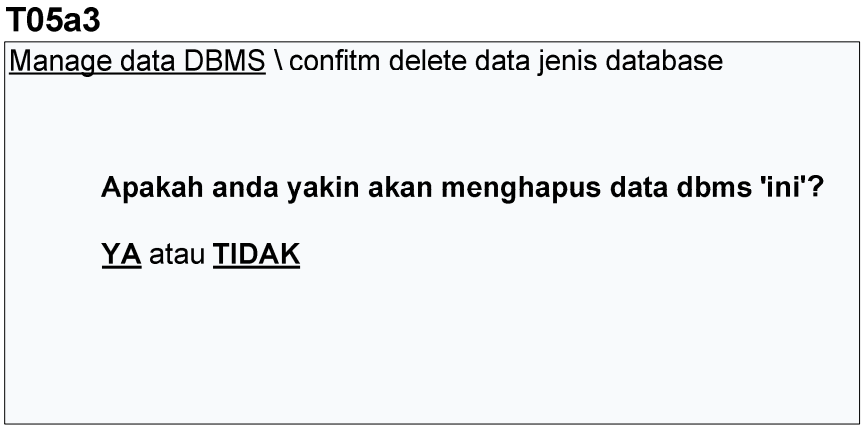


Gambar 3.34 Tampilan menu tambah DBMS (T05a1)



Gambar 3.35 Tampilan menu edit DBMS (T05a2)

Pesan berhasil pada gambar 3.34 dan 3.35 di atas dapat dilihat pada desain tampilan dengan kode M-01 sedangkan pesan kesalahan dapat dilihat pada desain tampilan dengan kode EM-01.



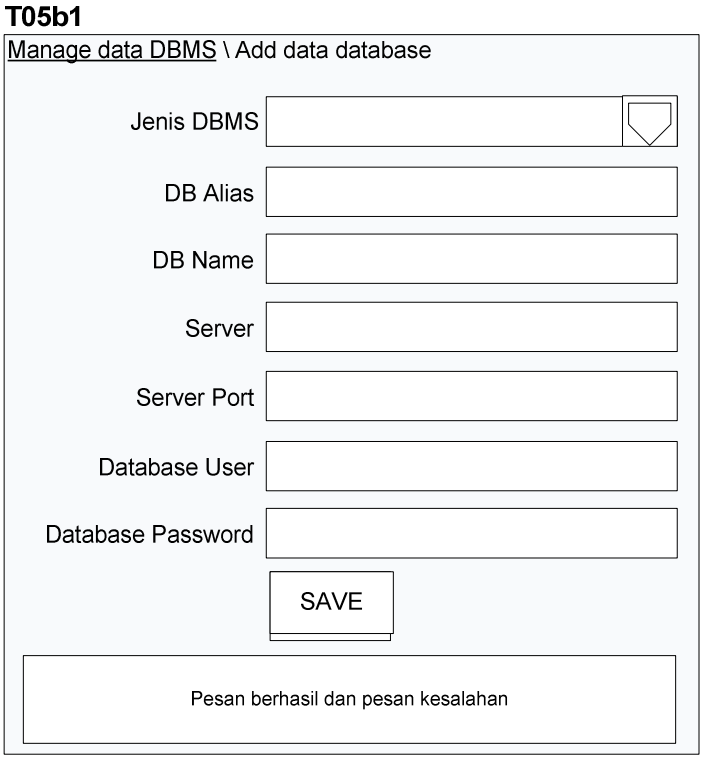
Gambar 3.36 Tampilan menu *delete* DBMS (T05a3)

Jika pengguna mengklik hyperlink YA maka data akan dihapus dan muncul pesan dengan kode M-02, jika pengguna mengklik hyperlink TIDAK maka data tidak akan dihapus dan kembali ke tampilan T05.

Hyperlink yang ada di dalam T05b meliputi:

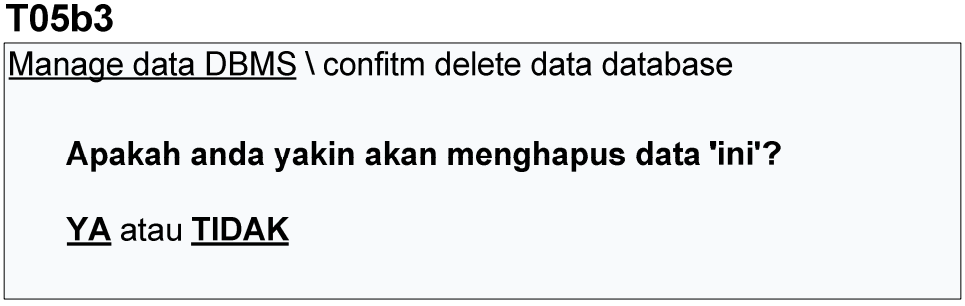
- Klik **Tambah data** untuk menuju menu tambah data parameter *database* yang ada di tampilan **T05b1**
- Klik **edit** untuk menuju menu *edit* data parameter *database* yang ada di tampilan **T05b2**
- Klik **delete** untuk menuju menu *delete* data parameter *database* yang ada di tampilan **T05b3**

Desain tampilan T05b1, T05b2, dan T05b3 dapat dilihat pada gambar 3.37.



Gambar 3.37 Tampilan menu tambah/edit parameter *database* (T05b1/T05b2)

Pesan berhasil pada gambar 3.37 di atas dapat dilihat pada desain tampilan dengan kode M-01 sedangkan pesan kesalahan dapat dilihat pada desain tampilan dengan kode EM-01.

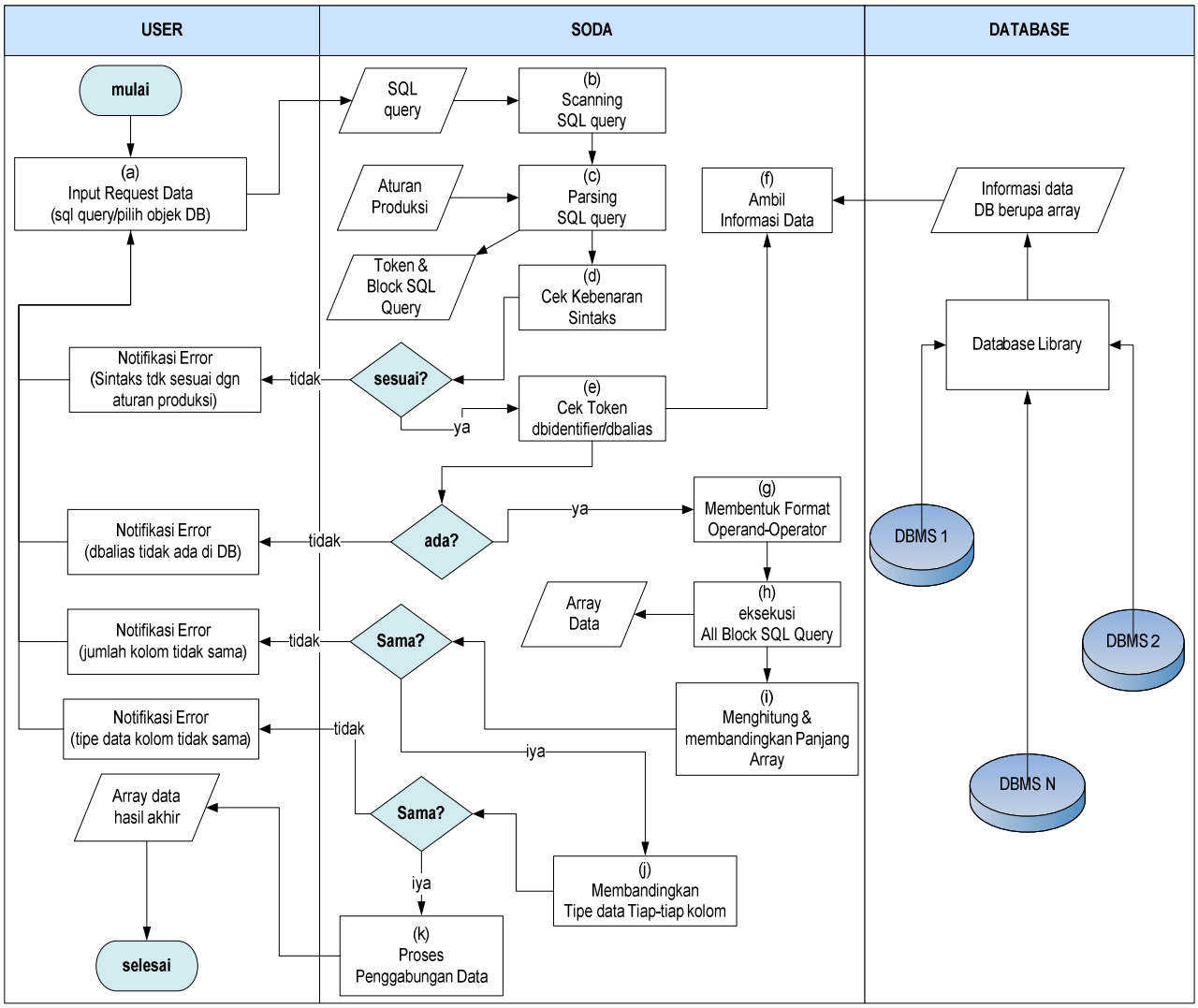


Gambar 3.38 Tampilan menu delete parameter *database* (T05b3)

Jika pengguna mengklik hyperlink YA maka data akan dihapus dan muncul pesan dengan kode M-02, jika pengguna mengklik hyperlink TIDAK maka data tidak akan dihapus dan kembali ke tampilan T05.

3.2.4 Perancangan Prosedural

Perancangan prosedural ini menjelaskan prosedur proses penggabungan data yang ada di dalam aplikasi yang akan dibangun. Untuk lebih jelasnya, digambarkan dalam *swimlane diagram* di bawah ini.



Gambar 3.39 Prosedur proses penggabungan data