BAB II

LANDASAN TEORI

2.1 DBMS (*Database Management System*)

DBMS merupakan perangkat lunak yang dirancang untuk dapat melakukan utilisasi dan mengelola koleksi data dalam jumah yang besar. DBMS juga dirancang untuk dapat melakukan manipulasi data secara lebih mudah. Sebelum adanya DBMS maka data pada umumnya disimpan dalam bentuk *flat file*, yaitu file teks yang ada pada sistem operasi. Sampai sekarangpun masih ada aplikasi yang menyimpan data dalam bentuk flat secara langsung. Menyimpan data dalam bentuk *flat file* mempunyai kelebihan dan kekurangan. Penyimpanan dalam bentuk ini akan mempunyai manfaat yang optimal jika ukuran filenya relatif kecil, seperti file password. File password pada umumnya hanya digunakan untuk menyimpan nama yang jumlahnya tidak lebih dari seribu orang. Selain dalam bentuk flat file, penyimpanan data juga dapat dilakukan dengan menggunakan program bantu seperti *spreadsheet*. Penggunaan perangkat lunak ini memperbaiki beberapa kelemahan dari *flat file*, seperti bertambahnya kecepatan dalam pengolahan data. Namun demikian metode ini masih memiliki banyak kelemahan, diantaranya adalah masalah manajemen dan keamanan data yang masih kurang.

2.1.1 Pengertian DBMS

Berikut ini adalah beberapa pengertian tentang DBMS menurut para ahli di bidang database.

1. Menurut C.J. Date : DBMS adalah merupakan *software* yang menghandel seluruh akses pada *database* untuk melayani kebutuhan *user*.
2. Menurut S, Attre : DBMS adalah *software*, *hardware*, *firmware* dan prosedur-prosedur yang mengelola *database*. *Firmware* adalah *software* yang telah menjadi modul yang tertanam pada *hardware* (ROM).
3. Menurut Gordon C. Everest : DBMS adalah manajemen yang efektif untuk mengorganisasi sumber daya data.

Jadi dapat disimpulkan bahwa DBMS adalah semua peralatan komputer (*Hardware*, *Software*, dan *Firmware*) yang dilengkapi dengan bahasa yang berorientasi pada data (*High level data language*) yang sering disebut juga sebagai bahasa generasi ke-4 (*fourth generation language*). Peralatan untuk menetapkan atau menentukan pendekatan *database* disebut DBMS. DBMS merupakan *software* (dan *hardware*) yang didesain khusus untuk melindungi dan mengelola *database*.

2.1.2 Fungsi dan Kegunaan DBMS

DBMS memiliki beberapa fungsi diantaranya sebagai berikut:

1. Definisi data dan hubungannya
2. Memanipulasi data
3. Keamanan dan integritas data

4. *Recovery* atau perbaikan dan *concurency* data
5. *Data dictionary*
6. Unjuk kerja atau *performance*

Kegunaan DBMS antara lain :

1. Mendefinisikan data dan hubungannya.
2. Mendokumentasikan struktur dan definisi data
3. Menggambarkan, mengorganisasikan dan menyimpan data untuk akses yang selektif dan efisien.
4. Hubungan yang sesuai antara *user* dengan sumber daya data.
5. Perlindungan terhadap sumber daya data akan terjamin, dapat diandalkan, konsisten dan benar.
6. Memisahkan masalah *logical* dan *physical* sehingga mengubah implementasi *database* secara fisik tidak menghendaki *user* untuk mengubah maksud data (*Logical*).
7. Menentukan pembagian data kepada para *user* untuk mengakses secara *concurent* pada sumber daya data.

2.1.3 Keuntungan Dan Kerugian Penggunaan DBMS

Beberapa keuntungan penggunaan DBMS adalah sebagai berikut:

1. Kebebasan data dan akses yang efisien
2. Mereduksi waktu pengembangan aplikasi
3. *Integritas* dan keamanan data
4. Administrasi keseragaman data
5. Akses bersamaan dan perbaikan dari terjadinya *crashes* (tabrakan/*error* dari proses serentak).

6. Mengurangi data *redundancy* : Data redundansi dapat direduksi atau dikurangi, tetapi tidak dapat dihilangkan sama sekali (untuk kepentingan *keyfield*)

Beberapa kerugian penggunaan DBMS adalah sebagai berikut:

1. Memperoleh perangkat lunak yang mahal (teknologi DBMS, *Operation*, *Conversion*, *Planning*, *Risk*). DBMS *mainframe* masih sangat mahal. DBMS berbasis mikro biayanya mencapai beberapa ratus dolar, dapat menggambarkan suatu organisasi yang kecil secara berarti.
2. Memperoleh konfigurasi perangkat keras yang besar. DBMS sering memerlukan kapasitas penyimpanan primer dan sekunder yang lebih besar daripada yang diperlukan oleh program aplikasi lain. Juga, kemudahan yang dibuat oleh DBMS dalam mengambil informasi mendorong lebih banyak terminal pemakai yang disertakan dalam konfigurasi daripada jika sebaliknya.
3. Mempekerjakan dan mempertahankan seorang DBA DBMS memerlukan pengetahuan khusus agar dapat memanfaatkan kemampuan secara penuh. Pengetahuan khusus ini paling baik diberikan oleh pengelola *database*.

2.1.4 Macam-macam DBMS

Banyak sekali DBMS yang dikembangkan saat ini untuk mengimbangi kebutuhan pengguna akan manajemen basis data. Beberapa DBMS yang sering dipakai oleh pengguna adalah sebagai berikut:

1. **Microsoft SQL Server**

   Microsoft SQL Server adalah program Sistem Manajemen Basis Data Relasional. Susunan dari Microsoft SQL Server dibagi menjadi tiga komponen:

   a. SQL OS yang melakukan layanan utama pada SQL Server, misalnya mengatur aktifitas, pengaturan memori, dan pengaturan *Input* atau *Output*;

   b. *Relational Engine* yang bekerja sebagai penghubung komponen *database*, *table*, *query*, dan perintah tersimpan;

   c. *Protocol Layer* yang mengatur fungsi-fungsi SQL Server.

2. **Oracle**

   Oracle adalah salah satu *software* sistem manajemen basis data relasional yang cukup diminati, dikenal, dan dipakai saat ini. *Software* ini biasa digunakan untuk pengaksesan data yang dilakukan secara online dan sangat baik dalam pengelolaan data dalam jumlah *record* yang sangat besar. Oracle adalah *Relational Database Management System* (RDBMS) untuk mengelola informasi secara terbuka, komprehensif dan terintegrasi. Oracle merupakan DBMS yang dirancang khusus untuk organisasi berukuran besar, bukan untuk ukuran kecil dan menengah. Kebutuhan organisasi berukuran besar tidaklah sama dengan organisasi yang kecil atau menengah yang tidak

akan berkembang menjadi besar. Organisasi yang berukuran besar membutuhkan fleksibilitas dan skalabilitas agar dapat memenuhi tuntutan akan data dan informasi yang bervolume besar dan terus menerus bertambah besar.

3. MySQL

MySQL adalah perangkat lunak sistem manajemen basis data yang diciptakan untuk dapat dilakukan instalasi secara gratis (*open source*). MySQL AB membuat MySQL tersedia sebagai perangkat lunak gratis di bawah lisensi GNU *General Public Licency* (GPL), tetapi mereka juga menjual di bawah lisensi komersial untuk kasus-kasus dimana penggunaannya tidak cocok dengan penggunaan GPL.

Beberapa contoh DBMS lainnya : Microsoft Visual Fox Pro, dBase, DB2, Microsoft Office Access, Ingres, Arago, Force, dbFast, dbXL, Quicksilver, Clipper, Xbase++, Flagship, Codebase, Harbour/Xharbour.

2.2 *Data Agregation* dan *Data Integration*

Data Agregation adalah sebuah sistem atau konsep yang digunakan untuk menghubungkan berbagai macam *database* atau lebih dari satu *database* ke dalam satu aplikasi. Perbedaan mendasar dari kedua sistem tersebut adalah *data agregation* tidak menggunakan *repository*, sedangkan *data integrasi* dalam implementasinya menggunakan *repository*.

2.2.1 *Data Agregation*

Data agregation mengambil konsep dari *agregasi*, yakni suatu konsep yang menghubungkan lebih dari satu entitas. Apabila dalam implementasi suatu aplikasi pengambilan data yang berasal dari *database* dengan lokasi yang berbeda beda, pengguna aplikasi tidak perlu tahu lokasi fisik data tersebut disimpan.

Keuntungan dari *Data Agregation* adalah :

1. Menghemat tempat penyimpanan data, tidak perlu *repository*
2. Tidak perlu sinkronisasi, karena akses langsung terhadap *data source.*
3. Data akan selalu *real time*

Namun konsep *Data Agregation* memiliki kelemahan antara lain :

1. Data yang digunakan tidak normal, berbeda-beda tergantung pada sistem yang ada pada *data source*-nya.
2. Mengurangi efektifitas dan kecepatan akses.
3. Sedikit banyaknya akan merubah struktur *query*.

Agregasi DBMS dapat dilakukan pada level *SQL Query*, artinya DBMS yang dapat diagregasikan juga terbatas pada DBMS yang sudah mengimplementasikan standar *query*.

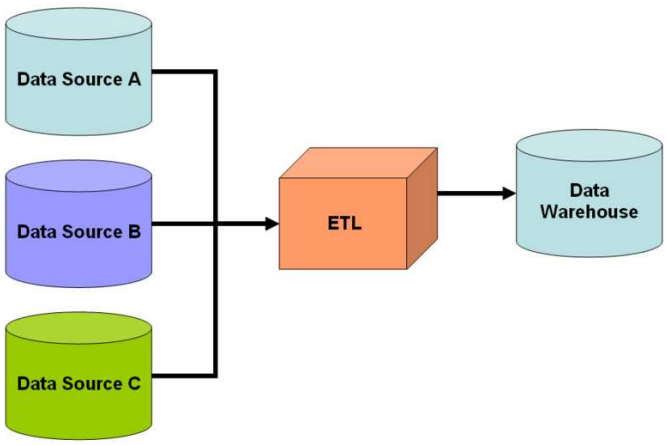
2.2.2 *Data Integration*

Sampai saat ini, definisi *data integration* masih abstrak, namun secara garis besarnya *data integration* adalah solusi umum untuk menyelesaikan permasalahan *heterogenous database*. *Data integration* biasanya diimplementasikan ke dalam suatu aplikasi, konsep *data integration* umumnya mengambil data dari *data source* yang berbeda-beda untuk kemudian di *store* dalam satu *database*

repository, karenanya lahirlah konsep *data integration*. Kemampuan *data integration* masih terbatas pada *Extract, Transform, Load.*

Data Integration System secara formal didefinisikan sebagai tripel [G,S,M], dimana G adalah global skema. S adalah *heterogenous data source*, dan M adalah pemetaan yang memetakan permintaan antara sumber dan skema global. Baik G dan S dinyatakan dalam bahasa lebih abjad terdiri dari simbol-simbol untuk setiap hubungan masing-masing. Pemetaan M terdiri dari pernyataan antara G dan pertanyaan atas pertanyaan di atas S. Sedangkan untuk implementasi *data integration* dalam aplikasi ditandai dengan kode ETL. Elemen dari integrasi data diantaranya adalah sebagai berikut:

- perpindahan data secara *realtime*
- *transformasi* atau perubahan bentuk
- sinkronisasi dan kualitas data
- manajemen data dan *data services*



Gambar 2.1 Konsep *Data Integration*

Kelebihan dari konsep *data integration* adalah

1. Data yang ditaruh dalam *repository* sudah diperbaharui dan dinormalisasikan sesuai dengan kebutuhan penggunanya.
2. Akses ke dalam data dapat lebih cepat, karena user langsung mengakses ke dalam *repository.*

Kekurangannya adalah

1. Data tidak sepenuhnya *realtime*.
2. Sistem perlu melakukan sinkronisasi dengan *data source* yang sebenarnya.
3. Karena menggunakan *repository* maka *data integration* membutuhkan DBMS *temporary*

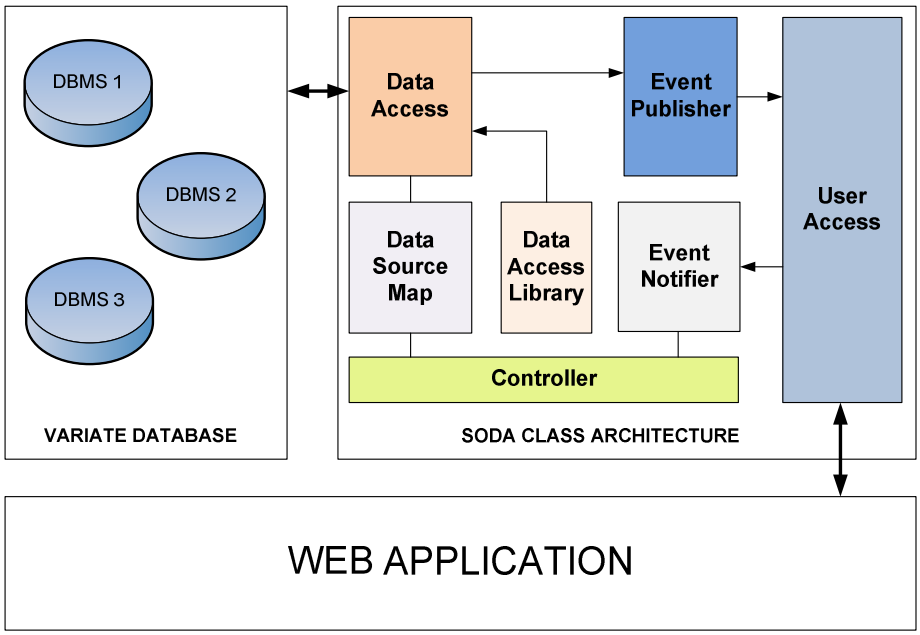
2.3 SODA (*Simple Object Database Access*)

SODA (*Simple Object Database Access*) adalah metode yang dapat digunakan untuk berkomunikasi dengan DBMS. Metode ini bertujuan untuk menyatukan komponen untuk mengakses lebih dari satu DBMS, membuatnya lebih sederhana dan lebih terstruktur.

SODA dapat diimplementasikan dalam sebuah *class* yang nantinya dapat digunakan dalam beberapa aplikasi berbasis objek, dan dapat dibentuk ke dalam sebuah prosedur untuk diimplementasikan ke dalam aplikasi berbasis prosedural. Komponen dasar dari SODA terdiri dari :

1. *User Access*
2. *Event Notifier*
3. *Controller*

4. *Data Source Map*
5. *Data Access Library*
6. *Data Access*
7. *Event Publisher*



Gambar 2.2 Ilustrasi SODA pada *web application* dengan *multiple* DBMS

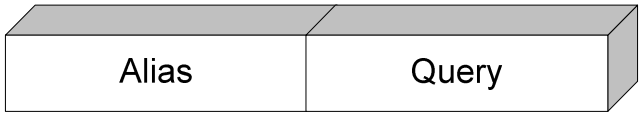
Komponen-komponen yang mendukung SODA pada gambar di atas dijelaskan sebagai berikut :

1. User Access

User access adalah komponen bagian depan dari *data agregator* yang digunakan untuk membaca dan mengkategorikan masukan. *User access* juga berfungsi sebagai keluaran data hasil eksekusi *query data agregator* dengan user dalam hal ini adalah aplikasi.

Komponen ini memiliki format yang telah ditentukan sebelumnya baik untuk masukan maupun keluaran, tujuan format tersebut adalah agar algoritma dibelakangnya dapat membaca dan mengkategorikan inputan.

Bentuk masukan pada komponen ini digambarkan sebagai berikut:

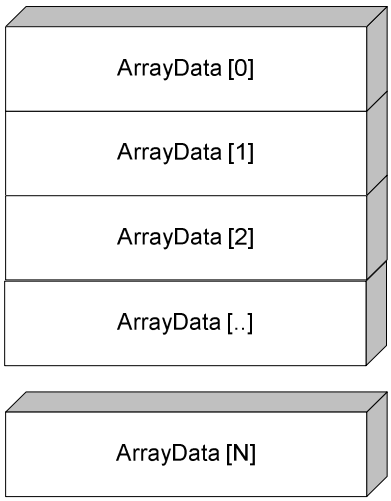


Gambar 2.3 Ilustrasi masukan

Alias : jenis alias koneksi ke DBMS yang digunakan

Query : SQL *query select* data dari *table* atau *table-table* pada *database*

Format keluarannya adalah *array* hasil eksekusi *query* berupa data yang didefinisikan dalam SQL *query*.



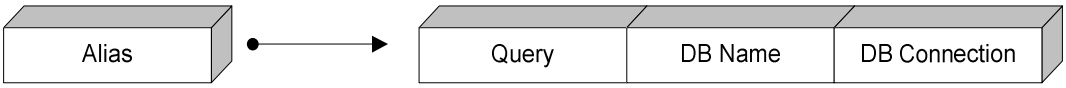
Gambar 2.4 Ilustrasi *array* data hasil keluaran

ArrayData : Data hasil *query* pada baris ke-0 s/d ke-N

2. Event Notifier

Event notifier adalah komponen bagian depan yang digunakan untuk merubah format dari format standart input kedalam format yang lebih dikenal oleh *controller* sesuai dengan *data source map*. Komponen ini sangat membantu memudahkan *controller* untuk membaca masukan dengan cara memisahkan komponen masukan ke dalam beberapa bagian.

Ilustrasinya dapat digambarkan sebagai berikut:



Gambar 2.5 Pemecahan format masukan

Alias : jenis alias koneksi *database* yang digunakan

Query : SQL *query select* data dari *table* atau *view* pada *database*

DB Name : nama *database* yang diakses

DB Connection : jenis koneksi *database* yang digunakan

Perubahan format ini dimaksudkan agar komponen *controller* dapat menemukan lokasi, jenis koneksi dan nama *database* yang akan diakses dengan menggunakan entitas *database_name* dan menyesuaikannya dengan *data source map* sehingga dapat ditentukan *data access* mana yang akan digunakan pada *data access library.*

3. Controller

Controller adalah komponen terpenting dalam *Simple Object Database Access*, dimaksudkan sebagai sentral dan letak dari implementasi algoritma pada *Simple Object Database Access*. *Controller* bertugas membaca inputan SQL *query* setelah di transformasi dan mencari lokasi, jenis dan nama *database* dalam *data source map* berdasarkan entitas aliasnya. Untuk kemudian memutuskan dengan *data access* yang mana DBMS tersebut akan diakses.

4. Data Source Map

Data source map adalah komponen yang memetakan bentuk, lokasi dan jenis *database*. *Data source map* memiliki format yang sudah didefinisikan, sehingga *user* bisa dengan bebas menambahkan lokasi, jenis DBMS dan nama *database* yang akan diakses dengan menambahkan indeks ke dalam masing-masing *source* yang ditambahkan.

5. Data Access Library

Data access library adalah komponen yang menyimpan tata cara akses ke dalam DBMS dengan cara yang sudah didefinisikan sebelumnya. *Data access library* dapat ditambahkan sesuai dengan koneksi yang akan diakses sesuai dengan format yang digunakan oleh *controller*. Misalnya adalah cara akses ke mysql, cara akses ke sql-server,cara akses ke oracle.

6. Data Access

Data access adalah komponen yang umum digunakan dalam mengakses *database*. *Data access* secara global disimpan dalam *data access library*. Di dalam SODA, *data access* dikendalikan oleh *controller* karena jenis DBMS yang akan diakses dapat bermacam-macam.

7. Event Publisher

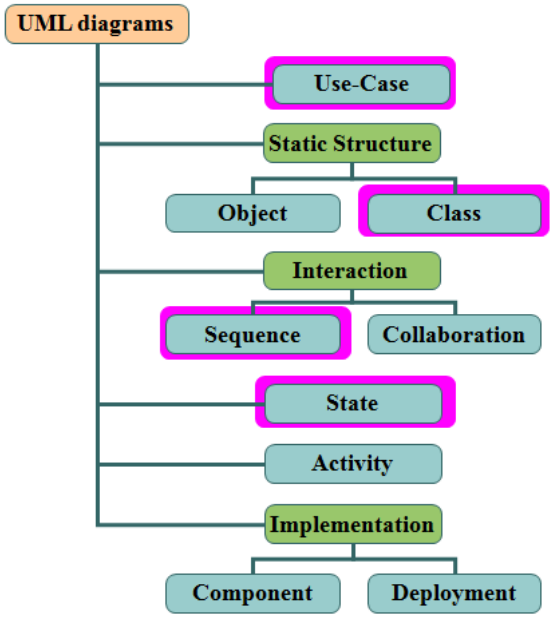
Event publisher adalah komponen bagian depan yang digunakan untuk meneruskan keluaran dari *data access* kepada *user access*. *Event publisher* memiliki format yang sama dengan keluaran hasil eksekusi *query* pada *user access*. *Event publisher* dapat juga dimanfaatkan sebagai *transformator* dari *data access* ke dalam sebuah *array* dengan format keluaran.

2.4 UML (*Unified Modelling Language*)

UML adalah bahasa untuk spesifikasi, visualisasi, konstruksi dan dokumentasi pembuatan perangkat lunak yang berorientasi objek. UML dapat membantu kita untuk :

1. **Memudahkan berpikir dan mendokumentasikan sistem sebelum mengimplementasikannya**
2. **Merencanakan dan menganalisa logika atau perilaku sistem**
3. **Membuat keputusan yang benar sedini mungkin (sebelum melangkah ke tahap *coding*)**
4. **Lebih mudah untuk memodifikasi atau mengelola sistem yang terdokumentasi dengan baik**

Diagram UML digambarkan dalam *tree* di bawah ini :



Gambar 2.6 Diagram UML

Dalam gambar 2.6 terdapat banyak macam diagram UML seperti :

1. *Use Case (relation of actors to system functions)*
2. *Class* (*static class structure*)
3. *Object* (*same as class - only using class instances – i.e. objects*)
4. *State* (*states of objects in a particular class*)
5. *Sequence (Object message passing structure)*
6. *Collaboration (same as sequence but also shows context - i.e. objects and their relationships)*
7. *Activity (sequential flow of activities i.e. action states)*
8. *Component (code structure)*
9. *Deployment (mapping of software to hardware)*

2.5 Teori Bahasa dan Otomata

Arti menurut *American Heritage Dictionary*:

1. *a robot*
2. *one that behaves in an automatic or mechanical fashion*

Arti dalam dunia matematika

Berkaitan dengan teori mesin abstrak, yaitu mesin sekuensial yang menerima input, dan mengeluarkan output, dalam bentuk diskrit.

Contoh :

1. Mesin Jaja / *vending machine*
2. Kunci kombinasi
3. *Parser/compiler*

Teori Otomata dan bahasa formal, berkaitan dalam hal :

1. Pembangkitan kalimat / *generation* : menghasilkan semua kalimat dalam bahasa L berdasarkan aturan yang dimilikinya
2. Pengenalan kalimat / *recognition* : menentukan suatu string (kalimat) termasuk sebagai salah satu anggota himpunan L.

2.5.1 Bahasa Formal

Suatu kalimat dibentuk dengan menerapkan serangkaian aturan produksi pada sebuah simbol ‘akar’. Proses penerapan aturan produksi dapat digambarkan sebagai suatu diagram pohon.

Teori dasar dari bahasa formal adalah sebagai berikut:

1. Def. 1 : Sebuah string dengan panjang n yang dibentuk dari himpunan A adalah barisan dari n simbol

   $a_1 a_2 \ldots a_n \qquad a_i \in A$

   Panjang string x dituliskan dengan |x|
2. Def 2 : String kosong (null string), dilambangkan dengan ε adalah untaian dengan panjang 0 dan tidak berisi apapun. Panjang string x dituliskan dengan |x|
3. Def 3 : Dua buah string $a = a_1 a_2 \ldots a_m$ dan $b = b_1 b_2 \ldots b_n$ dapat disambungkan menjadi string c dengan panjang m+n sebagai berikut $c = a_1 a_2 \ldots a_m b_1 b_2 \ldots b_n$. Operasi penyambungan tersebut dapat pula diterapkan pada himpunan

   $$Z = XY = \{st \mid s \in X \wedge t \in Y\}$$

4. Def 4 : (*Closure*) . A^n adalah himpunan string dengan panjang n yang dibentuk dari simbol-simbol di himpunan simbol/alfabet A: Transitif *Closure* atau *Kleen Closure* adalah himpunan seluruh string yang dapat dibentuk dari A dengan berbagai panjang

$A^* = A^0 \cup A^1 \cup A^2 \cup A^3 \cup \ldots$ Jika string kosong dikeluarkan , akan diperoleh *positive closure* $A^+ = A^1 \cup A^2 \cup A^3 \cup \ldots$

2.5.2 Tata Bahasa

Aturan yang disebutkan pada proses pengenalan dan pembangkitan kalimat. Secara formal, tata bahasa terdiri dari 4 komponen yaitu :

1. Himpunan berhingga, tidak kosong dari simbol-simbol non terminal T^1
2. Himpunan berhingga, dari simbol-simbol non-terminal N
3. Simbol awal S ∈ N, yang merupakan salah satu anggota dari himpunan simbol nonterminal.
4. Himpunan berhingga aturan produksi P yang setiap elemennya dituliskan dalam bentuk : α → β dimana α dan β adalah string yang dibentuk dari himpunan $T \cup N$ dan α harus berisi paling sedikit satu simbol non-terminal.

Keempat komponen tersebut sering dituliskan sbb : *G = (T,N,S,P).* Bahasa yang dihasilkan oleh G ditulis sebagai L(G), yaitu himpunan string yang dapat diturunkan dari simbol awal S dengan menerapkan aturan-aturan produksi yang terdapat pada P.

2.5.3 Aturan Produksi

Aturan produksi $\alpha \rightarrow \beta$ yang diterapkan pada suatu string $w=a\alpha c$ mengganti kemunculan α menjadi β, sehingga string tersebut berubah menjadi $w=a\beta c$, sehingga dapat dituliskan $a\alpha c \Rightarrow a\beta c$ ($a\alpha c$ memproduksi $a\beta c$).

Produksi tersebut dapat diterapkan berkali-kali

$$w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \ldots \Rightarrow w_n$$

atau dapat dituliskan

$$w_1 \Rightarrow^* w_n$$

jika minimal harus ada 1 aturan produksi yang diterapkan :

$$w_1 \Rightarrow^+ w_n$$

Contoh 1

Tatabahasa G = {{S} , {a,b}, S , P } dengan aturan produksi P adalah

$S \rightarrow aSb$

$S \rightarrow \varepsilon$

maka dapat dihasilkan suatu string $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$

sehingga dapat dituliskan $S \Rightarrow^* aabb$

Bahasa yang dihasilkan dari tatabahasa tersebut adalah

$L(G) = \{ \varepsilon , ab, aabb , aaabbb , aaaabbbb, ... \}$

atau dapat pula dituliskan

$L(G) = \{a^n b^n \mid n \geq 0 \}$

Contoh 2

Tatabahasa G = {{S,A} , {a,b}, S , P } dengan aturan produksi P adalah

$S \rightarrow Ab$

$A \rightarrow aAb$

$A \rightarrow \varepsilon$

maka dapat dihasilkan suatu *string*

$S \Rightarrow Ab \Rightarrow b$

$S \Rightarrow Ab \Rightarrow aAbb \Rightarrow abb$

$S \Rightarrow Ab \Rightarrow aAbb \Rightarrow aaAbbb \Rightarrow aaAbbb$

Bahasa yang dihasilkan dari tatabahasa tersebut adalah

$L(G) = \{ b , abb, aabbb , aaabbbb , aaaabbbbb, ... \}$

atau dapat pula dituliskan

$L(G) = \{a^n b^{n+1} \mid n \geq 0 \}$

Hirarki Bahasa

Kelas	Mesin pengenal
Regular language	Finite State Automata
Context free language	Push Down Automata
Context sensitive language	Linear Bounded Automata
Unrestricted language	Turing Machine

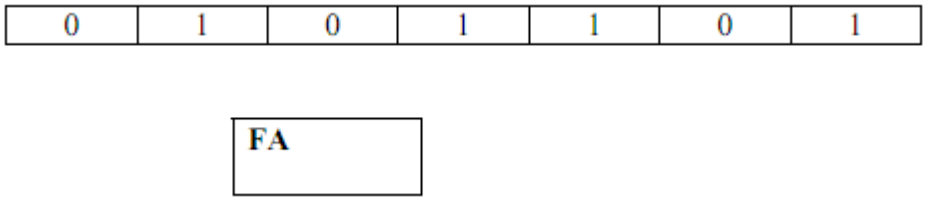
Kelas	Ruas kiri	Ruas Kanan	Contoh
Regular	$\alpha \in N$	≤ 1 non terminal (paling kiri/kanan)	$P \rightarrow abR$ $Q \rightarrow abc$ $R \rightarrow Scac$
Context free	$\alpha \in N$	-	$P \rightarrow aQb$ $Q \rightarrow abPRS$
Context sensitive	$\alpha \in (T \cup N)^+$	$\lvert\alpha\rvert \leq \lvert\beta\rvert$	$aD \rightarrow Da$ $AD \rightarrow aCD$
Unrestricted	$\alpha \in (T \cup N)^+$	-	$CB \rightarrow DB$ $ADc \rightarrow \varepsilon$

NB : Ruas kiri harus memuat simbol non-terminal

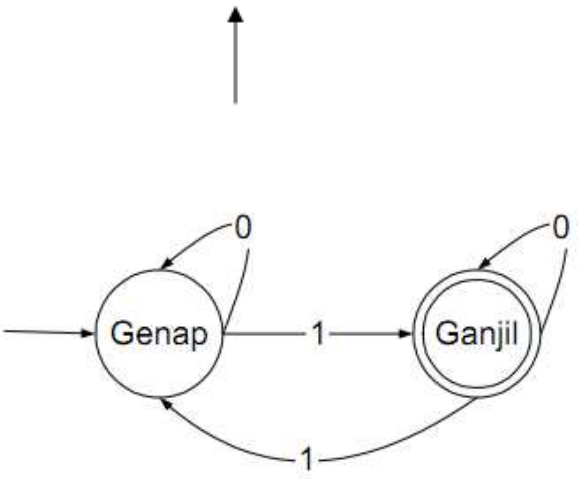
2.5.4 *Finite State Automata* (FSA)

Ciri-ciri dari *Finite State Automata* diantaranya adalah

1. Model matematika yang dapat menerima *input* dan mengeluarkan *output*
2. Memiliki *state* yang berhingga banyaknya dan dapat berpindah dari satu *state* ke *state* lainnya berdasar input dan fungsi transisi
3. Tidak memiliki tempat penyimpanan atau *memory*, hanya bisa mengingat *state* terkini.
4. Mekanisme kerja dapat diaplikasikan pada : *elevator*, *text editor*, analisa leksikal, pengecek *parity*.



Contoh pengecek *parity* ganjil



Misal input : **1101**

Genap 1 Ganjil 1 Genap 0 Genap 1 Ganjil

diterima mesin

Misal input : **1100**

Genap 1 Ganjil 1 Genap 0 Genap 0 Genap

ditolak mesin

Finite State Automata dinyatakan oleh lima *tuple*

$M = (Q , \Sigma , \delta , S , F)$

Q = himpunan state

Σ = himpunan simbol input

δ = fungsi transisi $\delta : Q \times \Sigma$

S = *state* awal / *initial state* , $S \in Q$

F = *state* akhir, $F \subseteq Q$

Pada contoh diatas,

Q = {Genap, Ganjil}

Σ = {0,1}

S = Genap

F = {Ganjil }

δ	0	1
Genap	Genap	Ganjil
Ganjil	Ganjil	Genap

atau

δ(Genap,0) = Genap

δ(Genap,1) = Ganjil

δ(Ganjil,0) = Ganjil

δ(Ganjil,1) = Genap

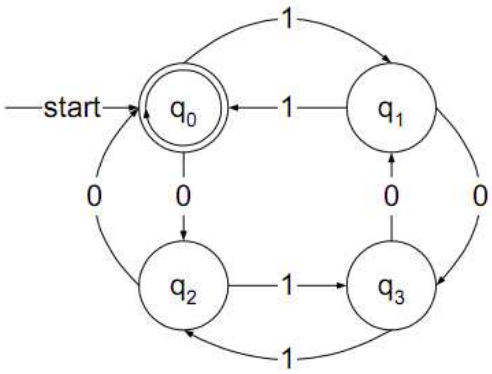
2.5.5 Jenis FSA

Deterministic Finite Automata (DFA) : dari suatu *state* ada tepat satu *state* berikutnya untuk setiap simbol masukan yang diterima.

Non-deterministic Finite Automata (NFA) : dari suatu *state* ada 0, 1 atau lebih *state* berikutnya untuk setiap simbol masukan yang diterima.

2.5.5.1 *Deterministic Finite Automata*

1. Contoh : pengujian *parity* ganjil.
2. Contoh lain : Pengujian untuk menerima bit *string* dengan banyaknya 0 genap, serta banyaknya 1 genap.
3. 0011 : diterima.
4. 10010 : ditolak, karena banyaknya 0 ganjil
5. Diagram transisi-nya :



6. DFA nya

   Q = {q0 , q1 , q2 , q3 }

   Σ = {0,1}

   S = q0

   F = { q0}

fungsi transisi

δ	0	1
q_0	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

$\delta(q_0,011) = \delta(q_2,11) = \delta(q_3,1) = q_2$ Ditolak

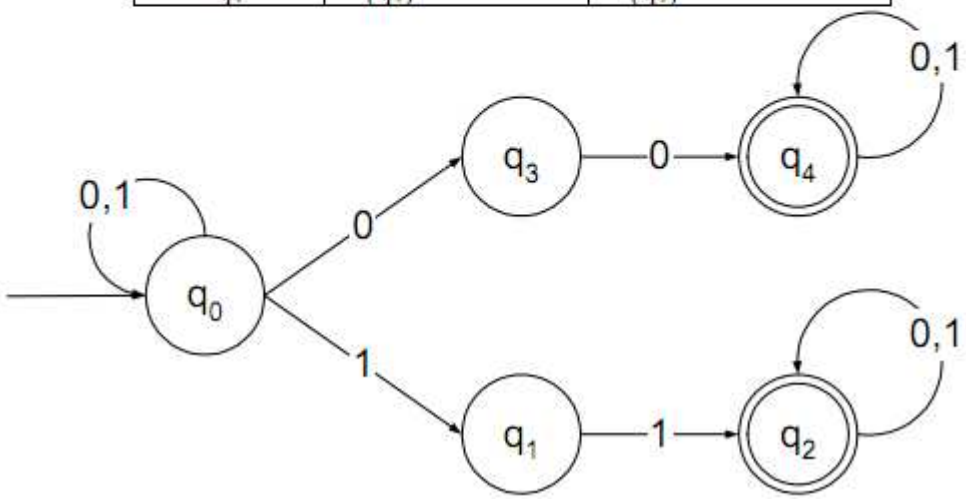
$\delta(q_0,1010) = \delta(q_1,010) = \delta(q_3,10) = \delta(q_2,0) = q_0$ Diterima

2.5.5.2 *Nondeterministic Finite Automata*

1. Perbedaan dengan NFA: fungsi transisi dapat memiliki 0 atau lebih fungsi transisi

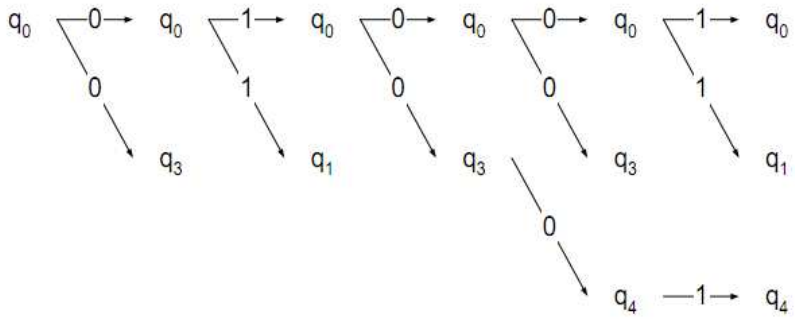
- $G = (\{q_0, q_1, q_2, q_3, q_4\}, \{0,1\}, \delta, q_0, \{q_2, q_4\})$

δ	0	1
q_0	$\{q_0,q_3\}$	$\{q_0,q_1\}$
q_1	ε	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	ε
q_4	$\{q_4\}$	$\{q_4\}$

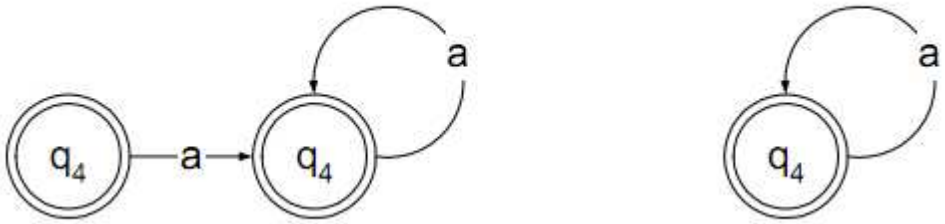


2. *String* diterima NFA bila terdapat suatu urutan transisi berdasar *input*, dari *state* awal ke *state* akhir.
3. harus mencoba semua kemungkinan.

4. Contoh : *string* 01001



Dua buah FSA disebut ekuivalen apabila kedua FSA tersebut menerima bahasa yang sama. Contoh : FSA yang menerima bahasa $\{a^n \mid n \geq 0\}$



Dua buah state dari FSA disebut *indistinguishable* (tidak dapat dibedakan) apabila : $\delta(q,w) \in F$ sedangkan $\delta(p,w) \notin F$ dan $\delta(q,w) \notin F$ sedangkan $\delta(p,w) \in F$ untuk semua $w \in \Sigma^*$.

Dua buah *state* dari FSA disebut *distinguishable* (dapat dibedakan) bila terdapat $w \in \Sigma^*$ sedemikian hingga:

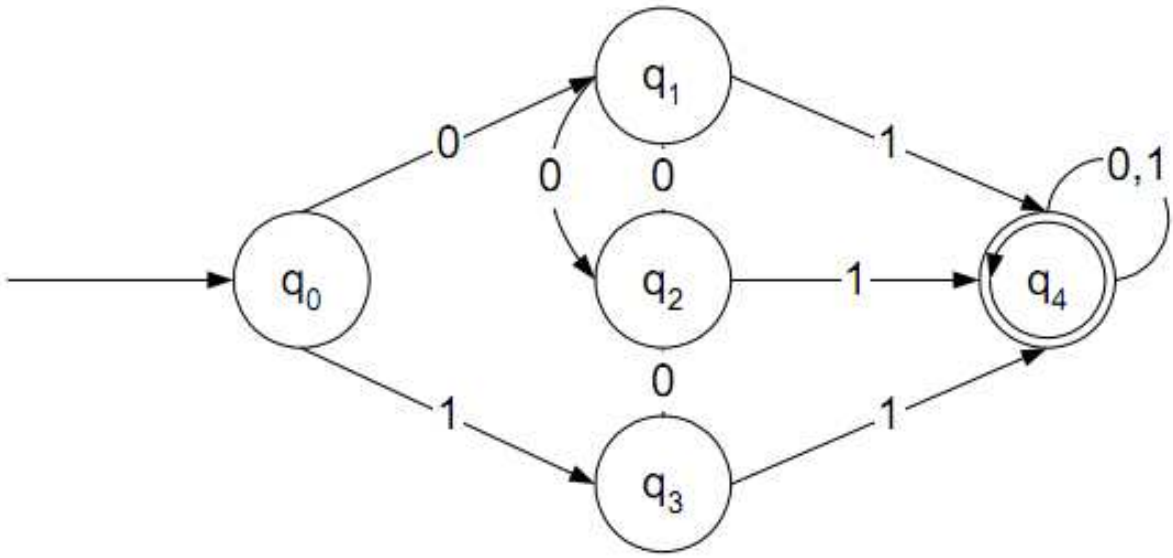
$\delta(q,w) \in F$ sedangkan $\delta(p,w) \notin F$ dan

$\delta(q,w) \notin F$ sedangkan $\delta(p,w) \in F$ untuk semua $w \in \Sigma^*$

Prosedur menentukan pasangan status *indistinguishable*

1. Hapus semua *state* yang tak dapat dicapai dari *state* awal.
2. Catat semua pasangan *state* (p,q) yang *distinguishable*, yaitu {(p,q) | p ∈ F ∧ q ∉ F}
3. Untuk setiap pasangan (p,q) sisanya, untuk setiap a∈ Σ, tentukan δ(p,a) dan δ(q,a)

Contoh :



1. Hapus *state* yang tidak tercapai -> tidak ada
2. Pasangan *distinguishable* (q_0,q_4), (q_1,q_4), (q_2,q_4), (q_3,q_4).
3. Pasangan sisanya (q_0,q_1), (q_0,q_2), (q_0,q_3), (q_1,q_2) (q_1,q_3) (q_2,q_3)

pasangan	state 1		state 2		hasil
	0	1	0	1	
(q_0,q_1)	q1	q3	q2	q4	distinguishable
(q_0,q_2)	q1	q3	q1	q4	distinguishable
(q_1,q_2)	q2	q4	q1	q4	indistinguishable
(q_0,q_3)	q1	q3	q2	q4	distinguishable
(q_1,q_3)	q2	q4	q2	q4	indistinguishable
(q_2,q_3)	q1	q4	q2	q4	indistinguishable

Catatan : jumlah pasangan seluruhnya : $C\binom{5}{2} = \frac{5!}{2!3!} = 10$

Prosedur Reduksi DFA

1. Tentukan pasangan status *indistinguishable*.
2. Gabungkan setiap group *indistinguishable state* ke dalam satu *state* dengan relasi pembentukan group secara berantai : Jika p dan q *indistingishable* dan jika q dan r *indistinguishable* maka p dan r *indistinguishable*, dan p,q serta r *indistinguishable* semua berada dalam satu group.
3. sesuaikan transisi dari dan ke *state-state* gabungan.

2.6 Notasi BNF (*Backus-Naur Form*)

Di dalam ilmu komputer, BNF (*Backus Normal Form atau Backus-Naur Form*) adalah notasi untuk tata bahasa CFG (*Context-free Grammar*), yang sering dipakai untuk mendiskripsikan sintaks tata bahasa yang digunakan, contoh: bahasa pemrograman, format dokumen, set intruksi, dan protokol komunikasi. Notasi BNF ini diterapkan dimanapun saat deskripsi dari tata bahasa diperlukan, misalnya dalam spesifikasi tata bahasa resmi, dalam buku manual, dan di buku teori bahasa pemrograman.

Banyak ekstensi dan varian dari notasi asli yang digunakan, beberapa yang pasti didefinisikan termasuk EBNF (*Extended Backus-Naur Form*) dan ABNF (*Augmented Backus-Naur Form*).

Sebuah spesifikasi BNF adalah serangkaian aturan derivasi, ditulis :

```
<symbol> ::= __expression__
```

Dimana `<symbol>` adalah sebuah non-terminal, dan `__expression__` terdiri dari satu atau lebih barisan simbol. Barisan yang dipisahkan oleh simbol | mempunyai arti sebuah pilihan. Semua simbol pada sisi kanan menjadi subtitusi bagi simbol non-terminal pada sisi kiri. Simbol terminal tidak boleh ada pada sisi kiri. Sisi kiri selalu diisi oleh satu simbol non-terminal yang diapit oleh simbol < dan >.

Berikut ini adalah aturan sintaks pada BNF yang sering dipakai :

- Untuk menandakan item yang opsional (bisa ada atau bisa tidak ada), ditandai dengan diapit dua buah kurung siku, yaitu [dan].

  Contoh: <kalimat> ::= <subjek> <predikat> [<objek>]

- Simbol non-terminal diapit oleh simbol < dan >. Simbol non-terminal selalu ada di sisi kiri. Untuk simbol terminal biasanya diapit oleh tanda kutip ‘. Contoh : `<angkasatu> ::= ‘1’`

- Item yang berulang sebanyak 0 kali atau lebih, diapit oleh kurung kurawal. Atau diapit dengan tanda kurung dengan diakhiri dengan tanda superscript bintang *.

  Contoh 1 : `<word> ::= <letter> {<letter>}`

  Contoh 2 : `<word> ::= <letter> (<letter>)`*

- Item yang berulang sebanyak 1 kali atau lebih, diapit dengan tanda kurung dengan diakhiri dengan tanda superscript tambah $^{+}$.

  Contoh : `<word> ::= (<letter>)`$^{+}$

- Simbol terminal bisa ditulis dengan cetak tebal (bold) dan simbol non-terminal bisa ditulis dengan cetak biasa (*plain text*)

  Contoh : `<angkasatu> ::= ‘1’`

- Untuk menandakan pilihan, biasanya digunakan tanda ‘|’

  Contoh : `<angkabiner> ::= ‘1’ | ‘0’`

- Untuk mengelompokkan simbol-simbol ke dalam group simbol bisa digunakan simbol tanda kurung.