BAB II

LANDASAN TEORI

II.1.Sistem Pakar(*Expert System*)

Sistem Pakar (*Expert Sistem*) adalah sistem yang berusaha mengadopsi pengetahuan manusia ke komputer yang dirancang untuk menyelesaikan masalah layaknya seorang pakar.(Andi, 2009)

II.1.1. Pengertian Sistem Pakar

Suatu sistem disebut sebagai sistem pakar jika mempunyai ciri dan karakteristik tertentu. Hal ini juga harus didukung oleh komponen-komponen sistem pakar yang mampu menggambarkan tentang ciri dan karakteristik tertentu. (Andi, 2009)

Sistem adalah aplikasi berbasis komputer yang digunakan untuk menyelsaikan masalah sebagaimana yang dipikirkan oleh pakar (Kusrini, 2008).Pakar yang dimaksud di sini adalah orang yang mempunyai keahlian khusus yang dapat menyelesaikan masalah yang tidak dapat diselesaikan oleh orang awam. Pengetahuan merupakan sumber utama yang sangat penting, tetapi hanya dimiliki oleh sedikit pakar saja. Oleh karena itu penting sekali untuk memperoleh kepakaran itu agar setiap orang bisa menggunakannya. Sistem pakar merupakan media langsung untuk melakukan pekerjaan seorang pakar.

Keahlian dipindahkan dari pakar ke komputer.Pengetahuan ini kemudian disimpan kedalam komputer untuk mendapatkan informasi, sistem pakar

menanyakan fakta-fakta dan dapat membuat penalaran (*inferensi*) dan sampai pada suatu kesimpulan (Turban, 2005).Kemudian sistem pakar memberikan penjelasan (memberikan kesimpulan atas hasil konsultasi yang telah dilakukan sebelumnya). (Kusrini, 2008).

II.1.2. Kegunaan Sistem Pakar

Sistem pakar (*expert system*) memiliki beberapa kegunaan diantaranya (Andi, 2009):

1. Orang awam yang bukan pakar untuk meningkatkan kemampuan mereka dalam memecahkan masalah.
2. Bisa melakukan proses secara berulang secara otomatis.
3. Menyimpan pengetahuan dan keahlian para pakar.
4. Meningkatkan output dan produktivitas.
5. Meningkatkan kualitas.
6. Mampu mengambil dan melestarikan keahlian para pakar (terutama yang termasuk keahlian langka).
7. Mampu beroperasi dalam lingkungan yang berbahaya.
8. Memiliki kemampuan untuk mengakses pengetahuan.
9. Memiliki reabilitas.
10. Meningkatkan kapabilitas sistem komputer.
11. Memiliki kemampuan untuk bekerja dengan informasi yang tidak lengkap dan mengandung ketidakpastian.
12. Sebagai media pelengkap dalam pelatihan.

13. Meningkatkan kapabilitas dalam penyelesaian masalah.
14. Menghemat waktu dalam pengambilan keputusan

II.1.3. Ciri-Ciri Sistem Pakar

Sistem pakar yang baik harus memenuhi ciri-ciri sebagai berikut (Andi, 2009):

1. Memiliki informasi yang handal.
2. Mudah dimodifikasi.
3. Dapat digunakan dalam berbagai jenis komputer.
4. Memiliki kemampuan untuk belajar beradaptasi.

II.1.4 Struktur Sistem Pakar

Komponen utama pada struktur sistem pakar meliputi(Andi, 2009):

1. Basis Pengetahuan (*Knowledge Base*)

   Basis pengetahuan merupakan inti dari suatu sistem pakar, yaitu berupa representasi pengetahuan dari pakar.Basis pengetahuan tersusun atas fakta dan kaidah.Fakta adalah informasi tentang objek, peristiwa, atau situasi. Kaidah adalah cara untuk membangkitkan suatu fakta baru dari fakta yang sudah diketahui.

2. Mesin Inferensi (*Inference Engine*)

   Mesin inferensi berperan sebagai otak dari sistem pakar. Mesin inferensi berfungsi untuk memandu proses penalaran terhadap suatu kondisi, berdasarkan pada basis pengetahuan yang tersedia. Di dalam mesin inferensi

terjadi proses untuk memanipulasi dan mengarahkan kaidah, model, dan fakta yang disimpan dalam basis pengetahuan dalam rangka mencapai solusi atau kesimpulan.

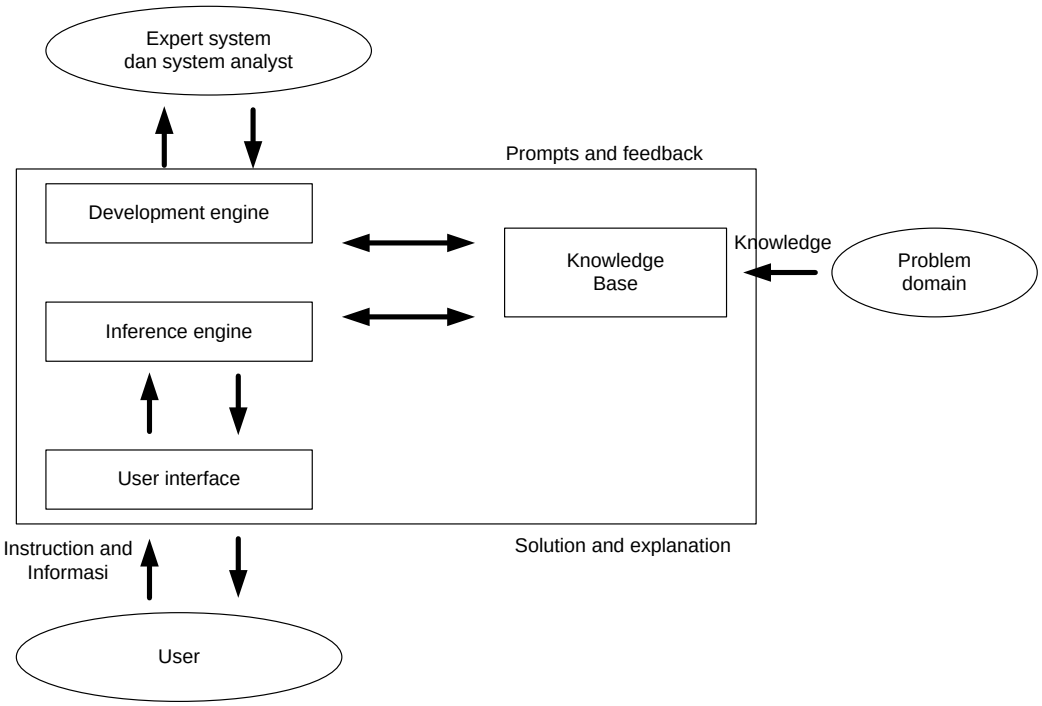
3. Basis Data (*Data Base*)

   Basis data terdiri atas semua fakta yang diperlukan, dimana fakta-fakta tersebut digunakan untuk memenuhi kondisi dari kaidah-kaidah dalam sistem. Basis data menyimpan semua fakta.

4. Pemakai (*User Interface*)

   Fasilitas ini digunakan sebagai perantara komunikasi antarapemakai.dengan komputer.

Berikut adalah gambar dari Komponen utama sistem pakar:



Gambar II.1: Komponen Utama Struktur Sistem Pakar
(Sumber : Yuni Wong ; 2015; 25)

II.1.5 Tahapan Pengembangan Sistem Pakar

Tahapan yang dilakukan dalam mengembangkan sistem pakar, diantaranya (Andi, 2009):

1. Penilaian (*Assessment*)

   Merupakan proses untuk menentukan kelayakan dan justifikasi atas permasalahan yang akan di ambil. Setelah itu masalah diperiksa lebih lanjut untuk menentukan tujuan keseluruhan dari proyek. Upaya ini dilakukan untuk menentukan fitur-fitur penting dan ruang lingkup dari proyek.

2. Akuisisi pengetahuan

   Merupakan proses untuk mendapatkan pengetahuan tentang permasalahan yang dibahas dan akan digunakan sebagai panduan dalam upaya pengembangan.

3. Desain

   Pengetahuan yang diperoleh selama tahap akuisisi pengetahuan digunakan sebagai pendekatan dalam merepresentasikan pengetahuan pakar dan strategi pemecahan masalah ke dalam sistem pakar.

4. Pengujian

   Merupakan tahap dimana dilakukan pengujian terhadap sistem pakar yang telah dibangun.

5. Dokumentasi

   Tahap dokumentasi diperlukan untuk mengkompilasi seluruh informasi proyek ke dalam bentuk dokumen yang dapat memenuhi persayaratan pengguna dan pengembang dari sistem pakar. Dokumentasi dibutuhkan untuk

mengakomodasi kebutuhan pengguna yang memenuhi persyaratan yang ditemukan pada sebagian besar proyek perangkat lunak. Dokumentasi tersebut menjelaskan tentang bagaimana mengoperasikan sistem dan menyediakan tutorial dalam mengoperasikan fitur utama dari sistem.

6. Pemeliharaan

   Setelah sistem digunakan dalam lingkungan kerja, maka selanjutnya diperlukan pemeliharaan secara berkala. Pengetahuan itu sifatnya tidak statis melainkan terus tumbuh dan berkembang. Pengetahuan dari sistem perlu diperbaharui atau disempurnakan untuk memenuhi kebutuhan saat ini.

II.1.6 Keunggulan dan Kelemahan Sistem Pakar

Adapun dari keunggulan sistem pakar:

1. Kemampuan menghimpun data dalam jumlah yang sangat besar.
2. Kemampuan menyimpan data tersebut untuk jangka waktu yang panjang dalam suatu bentuk yang tertentu.
3. Kemampuan mengerjakan perhitungan secara tepat dan tepat dan tanpa jemu mencari kembali data yang tersimpan dengan kecepatan tinggi.

Di samping memiliki beberapa keuntungan, sistem pakar juga memiliki beberapa kelemahan, antara lain:

1. Biaya yang diperlukan untuk membuat dan memeliharanya sangat mahal.
2. Sulit dikembangkan. Hal ini tentu saja erat kaitannya dengan ketersediaan pakar di bidangnya.
3. Sistem Pakar tidak 100% bernilai benar.

II.2 Representasi Pengetahuan

Agar pengetahuan dapat digunakan dalam sistem, pengetahuan harus dirpresentasikan dalam format tertentu yang kemudian dihimpun dalam suatu basis pengetahuan. Cara pakar merepresentasikan pengetahuan akan mempengaruhi perkembangan, efisiensi, dan perbaikan sistem.

II.2.1. Pengertian Pengetahuan

Definisi umum dari pengetahuan adalah fakta atau kondisi sesuatu atau keadaan yang timbul karena suatu pengalaman. Cabang ilmu filsafat, yaitu *Epistemology*, berkenaan dengan sifat, struktur dan keaslian dari *knowledge*.

Berikut adalah struktur dari *Epistemology* yang merupakan cabang dari ilmu filsafat (Hartati dan Iswanti, 2008):

1. Priori *Knowledge*

   Berarti yang mendahului (pengetahuan datang sebelumnya dan bebas dariarti) Kebenaran yang universal dan tidak dapat disangkal tanpa kontradiksi.

   Contoh: pernyataan logika, hukum matematika.

2. Posteriori *Knowledge*

   Knowledge yang diturunkan dari akal pikiran yang sehat. Kebenaran atau kesalahan dapat dibuktikan dengan menggunakan pengalamanakal sehat.

   Contoh: bola mata seseorang berwarna biru, tetapi ketika orang tersebut mengganti lensa kontaknya, bisa jadi bola matanya menjadi berwarna hijau.

II.2.2. Pengertian Representasi Pengetahuan

Representasi pengetahuan adalah suatu teknik untuk merepresentasikan basis pengetahuan yang diperoleh ke dalam suatu skema/diagram tertentu sehingga dapat diketahui relasi/keterhubungan antara suatu data dengan data yang lain. Teknik ini membantu *knowledge engineer* dalam memahami struktur pengetahuan yang akan dibuat sistem pakarnya.Manfaat representasi pengetahuan (Hartati dan Iswanti, 2008):

1. dengan representasi yang baik, membuat objek dan relasi yang penting menjadi jelas.
2. dengan represntasi buatandapat menyingkap constrain (batasan) dalam suatu permasalahan.
3. dengan representasi buatan kita akan dapatkan objek dan relasi secara bersama-sama.

II.2.3. Aturan IF-THEN

Forward Chaining adalahpencocokan fakta atau pernyataan dimulai dari bagian sebelah kiri (IF) dulu. Dengan perkataan lain, penalaran dimulai dari fakta terlebih dahulu untuk menguji kebenaran hipotesa.

Contoh-contoh aturan

No.	Aturan
R-1	IF A & B THEN C
R-2	IF C THEN D
R-3	IF A & E THEN F

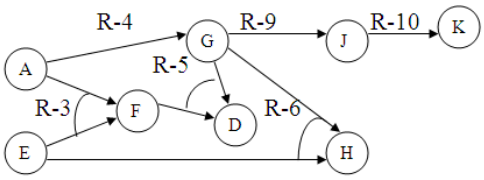
R-4	IF A THEN G
R-5	IF F & G THEN D
R-6	IF G & E THEN H
R-7	IF C & H THEN I
R-8	IF I & A THEN J
R-9	IF G THEN J
R-10	IF J THEN K

Pada tabel di atas ada 10 aturan (*rule*) yang tersimpan dalam basis pengetahuan. Fakta awal yang diberikan hanya: A & E (yaitu berarti A dan F bernilai benar). Hipotesanya adalah apakah K bernilai benar ? Untuk itu dilakukan langkah-langkah inferensia sebagai berikut:

1. Start dari R-1. A merupakan fakta sehingga bernilai benar, sedangkan B belum diketahui kebenarannya, sehingga C pun belum diketahui kebenarannya. Oleh karena itu pada R-1 kita tidak mendapatkan informasi apapun. Sehingga kita menuju ke R-2.
2. Pada R-2 juga sama kita tidak dapat memastikan kebenaran D karena C belum diketahui apakah benar atau salah sehingga kita tidak mendapatkan informasi apapun , sehingga kita menuju ke R-3.
3. Pada R-3 A dan E adalah fakta sehingga jelas benar. Dengan demikian F sebagai konsekuensi juga benar. Dari sini kita mendapat fakta baru yaitu F, tetapi karena F bukan hipotesa maka langkah diteruskan ke R-4.

4. Pada R-4 A adalah fakta berarti jelas benar, sehingga G sebagai konsekuen juga benar. Jadi terdapat fakta baru yaitu G, tetapi G bukan hipotesa sehingga langkah diteruskan ke R-5.
5. Pada R-5 F dan G benar berdasarkan aturan R-3 dan R-4, sehingga D sebagai konsekuen juga benar. Terdapat fakta baru yaitu D, tetapi D bukan hipotesa sehingga diteruskan ke R-6.
6. Pada R-6, E dan G benar berdasarkan fakta dan R-4, maka H benar. Sehingga terdapat fakta baru yaitu H, tetapi H bukan hipotesa, sehingga diteruskan ke R-7.
7. Pada R-7, karena C belum diketahui, maka I juga belum dapat diketahui kebenarannya, sehingga kita tidak mendapatkan informasi apapun. Diteruskan ke R-8.
8. Pada R-8, meskipun A benar karena fakta tetapi I belum diketahui, sehingga J juga belum dapat diketahui kebenarannya. Diteruskan ke R-9.
9. Pada R-9, G benar menurut R-4, sehingga konsekuennya J juga benar, tetapi J bukan hipotesa, maka diteruskan ke R-10.
10. Pada R-10, K benar karena J benar menurut R-9. Karena K merupakan hipotesa yang dibuktikan maka selesai.

Secara diagram dapat digambarkan sebagai berikut:

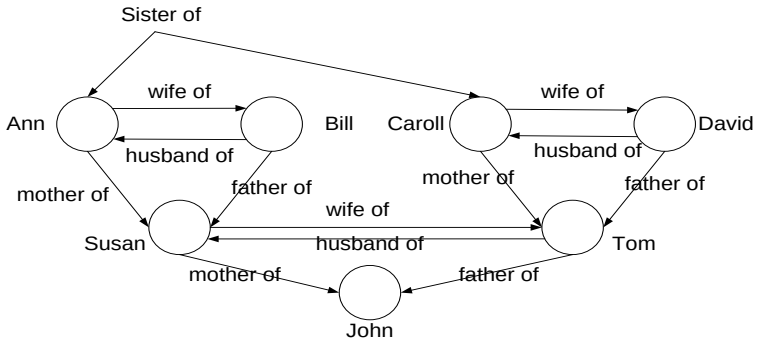


Gambar II.2. Certainty Factor
(Sumber : Hartati dan Iswanti, 2008:23)

II.2.4. Jaringan Semantik

Dibangun oleh M.R.Quillian, sebagai model memori manusia. Representasi grafisnya didapat dari informasi Propositional. Proposisi adalah pernyataan yang dapat bernilai benar atau salah. Jaringan semantik disajikan dalam bentuk graf berarah. Sedangkan Node merepresentasikan konsep, objek atau situasi: Hartati dan Iswanti, 2008:17)

Contoh Jaringan Semantik:

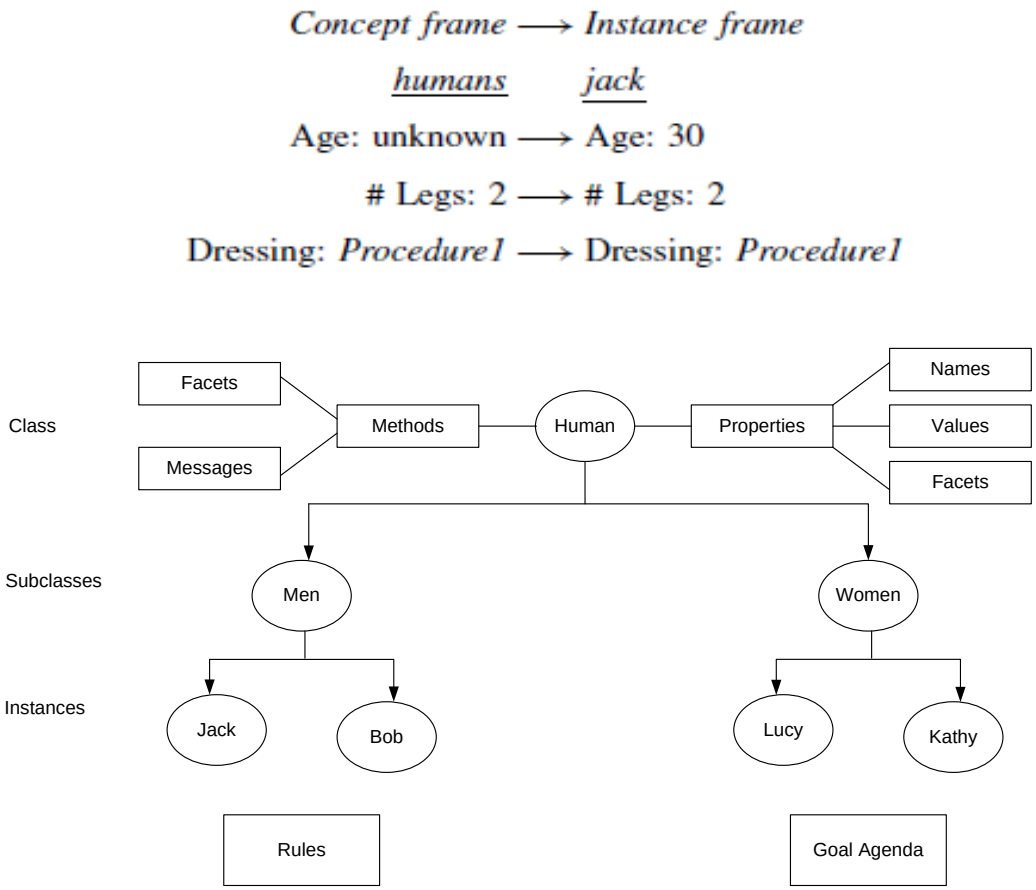


Gambar II.3: Contoh Jaringan Semantik
(Sumber : Yuni Wong,:2015:23)

II.2.5. Frame-Based

Pengetahuan direpresentasikan dalam suatu bentuk hirarki atau jaringan *frame*. Di dalam *Frame-based* masalah dipandang sebagai sekumpulan objek yang secara alami menjelaskan permasalahannya.

Suatu *frame* memilik nama, slot dengan label yang menjelaskan ciri/sifat *(property)* utama konsep tersebut, dan nilai yang mungkin untuk setiap ciri/sifat. Suatu *frame* juga dapat disertai dengan proseduryang terkait dengan konsep tersebut. Apabila suatu *instance* dari suatu konsep, nilaidari cirinya dimasukkan ke dalam frame tersebut.(Hartati dan Iswanti, 2008)



Gambar II.4. Contoh Frame-Based

(Sumber : Yuni Wong : 2015:24)

II.2.6. Script

Skrip *(script)* merupakan representasi terstruktur yang menggambarkan urutan stereotip dari kejadian-kejadian dalam sebuah konteks khusus. Skrip mula-mula dirancang oleh Schank dan kelompok risetnya sebagai alat pengorganisasi struktur-struktur *ketergantungan konseptual menjadi* deskripsi khusus.

Komponen-komponen skrip adalah:

1. Kondisi entriatau deskriptor dunia sekitar kita yang harus benar agar skrip dapat dipanggil. Contoh: dalam hal skrip restoran, ini mencakup restoran yang sedang buka dan pelanggan yang sedang lapar.
2. Hasil atau fakta yang benar begitu skrip diakhiri. Misalnya, pelanggan sudah kenyang, dan pemilik restoran memiliki uang yang lebih banyak (karena pembayaran oleh pelanggan tersebut).
3. Penyanggaatau apa-apa yang merupakan isi skrip. Di sini meliputi meja,kursi, pelayan, dan menu.
4. Peranadalah tindakan yang dilakukan oleh partisipan individual. Misalnya, pelayan yang mengantar pesanan, dan memberikan tagihan pada pelanggan, serta pesanan pelanggan, makan, dan membayar.
5. Adegan yang merupakan kejadian yang menunjukkan aspek waktu dari skrip. Di sini dapat berupa: masuk ke restoran, memesan, makan, dan lain-lain. (Sumber :Hartati dan Iswanti, 2008:20).

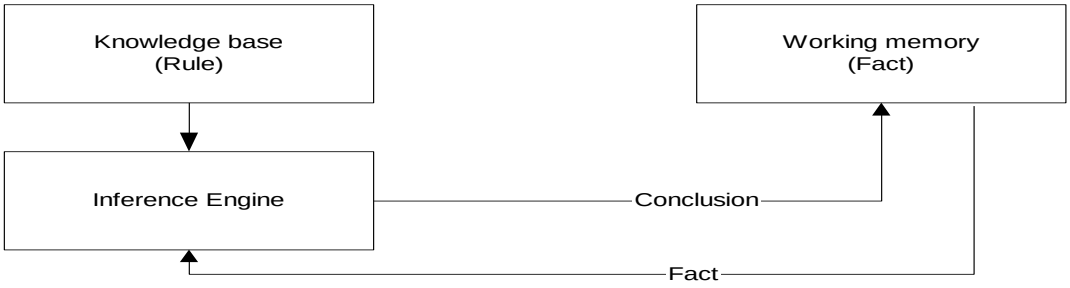
II.2.7. Inferensi dengan Rules

Inferensi dengan *rules* merupakan implementasi dari modus ponen, yang direfleksikan dalam mekanisme pencarian *(search)*. Dapat pula mengecek semua *rule* pada *knowledge base* dalam arah *forward* maupun *backward*. Proses pencarian berlanjut sampai tidak ada *rule* yang dapat digunakan atau sampai sebuah tujuan (*goal*) tercapai. Ada dua metode *inferencing* dengan *rules*, yaitu *Certaity Factor* atau data-*driven* dan *backward chaining* atau *goal-driven*.

II.2.8. Certaity Factor

Dalam *Certaity Factor* tidak ada *goal* yang akan dibuktikan. Sistem ini menggali sebanyak mungkin informasi dari fakta-fakta yang terkait permasalahan. Fakta-fakta tersebut memunculkan fakta-fakta baru yang dihasilkan dengan menggunakan kaidah.

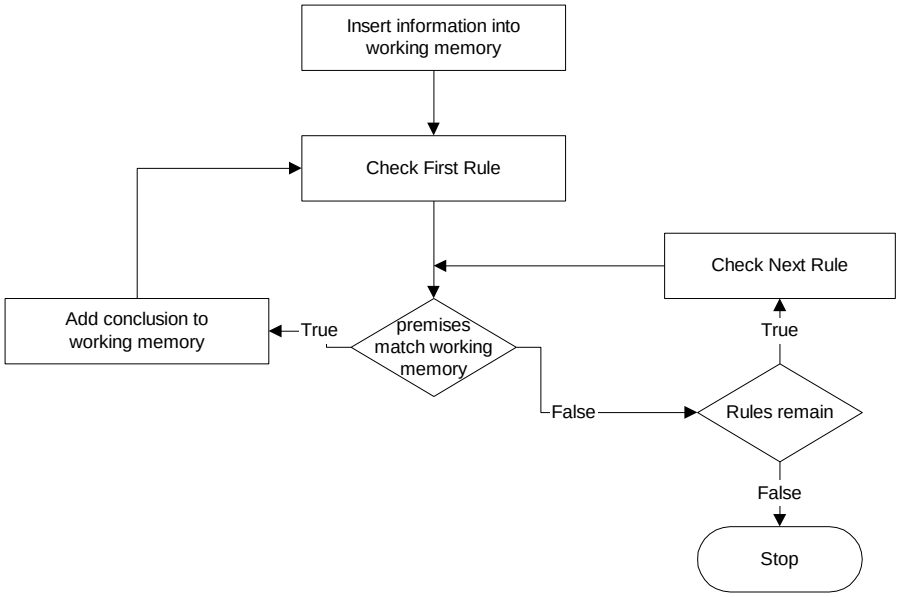
Certaity Factor cocok untuk masalah dimana diinginkan pembelajaran sebanyak mungkin dari informasi yang ada mengenai permasalahan. Gambar berikut ini merupakan bentuk dari *Certaity Factor* dari sebuah Sistem pakar *Certaity Factor* berbasis aturan yang dapat dimodelkan seperti Gambar II.5 berikut,



Gambar II.5: Model Berbasis Aturan
(Sumber : Yuni Wong ; 2015 ; 26)

Operasi dari sistem *forward chaining* dimulai dengan memasukkan sekumpulan fakta yang diketahui ke dalam memori kerja (*working memory*), kemudian menurunkan fakta baru berdasarkan aturanyang premisnya cocok dengan fakta yang diketahui. Proses ini dilanjutkan sampai dengan mencapai *goal*

atau tidak ada lagi aturan yang premisnya cocok dengan fakta yang diketahui. Operasi tersebut dapat digambarkan seperti gambar II.6.



Gambar II.6: Operasi Sistem Forward Chaining
(Sumber : Tilotma Sharma : 2015 :26)

Contoh dari *forward chaining*

Misalkan kita memiliki kaidah dari modus ponen

$$p \rightarrow q$$

$$\underline{p \quad\quad}$$

$$\therefore q$$

dalam bentuk :

gajah (x)_mamalia (x)

mamalia (x)_binatang (x)

Kaidah ini dapat digunakan dalam rantai sebab-akibat dari inferensi *forward* yang menarik kesimpulan bahwa Bona adalah binatang, jika diketahui Bona adalah gajah.

gajah (Bona)

‖

gajah (x) ===➔ mamalia (x)

‖

mamalia (x) ===➔ binatang (x)

‖

binatang (Bona)

Certainty Factor semula dikembangkan pada MYCIN. Faktor kepastian adalah suatu cara penggabungan kepercayaan (*belief*) dan ketidakpercayaan (*disbelief*) menjadi suatu nilai. Pendekatan ini dapat dijabarkan dalam rumus berikut:

$$RI_k(cf) = \min\{P_i(cf)\} \text{ untuk semua } P_i\,(cf) \geq \delta\, i$$

$$cf_k = RI_k(cf) * [\, RI_k(cf)\,]$$

dimana : $RI_k(cf)$: Faktor keyakinan premis dari aturan ke- k

$P_i(cf)$: Faktor keyakinan premis ke- i

δ : Nilai ambang dari faktor keyakinan premis

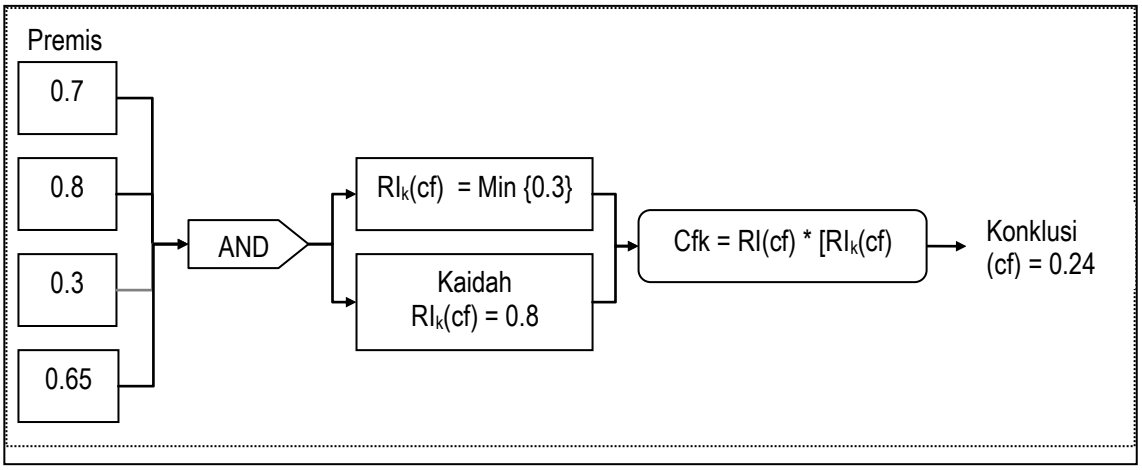
cf_k : Faktor keyakinan hasil dari suatu aturan

$R_k(cf)$: Faktor keyakinan dari aturan ke- k

Certainty factor atau CF merupakan nilai yang menyatakan tingkat keyakinan dari premis atau kaidah tertentu. Nilai CF memiliki selang antara 1 dan 0. Pada Gambar 2.5 diberikan ilustrasi penggunaannya. Pendekatan nilai tingkat keyakinan dari probabilitas dapat diberikan dengan nilai diantara ≥ 0 dan ≤ 1, sehingga sebagai contoh dapat diasumsikan bahwa faktor kepercayaan dari *rule*

itu sendiri adalah 0.8 maka faktor kepercayaan gabungan dari premis-premis yang ada adalah : $RI_k(cf) = \min\{P_i(cf)\} = \min\{0.7,0.8,0.3,0.65\} = 0.3$

Selanjutnya untuk menentukan faktor kepercayaan output (faktor kepercayaan konklusi) dapat dihitung dengan : $cf_k = RI_k(cf) * [\ RI_k(cf)\ ] = 0.3 * 0.8 = 0.24$



Gambar II.20 Faktor Kepercayaan Dengan Beberapa Premis Gabungan

Sumber : M. Arhami, 2010

Berikut diberikan sebuah contoh sederhana untuk melakukan proses analisis terhadap sistem pakar yang dibuat.

1. Gejala pertama yaitu anemia, dimiliki oleh keempat penyakit tersebut.
2. Berarti setelah analisa gejala pertama, maka pasien kemungkinan menderita keempat jenis penyakit tersebut.
3. Gejala kedua adalah sesak nafas, dimiliki oleh penyakit LLA dan LMA.
4. Berarti setelah analisa gejala kedua, maka pasien kemungkinan menderita penyakit LLA dan LMA.
5. Gejala ketiga yaitu nyeri dada, hanya dimiliki oleh penyakit LMA.

6. Berarti setelah melakukan proses diagnosa dengan menggunakan ketiga gejala tersebut, maka kemungkinan besar pasien menderita penyakit Leukemia Mielositik Akut (LMA), karena ketiga gejala dimiliki oleh penyakit LMA.

Analisa berdasarkan metode *Certainty Factor* dapat dirincikan sebagai berikut:

Keterangan:

H_1 = anemia

H_2 = sesak nafas

H_3 = nyeri dada

Penyakit : Leukemia Limfositik Akut (L-001)

G1 = 0.46 = P(E|H1) = CF(H1)

G2 = 0.65 = P(E|H3) = CF(H3)

Semesta = 0.46 + 0.65

Semesta = 1.11

P(H1) = 0.46 / 1.11 = 0.414414

P(H2) = 0.65 / 1.11 = 0.585586

Probabilitas = (0.414414 * 0.46) + (0.585586 * 0.65)

Probabilitas = 0.571261

P(H1|E) = (0.46 * 0.414414) / 0.571261 = 0.333701

P(H2|E) = (0.65 * 0.585586) / 0.571261 = 0.666299

CF Penyakit = (0.46 * 0.333701) + (0.65 * 0.666299)

CF Penyakit = 0.586597

Penyakit : Leukemia Mielositik Akut (L-002)

G1 = 0.36 = P(E|H1)

G2 = 0.64 = P(E|H2)

G3 = 0.81 = P(E|H3)

Semesta = 0.36 + 0.64 + 0.81

Semesta = 1.81

P(H1) = 0.36 / 1.81 = 0.198895027624309

P(H2) = 0.64 / 1.81 = 0.353591160220994

P(H3) = 0.81 / 1.81 = 0.447513812154696

Probabilitas = (0.198895027624309 * 0.36) + (0.353591160220994 * 0.64) +

(0.447513812154696 * 0.81)

Probabilitas = 1.32077348066298

P(H1|E) = (0.36 * 0.198895027624309) / 1.32077348066298 =

0.0542123316322262

P(H2|E) = (0.64 * 0.353591160220994) / 1.32077348066298 =

0.171337739479629

P(H3|E) = (0.81 * 0.447513812154696) / 1.32077348066298 =

0.274449928888145

CF Penyakit = (0.36 * 0.0542123316322262) + (0.64 * 0.171337739479629) +

(0.81 * 0.274449928888145)

CF Penyakit = 0.702954070107923

Penyakit : Leukemia Mielositik Kronik (L-003)

G1 = 0.32 = P(E|H1)

Semesta = 0.32

P(H1) = 0.32 / 0.32 = 1

Probabilitas = (1 * 0.32)

Probabilitas = 0.32

P(H1|E) = (0.32 * 1) / 0.32 = 1

CF Penyakit = (0.32 * 1)

CF Penyakit = 0.32

Penyakit : Leukemia Limfositik Kronik (L-004)

G1 = 0.59 = P(E|H1)

Semesta = 0.59

P(H1) = 0.59 / 0.59 = 1

Probabilitas = (1 * 0.59)

Probabilitas = 0.59

P(H1|E) = (0.59 * 1) / 1.18 = 0.5

CF Penyakit = (0.59 * 0.5)

CF Penyakit = 0.295

Berdasarkan nilai CF yang diperoleh keempat penyakit tersebut, maka nilai CF tertinggi adalah CF = 0.702954070107923, yang dimiliki oleh penyakit Leukemia Mielositik Akut (L-002).

II.3 Unified Modeling Language

Unified Modeling Language (UML) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem perangkat lunak. UML tidak berdasarkan pada bahasa pemrograman tertentu. Standar

spesifikasi UML dijadikan standar defacto oleh OMG (*Object Management Group*) pada tahun 1997. UML yang berorientasikan object mempunyai beberapa notasi standar (Bambang Hariyanto, 2011).

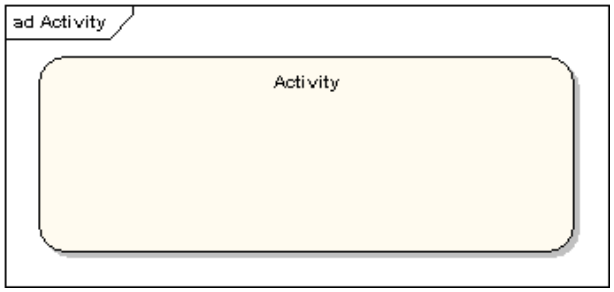
II.3.1. Activity Diagram

Diagram aktivitas atau *activity diagram* menggambarkan *workflow* (aliran kerja)atau aktivitas dari sebuah sistema atau proses bisnis.Yang perlu diperhatikan disini adalah bahwa diagram aktivitas menggambarkan aktivitas sistema bukan apa yang dilakukan actor,jadi aktivitas yang dapat dilakukan oleh sistem (Hendy Setiady, 2013).

Elemen-elemen yang digunakan untuk membentuk suatu *activity diagram* adalah sebagai berikut:

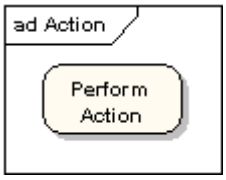
1. *Activity*

   Activity adalah spesifikasi dari urutan parameter dari kelakuan. Sebuah *activity* ditunjukkan dengan menggunakan sebuah persegi panjang dengan sudut bulat dan menyertakan semua aksi, kontrol aliran dan elemen lainnya yang membentuk *activity*.



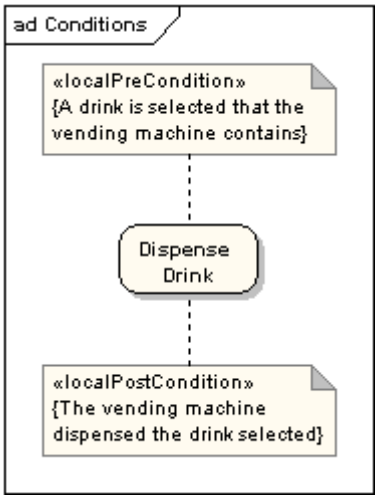
2. *Action*

   Action merepresentasikan sebuah langkah tunggal diantara sebuah *activity*. Action didenotasikan dengan persegi panjang dengan sudut bulat.



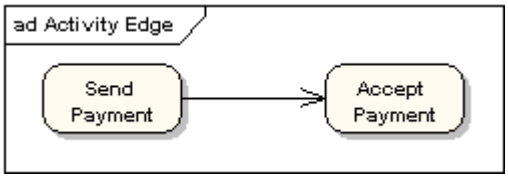
3. *Action Constraint*

   Constraint dapat ditambahkan ke sebuah *action*. Diagram berikut menunjukkan sebuah *action* dengan kondisi lokal pre- dan post-.



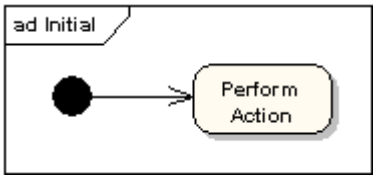
4. *Control Flow*

   Control Flow menunjukkan aliran dari kontrol dari satu *action* ke berikutnya. Notasinya adalah sebuah garis dengan arah panah.



5. *Initial Node*

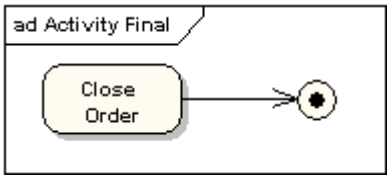
   Sebuah *initial* atau *start node* disimbolkan dengan sebuah titik hitam besar, seperti terlihat pada gambar berikut:



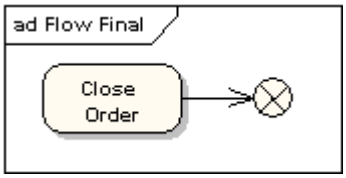
6. *Final Node*

   Terdapat dua tipe dari *final node* yaitu:

   - *Activity final node* yang disimbolkan denagn sebuah lingkaran dengan sebuah titik didalamnya.



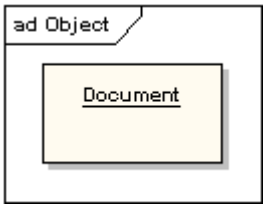
   - *Flow final node* yang disimbolkan dengan sebuah lingkaran dengan sebuah tanda silang didalamnya.



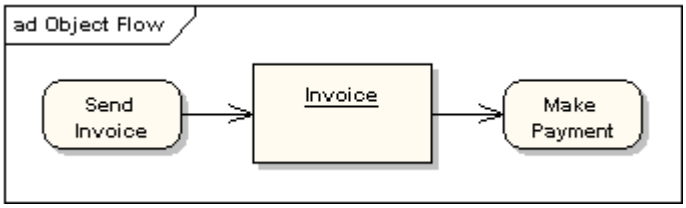
   Perbedaan dari dua tipe *node* adalah *flow final node* melambangkan akhir dari sebuah aliran kontrol tunggal, sedangkan *activity final node* melambangkan akhir dari semua aliran kontrol dalam *activity*.

7. *Object* dan *Object Flow*

   Sebuah *object flow* adalah sebuah *path* dimana *object* atau data dapat dilewatkan. Sebuah object dilambangkan dengan sebuah persegi panjang.

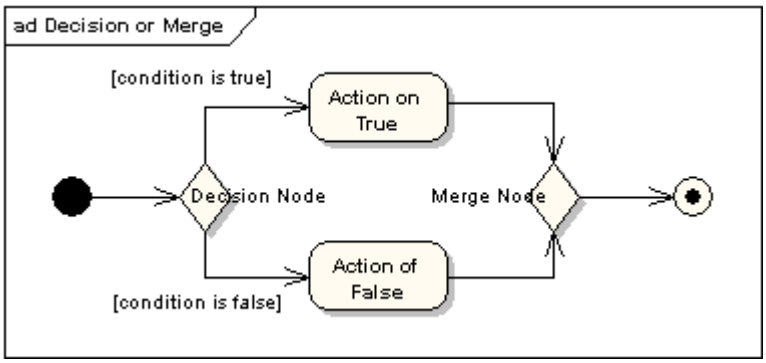


Sebuah *object flow* dilambangkan dengan sebuah *connector* dengan arah panah yang menandakan arah objek dilewatkan.



8. *Decision* dan *Merge Node*

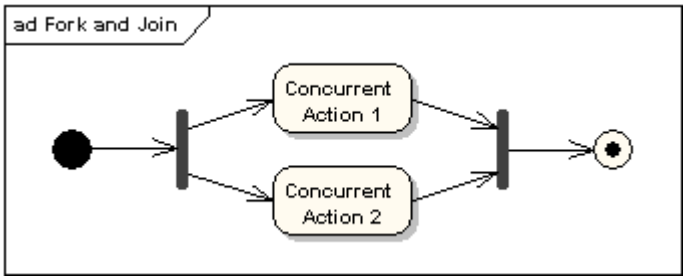
   Decision node dan *merge node* memiliki notasi yang sama yaitu bentuk diamond yang dapat diberi nama. Aliran kontrol yang datang dari sebuah *decision node* akan memiliki kondisi kunci yang mengizinkan kontrol untuk mengalir apabila kondisi kunci terpenuhi. Diagram berikut menunjukkan cara penggunaan dari *decision node* dan *merge node*.



9. *Fork* dan *Join Node*

   Fork dan *join* memiliki notasi yang sama yaitu batang horizontal atau vertikal (orientasi tergantung pada apakah kontrol aliran berjalan dari kiri ke kanan atau dari atas ke bawah). Mereka mengindikasikan awal dan

akhir dari urutan kontrol yang terjadi bersamaan. Diagram berikut menunjukkan contoh penggunaannya:



II.3.2. Use Case Diagram

Diagram *use case* adalah diagram yang menunjukkan fungsionalitas suatu sistem atau kelas dan bagaimana sistem tersebut berinteraksi dengan dunia luar dan menjelaskan sistem secara fungsional yang terlihat user. Biasanya dibuat pada awal pengembangan. *Use case* diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana".

Secara umum diasumsikan bahwa *use case* yang di-include akan dipanggil setiap kali *use case* yang meng-include dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang common. Sebuah *use case* juga dapat meng-extend *use case* lain dengan *behavior*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

Diagram ini menunjukkan fungsionalitas suatu sistem atau kelas dan bagaimana sistem tersebut berinteraksi dengan dunia luar.

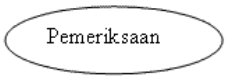
Notasi Gambar Yang Dipakai *Use case*:

1. *Actor*

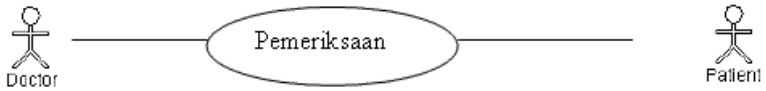
   Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

2. *Case*

   Menggambarkan deskripsi yang melibatkan *actor*.



   Contoh *Case* - *Actor*:



3. *Extend*

   Relasi yang digunakan jika *use case* yang satu mirip dengan *use case* yang lain.

4. *Include*

   Relasi jika terdapat perilaku yang mirip dengan beberapa *use case*.

Cara Menemukan *Use case*(Bambang Hariyanto, 2011):

1. Pola perilaku perangkat lunak aplikasi.
2. Gambaran tugas dari sebuah *actor*.

3. Sistem atau “benda” yang memberikan sesuatu yang bernilai kepada *actor*.
4. Apa yang dikerjakan oleh suatu perangkat lunak (bukan bagaimana cara mengerjakannya).

II.4.Basis Data

Basis data tidak hanya merupakan kumpulan *file*. Lebih dari itu, basis data adalah pusat sumber data yang caranya dipakai oleh banyak pemakai untuk berbagai aplikasi. Inti dari basis data adalah *database management system* (DBMS), yang membolehkan pembuatan, modifikasi, dan pembaharuan basis data; mendapatkan kembali data dan membangkitkan laporan.

Tujuan basis data yang efektif yaitu (Kendall, 2010):

1. Memastikan bahwa data dapat dipakai di antara pemakai untuk berbagai aplikasi.
2. Memelihara data baik keakuratan maupun kekonsistenannya.
3. Memastikan bahwa semua data yang diperlukan untuk aplikasi sekarang dan yang akan datang akan disediakan dengan cepat.
4. Membolehkan basis data untuk berkembang dan kebutuhan pemakai untuk berkembang.
5. Membolehkan pemakai untuk membangun pandangan personalnya tentang data tanpa memperhatikan cara data disimpan secara fisik.

Beberapa konsep *database* untuk analisis sistem yaitu (Jeffery L. Whitten, et.al, 2009):

1. *Field*

   Merupakan implementasi fisik pada atribut data. *Field* adalah unit terkecil dari data *meaningful* yang telah disimpan pada sebuah *file* atau *database*. *Field*mempunyai empat tipe yaitu:

   a. *Primary key*, yaitu sebuah *field* yang nilainya mengidentifikasikan satu dan hanya satu *record* pada sebuah *file*.
   b. *Secondary key*, yaitu sebuah pengidentifikasi alternatif pada sebuah *database*. Nilai *secondary key* mungkin mengidentifikasi sebuah *record*tunggal atau sebuah subset dari semua *record*.
   c. *Foreign key*, yaitu pointer ke *record-record* dari sebuah *file* lain pada sebuah *database*.
   d. *Descriptive key*, yaitu semua *field* lainnya (*nonkey*) yang menyimpan data bisnis.

2. *Record*

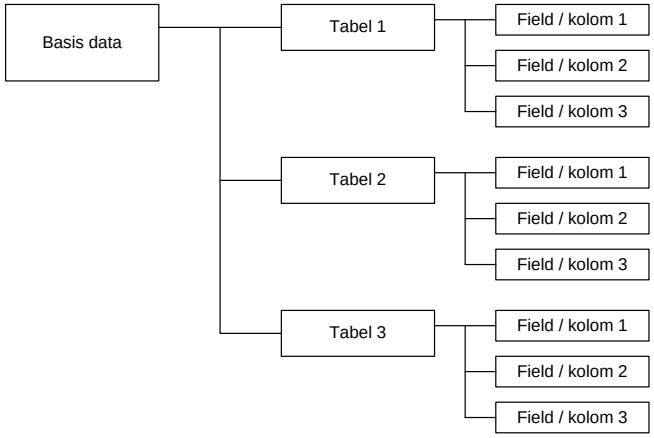
   Merupakan sebuah kumpulan *field* yang disusun pada format yang telah ditentukan.

3. *File* dan tabel

   File merupakan kumpulan dari semua kejadian dari sebuah struktur *record* yang ditentukan. Tabel merupakan ekuivalen *database* relasional dari sebuah *file*.

Sebuah basis data terdiri atas beberapa tabel (sesuai dengan kebutuhan program). Tabel adalah kumpulan dari *record-record* sejenis dengan panjang elemen yang sama tapi *data value*nya berbeda.Sebuah tabel terdiri atas beberapa

record. *Record* adalah kumpulan dari atribut-atribut yang menginformasikan sebuah entitas secara lengkap. Sebuah *record* terdiri atas beberapa *field*. *Field*adalah item-item yang terdapat pada sebuah entitas yang dapat bertindak sebagai pengenal bagi entitas tersebut.Gambar struktur dari basis datadapat dilihat pada Gambar II.7 berikut:



Gambar II.7. Struktur Basis Data

(Sumber: Harianto Kristanto, 2011)

Pembangunan sistem informasi bertumpu pada kualitas *database* yang disusun dan dibentuk. *Database* yang dibentuk diharapkan memiliki sifat-sifat antara lain (Oetomo, Budi Sutejo Dharma, 2012):

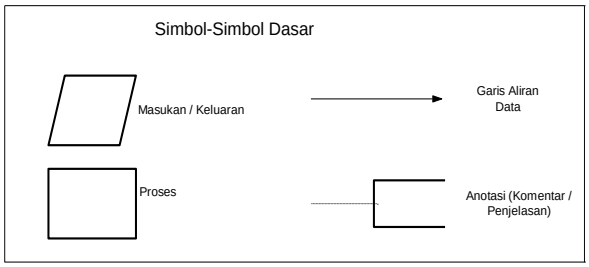
1. Efisien dan efektif dalam pengorganisasiannya, artinya untuk menambah, menyisipkan atau menghapus data dapat dilakukan dengan mudah dan sederhana.
2. Bebas redudansi, meskipun pada batas-batas tertentu yang dapat ditolerir, redudansi juga diperbolehkan misalnya untuk mengurangi kompleksitas dalam penulisan program.

3. Fleksibel, artinya *database* dapat diakses dengan mudah, dinamis dan tidak tergantung sepenuhnya pada aplikasi-aplikasi tertentu.
4. Sistem *database* yang dapat dikases secara bersama dalam lingkungan jaringan sehingga mendukung penggunaan bersama dan distribusi data.

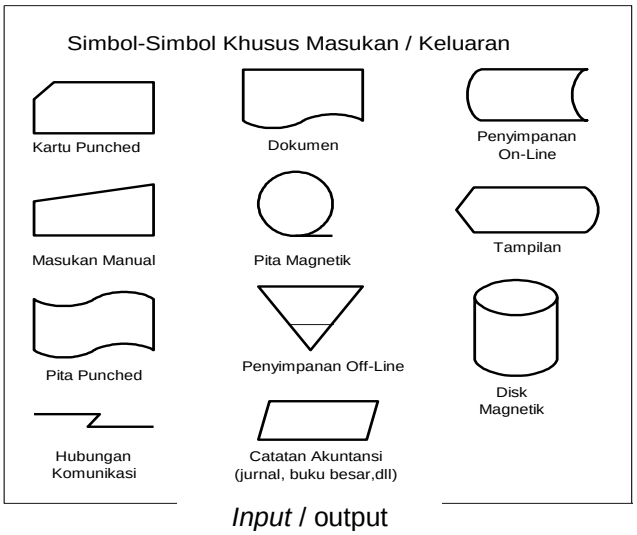
II.5. Flowchart

Flowchart adalah penggambaran secara grafik dari langkah-langkah danurut-urutan prosedur dari suatu program. *Flowchart* menolong analis dan *programmer* untuk memecahkan masalah kedalam segmen-segmen yanglebih kecil dan menolong dalam menganalisis alternatif-alternatif lain dalam pengoperasian.

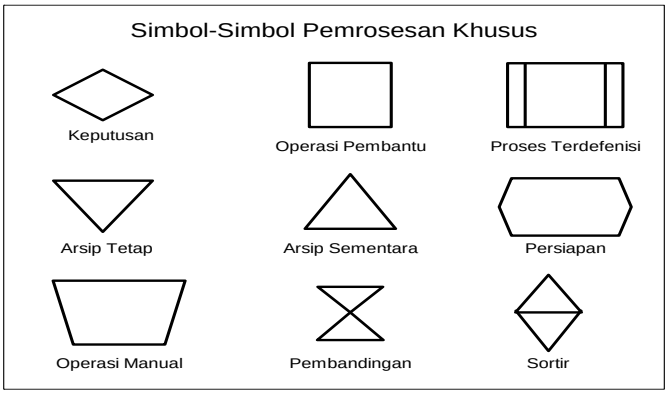
Simbol-simbol *flowchart* yang biasanya dipakai adalah simbol *flowchart* standar yang dikeluarkan oleh ANSI dan ISO. Simbol-simbol ini dapat dilihat pada gambar berikut ini :



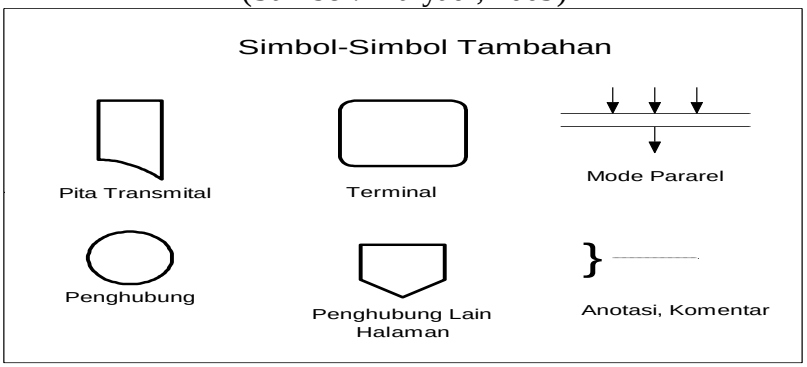
Gambar II.8 Simbol-Simbol Dasar Flowchart
(Sumber: Mulyadi, 2009)



Gambar II.9 Simbol-Simbol Khusus Masukan/Keluaran dari Flowchart
(Sumber: Mulyadi, 2009)



Gambar II.10 Simbol-Simbol Pemrosesan Khusus dari Flowchart
(Sumber: Mulyadi, 2009)



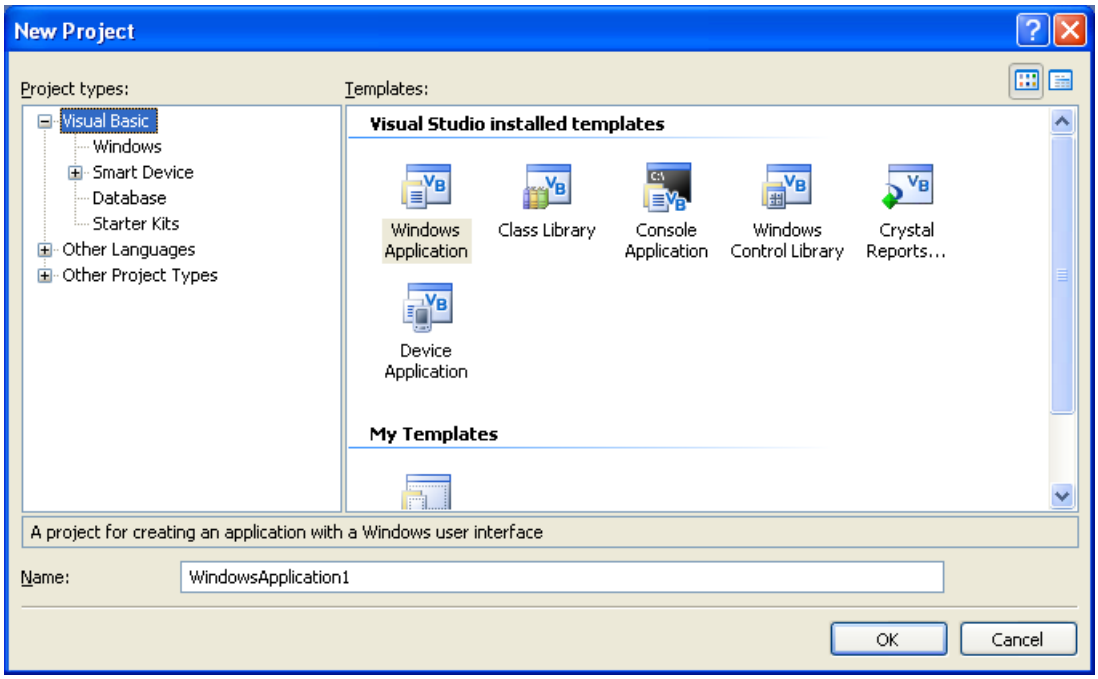
Gambar II.11 Simbol-Simbol Tambahan Flowchart

(Sumber: Mulyadi, 2009(

II.6. Software yang Digunakan

II.6.1 Microsoft Visual Basic.NET

Visual Basic.NET adalah generasi selanjutnya dari *Visual Basic*. *Visual Basic.NET* memungkinkan kita untuk membangun aplikasi *database client* atau *server* performa tinggi dan sangat cocok didampingkan dengan perangkat lunak *SQL Server* 2000.



Gambar II.12 Contoh Tampilan *New Project*
(Sumber: Kusrini, 2009)

Dalam dialog *New Project* terdapat beberapa jenis apliksi yang akan dibuat termasuk bahasa pemrograman yang digunakan. Jenis aplikasi yang dapat dibuat adalah (Junindar, 2008):

1. *Windows Application*: aplikasi yang paling umum dibuat, menggunakan *interfacewindows*. Biasanya, *Windows Application* merupakan *interface* aplikasi, sedangkan *logic* aplikasi terdapat di dalam *Class Library*. *Windows*

Application dapat berisi *form*, *class*, *XMLfile*, maupun *fileVB Script* dan *Jscript*.

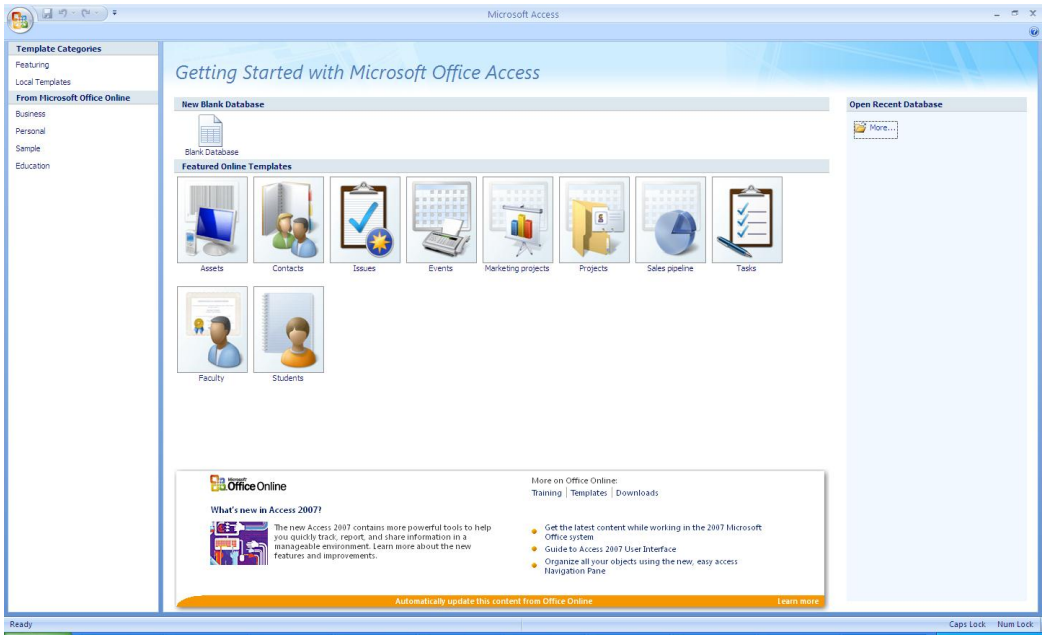
2. *Class Library*: fondasi dasar untuk membuat komponen yang menjalankan fungsi tertentu. *Class* merupakan fondasi dasar untuk membentuk obyek dalam pemrograman berorientasi obyek. *Class Library* tidak memiliki *interface* tertentu seperti *form*, tetapi dapat diakses oleh aplikasi lain untuk menjalankan berbagai fungsi yang terdapat di dalamnya. ogi *ActiveXDLL* (.dll) dan *ActiveXEXE* dalam pemrograman *VB6*.
3. *Windows Control Library*: tidak puas dengan built in control yang disediakan *VS.NET* .Anda dapat berkreasi membuat kontrol sendiri dan memasukkan berbagai fungsi yang Anda inginkan di dalam kontrol tersebut. Fasilitas untuk membuat kontrol tersebut adalah *Windows Control Library*.
4. *ASP.NET Web Application*: project yang digunakan untuk membuat aplikasi web. Teknologi yang digunakan adalah *ASP.NET* yang memiliki berbagai kelebihan dibandingkan *ASP* klasik.
5. *ASP.NET Web Service*: *Web service* merupakan salah satu ide utama dalam *.NET*. Anda dapat membuat *web service* dan meletakkannya di *web server* untuk diakses berbagai aplikasi. Sebuah *web service* dapat diakses oleh aplikasi *windows*, *web*, *console*, maupun *mobile device*. *Web service* hampir sama dengan *Class Library*.
6. *Console Application*: aplikasi dengan tampilan text mode atau DOS (*Disk Operating System*). Aplikasi jenis ini biasa digunakan sebagai *monitoring*

service atau *remote application* di mana sumber daya komputer dan *bandwidth* sangat terbatas.

7. *Windows Service*: aplikasi yang berjalan sebagai di *windows*, yang di-*load* bersamaan dengan proses *start up windows*. Aplikasi ini berjalan di *background* dan biasanya tidak memiliki *interface*.
8. *Web Control Library*: hampir sama dengan *Windows Control Library* tapi digunakan untuk aplikasi *web*.

II.6.2 Microsoft Access 2007

Microsoft Access adalah salah satu program aplikasi RDBMS (*Relational Database Management System*), dimana semua data yang ada disimpan dalam tabel-tabel yang terdiri dari atas lajur kolom dan baris. Dengan RDBMS, pengelolaan sebuah *database* akan mudah dilakukan walaupun jumlah datanya banyak dan kompleks.Dibandingkan dengan program aplikasi pembuatan *database* lain, *Microsoft Access* sangat mudah digunakan dan fleksibel dalam pembuatan dan perancangan suatu *database*. Perancangan dan pengelolaan *database* pada *Microsoft Access* meliputi pembuatan *Table*, *Form*, *Query*, *Macro*, *Modul* dan *Pages*, yang dapat dilihat pada Gambar 2.10:



Gambar II.13 Menu *Microsoft Access*
(Sumber :Djoko Pramono, 2010)

Sedangkan, tipe data yang ada dalam *Microsoft Access* adalah (Djoko Pramono, 2010):

1. *Text* : untuk menerima data teks sampai 255 karakter yang terdiri dari huruf, angka dan simbol grafik.
2. *Memo* : untuk menerima data teks sampai 65,535 karakter yang terdiri dari huruf, bilangan, tanda baca serta simbol grafik. Tipe data ini tidak dapat digunakan sebagai acuan untuk pengurutan data (indeks).
3. *Number* : untuk menerima digit, tanda minus dan titik desimal. Tipe data *number* mempunyai lima pilihan ukuran bilangan dan jumlah digit tertentu.
4. *Date/Time* : untuk menerima data tanggal dan waktu, serta nilai tahun yang dimulai tahun 100 sampai dengan tahun 9999.

5. *Currency* : untuk menerima data digit, tanda minus, dan tanda titik desimal dengan tingkat ketepatan 15 digit desimal di sebelah kiri tanda titik desimal dan 4 digit di sebelah kanan tanda titik desimal.
6. *Autonumber* : untuk menampilkan nomor urut otomatis, yaitu berupa data angka mulai dari 1 dengan nilai selisih 1.
7. *Yes/No* : tipe ini untuk menerima salah satu data dari dua nilai, yaitu *Yes/No, True/False* atau *On/Off*.
8. *OLE Object* : untuk menerima data yang berupa objek grafik, *spreadsheet*, foto digital, rekaman suara atau video yang dapat diambil dari program aplikasi lain. Ukuran maksimum adalah 1 *Gigabyte*.
9. *Hyperlink* : untuk menerima data yang berupa teks yang berwarna dan bergaris bawah serta grafik dimana tipe data ini berhubungan dengan jaringan.
10. *Attachment* : untuk menerima data yang berupa *file* gambar, *spreadsheet*, dokumen, grafik dan tipe data lain.
11. *Lookup Wizard* : untuk menampilkan satu dari beberapa tipe data yang ada dalam suatu daftar. Data tersebut dapat diambil dari tabel maupun *query* yang ada.