BAB II

TINJAUAN PUSTAKA

II.1. Kecerdasan Buatan

II.1.1. Pengertian Kecerdasan Buatan

Kecerdasan buatan berasal dari bahasa Inggris "*Artificial Intelligence"* atau disingkat AI, yaitu *intelligence* adalah kata sifat yang berarti cerdas, sedangkan *artificial* artinya buatan. Kecerdasan buatan yang dimaksud di sini menunjuk pada mesin yang mampu berfikir, menimbang tindakan yang akan diambil, dan mampu mengambil keputusan seperti yang dilakukan oleh manusia (T.Sutojo, dkk ; 2011 :1).

Berikut ini adalah beberapa definisi kecerdasan buatan yang telah didefinisikan oleh beberapa ahli :

1. Hebert Alexander Simon (June 15, 1916 February 9, 2001) :

   "Kecerdasan buatan (*Artificial Intelligence*) merupakan kawasan penelitian, aplikasi, dan instruksi yang terkait dengan pemograman komputer untuk melakukan sesuatu hal yang dalam pandangan adalah cerdas".

2. Rich and Knight (1991) :

   "Kecerdasaan buatan (AI) merupakan sebuah studi tentanbagaimana membuat computer melakukan hal-hal yang pada saat ini dapat dilakukan lebih baik oleh manusia".

3. Encyclopedia Britannica

   “Kecerdasan buatan (AI) merupakan cabang dari ilmu komputer yang dalam merepresentasi pengetahuan lebih banyak menggunakan symbol-simbol daripada bilangan dan memproses informasi berdasarkan metode heuristic atau dengan berdasarkan sejumlah aturan”.

4. Menurut Winston dan Prendergast (1984), tujuan dari kecerdasan buatan adalah :

   a. Membuat mesin menjadi lebih pintar (tujuan utama).
   b. Memahami apa itu kecerdasan (tujuan ilmiah).
   c. Membuat mesin lebih bermanfaat (tujuan *entrepreneurial*).

II.2. Sistem Pakar (*Expert System*)

II.2.1. Pengertian Sistem Pakar

Sistem Pakar (*Expert System*) adalah sistem yang dirancang untuk dapat menirukan keahlian seorang pakar dalam menjawab pernyataan dan memecahkan suatu masalah (T.Sutojo, dkk;2011:13).

Sistem pakar adalah sebuah sistem yang menggunakan pengetahuan manusia dimana pengetahuan tersebut dimasukkan ke dalam sebuah komputer dan kemudian digunakan untuk menyelesaikan masalah-masalah yang biasanya membutuhkan kepakaran atas keahlian manusia(T.Sutojo, dkk;2010:160).

Dengan kata lain, sistem pakar adalah sistem komputer yang ditunjukkan untuk meniru semua aspek (emulates) kemampuan pengambilan keputusan (*decision making*) seorang pakar. Sistem pakar memanfaatkan secara maksimal

pengetahuan khusus selayaknya seorang pakaruntuk memecahkaan masalah (Rika Rosnelly;2012:2).

II.2.2. Manfaat Sistem Pakar

Menurut (T.sutojo,dkk;2011:160-161) Sistem pakar menjadi sangat populer karena sangat banyak kemampuan dan manfaat yang diberikan, diantaranya:

1. Meningkatkan produktivitas, karena Sistem Pakar dapat bekerja lebih cepat daripada manusia.
2. Membuat seorang yang awam bekerja seperti layaknya seorang pakar.
3. Meningkatkan kualitas, dengan memberikan nasehat yang konsisten dan mengurangi kesalahan.
4. Mampu menangkap pengetahuan dan kepakaran seseorang.
5. Dapat beroperasi di lingkungan yang berbahaya.
6. Memudahkan akses pengetahuan seorang pakar.
7. Andal. Sistem pakar tidak pernah menjadi bosan dan kelelahan atau sakit.
8. Meningkatkan kapabilitas sistem komputer. Integritas Sistem Pakar dengan sistem komputer lain membuat sistem lebih efektif dan mencangkup lebih banyak sistem.
9. Mampu bekerja dengan informasi yang tidak lengkap atau tidak pasti. Berbeda dengan sistem komputer konvensional, Sistem Pakar dapat bekerja dengan informasi yang tidak lengkap. Pengguna dapat merespons dengan: “tidak tahu”

atau “tidak yakin” pada satu atau lebih pertanyaan selama konsultasi dan sistem pakar akan tetap memberikan jawabannya.

10. Bisa digunakan sebagai media pelengkap dalam pelatihan. Pengguna pemula bekerja dengan Sistem Pakar akan menjadi lebih berpengalaman karena adanya fasilitas penjelas yang berfungsi sebagai guru.
11. Meningkatkan kemampuan untuk menyelesaikan masalah karena Sistem Pakar mengambil sumber pengetahuan dari banyak pakar.

II.2.3. Kekurangan Sistem Pakar

Selain manfaat, sistem pakar juga memiliki beberapa kelemahan, diantaranya:

1. Memerlukan biaya yang sangat mahal untuk membuat dan memelihatanya.
2. Sulit dikembangkan karena keterbatasan keahlian dan ketersediaan pakar.
3. Sistem Pakar tidak selamanya 100% bernilai benar.

II.2.4. Ciri-ciri Sistem Pakar

Sistem Pakar memiliki beberapa ciri-ciri, diantanya sebagai berikut:

1. Terbatas pada domain keahlian tertentu.
2. Dapat memberikan penalaran untuk data-data yang tidak lengkap atau tidak pasti.
3. Dapat menjelaskan alasan-alasan dengan cara yang dapat dipahami.
4. Bekerja berdasarkan kaidah/*rule* tertentu.
5. Mudah dimodifikasi

6. Basis pengetahuan dan mekanisme inferensi terpisah.
7. Keluarannya bersifat anjuran.
8. Sistem dapat mengaktifkan kaidah secara searah yang sesuai, dituntun oleh dialog dengan pengguna (T.Sutojo, dkk ; 2011 :162).

II.2.5. Konsep Dasar Sistem Pakar

Konsep dasar sistem pakar meliputi enam hal berikut ini:

II.2.5.1. Kepakaran (*Expertise*)

Kepakaran merupakan suatu pengetahuan yang diperoleh dari pelatihan, membaca dan pengalaman. Kepakaran inilah yang memungkinkan para ahli dapat mengambil keputusan lebih cepat dan lebih baik daripada seseorang yang bukan pakar. Kepakaran itu sendiri meliputi pengetahuan tentang.

1. Fakta-fakta tentang bidang permasalahan tertentu.
2. Teori-teori tentang bidang permasalahan tertentu.
3. Aturan-aturan dan prosedur-prosedur menurut bidang permsalahan umumnya.
4. Atuan *heuristic*yang harus dikerjakan dalam suatu situasi tertentu.
5. Strategi global untuk memcahkan permasalahan
6. Pengetahuan tentang pengetahuan (*meta knowledge*).

II.2.5.2. Pakar (*Expert*)

Pakar adalah seseorang yang mempunyai pengetahuan, pengalaman dan metode khusus serta mampu menerapkannya untuk memecahkan masalah atau memberi nasehat. Seorang pakar harus mampu menjelaskan dan mempelajari hal-

hal baru yang berkaitan dengan topik permasalahan, jika perlu harus mampu menyusun kembali pengetahuan-pengetahuan yang didapatkan dan dapat memecahkan aturan-aturan serta menentukan relevansi kepakarannya. Jadi seorang pakar harus mampu melakasanankan kegiatan-kegiatan berikut ini:

1. Mengenali dan memformulasikan permasalahan.
2. Memecahkan permasalahan secara cepat dan tepat.
3. Menerangkan pemecahannya.
4. Belajar dari pengalaman.
5. Merestrukturasikan pengetahuan.
6. Memecahkan aturan-aturan.
7. Menentukan relevansi.

II.2.5.3. Pemindahan Kepakaran (*Transfering Expertisi*)

Tujuan dari Sistem Pakar adalah memindahkan kepakaran dari seorang pakar ke dalam komputer, kemudian kepada orang lain yang bukan pakar. Proses ini melibatkan empat kegiatan, yaitu:

1. Akuisisi pengetahuan (dari pakar atau sumber lain).
2. Representase pengetahuan (pada komputer).
3. Inferensi pengetahuan.
4. Pemindahan pengetahuan ke pengguna.

II.2.5.4. Inferensi (*Inferencing*)

Inferensi adalah sebuah prosedur (program) yang mempunyai kemampuan dalam melakukan penalaran. Inferensi ditampilkan pada suatu komponen yang disebut mesin inferensi yang mencangkup prosedur-prosedur mengenai pemecahan masalah. Semua pengetahuan yang dimiliki oleh seorang pakar disimpan pada basis pengetahuan oleh sistem pakar. Tugas mesih adalah mengammbil kesimpulan berdasarkan basis pengetahuan.

II.2.5.5. Aturan-aturan (*Rules*)

Kebanyakan *software* pakar komersil adalah sistem yang berbasis *rule* (*rule-based system*), yaitu pengetahuan disimpan terutama dalam bentuk *rule,* sebagai prosedure pemecahan masalah.

II.2.5.6. Kemampuan Menjelaskan (*Explanation Capability*)

Fasilitas lain dari sistem pakar adalah kemampuannya untuk menjelaskan saran atau rekomendasi yang diberikannya. Penjelasan dilakukan dalam subsistem yang disebut subsistem penjelasan (*explanation*). Bagian dari sistem ini memungkinkan sistem untuk memerikasa penalaran yang dibuatnya sendiri dan menjelaskan operasi-operasinya.

Karakteristik dan kemampuan yang dimiliki oleh sistem pakar berbeda dengan sistem konvensional. Perbedaan ini ditunjukkan pada Tabel II.1.

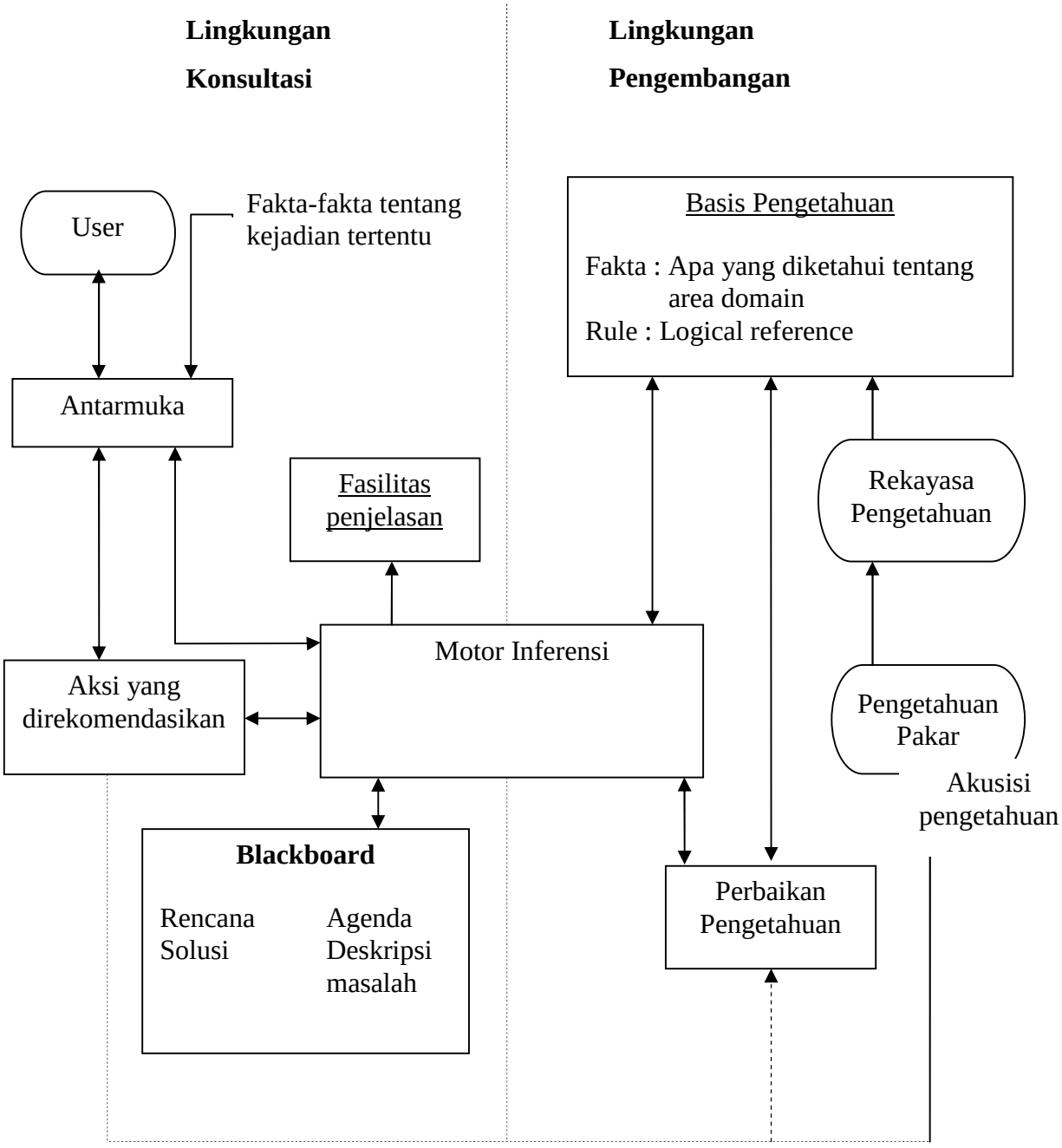
Tabel II.1. Perbandingan antara Sistem Konvensional dengan Sistem Pakar

Sistem Konvensional	Sistem Pakar
Informasi dan pemprosesannya biasanya digabungkan dalam satu program.	Basis pengetahuan diposahkan secara jelas dengan mekanisme inferensi.
Program tidak membuat kesalah (yang membuat kesalahan: Pemrogram atau Pengguna).	Program dapat berbuat kesalahan.
Biasanya tidak menjelaskan mengapa data masukan diperlakukan atau bagaimana output dihasilkan.	Penjelasan merupakan bagian terpenting dari semua sistem pakar.
Perubahan program sangat menyulitkan.	Perubahan dalam aturan-aturan mudah untuk dilakukan.
Sistem hanya bisa beroperasi setelah lengkap atau selesai.	Sistem dapat beroperasi hanya dengan aturan-aturan yang sedikit (sebagai prototipe awal).
Eksekusi dilakukan berdasarkan langkah demi langkah (*algoritmatic*).	Eksekusi dilakukan dengan menggunkaan *heuristic* dan logika pada seluruh basi pengetahuan.
Perlu informasi lengkap agar bisa beroperasi.	Dapat beroperasi dengan infomasi yang tidak lengkap atau mengandung ketidakpastian.
Menaipulasi efektif dari basis data yang besar.	Manipulasi efektif dari basis pengetahuan yang besar.
Menggunkan data.	Menggunakan pengetahuan.
Tujuan utama efisiensi.	Tujuan utama efektivitas.
Mudah berurusan dengan data kuantutatif.	Mudah berurusan dengan data kulaitatif.
Menagkap, menambah dan mendistribusikan akses ke data numerik atau informasi.	Menangkap, menambah dan mendistribusikan akses ke pertimbangan dan pengetahuan.

Sumber: (T.Sutojo, dkk;2011:165)

II.2.5.7. Struktur Sistem Pakar

Ada dua bagian penting dari sistem pakar, yaitu lingkungan pengembangan (development environment) dan lingkungan konsultasi (consultation environment). Lingkungan pengembangan digunakan oleh pembuat sistem pakar untuk membangun komponen – komponennya dan memperkenalkan pengetahuan kedalam knowledge base (basis pengetahuan). Lingkungan konsultasi digunakan oleh pengguna untuk berkonsultasi sehingga pengguna mendapatkan pengetahuan dan nasihat dari sistem pakar layaknya berkonsultasi dengan sorang pakar. Gambar II.1 menunjukkan komponen- komponen yang penting dalam sebuah sistem pakar.



Gambar II.1 Komponen-komponen yang Penting dalam Sebuah Sistem Pakar

Sumber : (T.Sutojo,dkk ;2011:167)

Keterangan:

1. Akuisi Pengetahuan

   Subsistem ini digunakan untuk memasukkan pengetahuan dari seorang pakar dengan cara merekayasa pengetahuan agar bisa diproses oleh komputer dan menaruhnya kedalam basis pengetahuan dengan format tertentu (dalam bentuk representasi pengetahuan). Sumber – sumber pengetahuan bisa diperoleh dari pakar, buku, dokumen multimedia, basis data, laporan riset khusus, dan informasi yang terdapat diweb.

2. Basis pengetahuan (*Knowledge Base*)

   Basis pengetahuan mengandung pengetahuan yang diperlukan untuk memahami merformulasikan, dan menyelesaikan masalah. Basis pengetahuan terdiri dari dua elemen dasar, yaitu:

   a. Fakta, misalnya situasi, kondisi, atau permasalahan yang ada.
   b. Rule (Aturan), untuk mengarahkan penggunaan pengetahuan dalam memecahkan masalah.

3. Mesin Inferensi (*Inference Engine*)

   Mesin inferensi adalah sebuah program yang berfungsi untuk memandu proses penalaran terhadap suatu kondisi berdasarkan pada basis pengetahuan yang ada, memanipulasi dan mengarahkan kaidah, model, dan fakta yang disimpan dalam basis pengetahuan untuk mencapai solusi atau kesimpulan. Dalam prosesnya, mesin inferensi mengunakan strategi pengendalian, yaitu strategi yang berfungsi sebagai panduan arah dalam melakukan proses penalaran. Ada

dua teknik pengendalian yang digunakan, yaitu *forward chaining, backward chaining,* dan gabungan dari kedua teknik tersebut.

a. Forward Chaining

Forward chaining adalah teknik pencarian yang dimulai dengan fakta yang diketahui, kemudian mencocokan fakta-fakta tersebut dengan bagian IF dari *rules* IF-THEN. Bila ada fakta yang cocok dengan bagian IF, maka rule tersebut dieksekusi. bila sebuah rule dieksekusi, maka sebuah fakta baru (bagian THEN) ditambahkan ke dalam database. setiap kali pencocokan, di mulai dari rule teratas. Setiap rule hanya boleh dieksekusi sekali saja. proses pencocokan berhenti bila tidak ada lagi rule yang bisa dieksekusi.

b. Backward Chaining

Backward chaining adalah metode inferensi yang bekerja mundur ke arah kondisi awal. proses diawali dari goal (yang berada dibagian THEN dari rule IF-THEN), kemudian pencarian mulai dijalankan untuk mencocokan apakah fakta-fakta yang ada cocok dengan premis-premis di bagian IF. Jika cocok, rule dieksekusi, kemudian hipotesis di bagian THEN ditempatkan di basis data sebagai fakta baru.

4. Daerah kerja (*Blackboard*)

Untuk merekam hasil sementara yang akan dijadikan sebagai keputusan dan untuk menjelaskan sebuah masalah yang sedang terjadi, sistem pakar membutuhkan *blackboard,* yaitu area pada memori yang berfungsi sebagai basis data. Tiga tipe keputusan yang dapat direkam pada *blackboard,* yaitu:

a. Rencana : bagaimana menghadapi masalah.

b. Agenda : aksi – aksi potensial yang sedang menunggu untuk dieksekusi.

c. Solusi : calon aksi yang akan dibangkitkan

5. Antarmuka Pengguna (*User Interface*)

   Digunakan sebagai media komunikasi antara pengguna dan sistem pakar. Komunikasi ini paling bagus bila disajikan dalam bahasa alami (*natural language*) dan dilengkapi dengan grafik, menu, dan formulis elektronik. Pada bagian ini akan terjadi dialog antara sistem pakar dan pengguna.

6. Subsistem Penjelasan (*Explanation Subsystem / Justifier*)

   Berfungsi memeberi penjelasan kepada pengguna, bagaimana suatu kesimpulan dapat diambil. Kemampuan seperti ini sangat penting bagi pengguna untuk mengetahui proses pemindahan keahlian pakar maupun dalam pemecahan masalah.

7. Sistem Perbaikan Pengetahuan (*Knowledge Refining System*)

   Kemapuan memperbaiki pengetahuan (*Knowledge Refining System*) dari seorang pakar diperlukan untuk menganalisa pengetahuan, belajar dari kesalhan masa lalu, kemudian memperbaiki pengetahuannya sehingga dapat dipakai pada masa mendatang. Kemampuan evaluasi diri seperti itu diperlukan oleh program agar dapat menganalisa alasan – alasan kesuksesan dan kegagalannya dalam mengambil kesimpulan. Dengan cara ini basis pengetahuan yang lebih baik dan penalaran yang lebih efektif akan dihasilkan (T.Sutojo, dkk ; 2011 :169).

8. Pengguna (*User)*

   Pada umumnya pengguna sistem pakar bukanlah seorang pakar (*non-expert*) yang membutuhkan solusi, saran atau pelatihan (*training*) dari berbagai permasalahan yang ada.

II.2.5.8. Karakteristik Sistem Pakar

Sistem Pakar pada umumnya dirancang untuk memenuhi beberapa karakteristik umum berikut ini :

1. Kinerja sangat baik (*high performance*). Sistem harus mampu memberikan respon berupa saran (*advice*) dengan tingkat kualitas yang sama dengan seorang pakar atau melebihinya.
2. Waktu respon yang baik (*adequate respon time*). Sistem juga harus mampu bekerja dalam waktu yang sama baiknya (*reasonable*) atau lebih cepat dibandingkan dengan seorang pakar dalam menghasilkan keputusan.
3. Dapat diandalkan (*good reliability*). Sistem harus dapat diandalkan dan tidak mudah rusak/ crash.
4. Dapat dipahami (*understandable*). Sistem harus mampu menjelaskan lngkah-langkah penalaran yang dilakukannya seperti seorang pakar.
5. Fleksibel (*fleksibility*). Sistem harus menyediakan mekanisme untuk menambah, mengubah, dan menghapus pengetahuan (Rika Rosnelly; 2012 :20)

II.3. Penyakit Mata

Mata merupakan suatu panca indera yang penting dalam kehidupan manusia untuk dapat melihat. Jika mata mengalami gangguan atau penyakit mata, maka akan berakibat fatal dalam kehidupan manusia. Jadi sudah semestinya mata merupan aknggota tubuh yang perlu dijaga dalam kehidupan sehari – hari.

Meskipun sangat penting sering kali kita lupa merawatnya secara baik yang dikarenakan kurangnya pengetahuan masyarakat untuk mencegah penyakit itu. Selain itu, terbatasnya sarana dan pelayanan kesehatan mata di puskesmas dan rumah sakit, serta kurangnya tenaga dokter spesialis mata yang memeriksa dan melakukan operasi mata, membuat gangguan penyakit mata tak tertangani sejak dini.

II.4. Metode *Forward Chaining*

II.4.1. Pengertian Metode *Forward Chaining*

Metode *Forward Chaining* adalah metode pencarian atau teknik pelacakan ke depan yang dimulai dengan informasi yang ada dan penggabungan *rule* untuk menghasilkan suatu kesimpulan atau tujuan. (Russel S,Norvig P, 2003). Pelacakan maju ini sangat baik jika bekerja dengan permasalahan yang dimulai dengan rekaman informasi awal dan ingin dicapai penyelesaian akhir, karena seluruh proses akan dikerjakan secara berurutan maju. Berikut adalah diagram *Forward Chaining* secara umum untuk menghasilkan sebuah *goal*. *Forward chaining* merupakan metode inferensi yang melakukan penalaran dari suatu masalah kepada solusinya. Jika klausa premis sesuai dengan situasi (bernilai *TRUE*), maka

proses akan menyatakan konklusi. *Forward chaining* adalah *data-driven* karena inferensi dimulai dengan informasi yang tersedia dan baru konklusi diperoleh. Jika suatu aplikasi menghasilkan *tree* yang lebar dan tidak dalam, maka gunakan *forward chaining*.

Tipe sistem yang dapat dicari dengan *Forward Chaining* :

1. Sistem yang dipersentasikan dengan satu atau beberapa kondisi.
2. Untuk setiap kondisi, sistem mecari rule-rule dalam *knowledge base* untuk *rule-rule* yang berkorespondensi dengan kondisi dalam bagian *IF*
3. Setiap *rule* dapat menghasilkan kondisi baru dari konklusi yang diminta pada bagian *THEN*. Kondisi baru ini ditambahkan ke kondisi lain yang sudah ada.
4. Setiap kondisi yang ditambahkan ke sistem akan diproses. Jika ditemui suatu kondisi baru dari konklusi yang diminta, sistem akan kembali ke langkah 2 dan mencari *rule-rule* dalam *knowledge* base kembali. Jika tidak ada konklusi baru, sesi ini berakhir.

Contoh :

Terdapat 10 aturan yang tersimpan dalam basis pengetahuan yaitu :

R1 : *if* A *and* B *then* C

R2 : *if* C *then* D

R3 : *if* A *and* E *then* F

R4 : *if* A *then* G

R5 : *if* F *and* G *then* D

R6 : *if* G *and* E *then* H

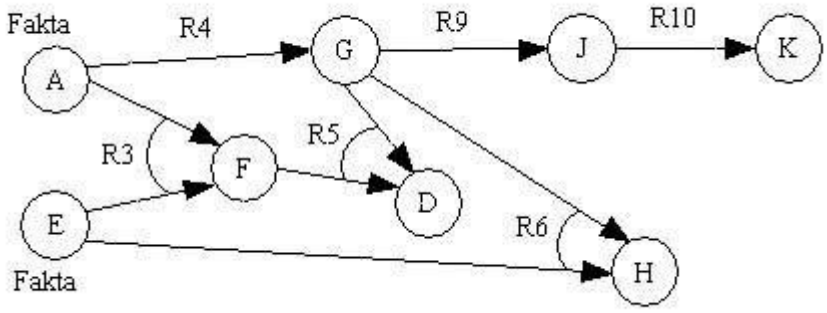
R7 : *if* C *and* H *then* I

R8 : *if* I *and* A *then* J

R9 : *if* G *then* J

R10 : if J *then* K

fakta awal yang diberikan hanya A dan E, ingin membuktikan apakah K bernilai benar. Proses penalaran forward chaining terlihat pada gambar II.1 dibawah :



Gambar II.2. Forward Chaining

Sumber : (R. Reisa, Jusak, P. Sudarmaningtyas / JSIKA Vol 2, No 2 (2013)/ ISSN 2338-137X)

II.5. Visual Basic 2010

II.5.1. Pengertian Visual Basic 2010

Microsoft Visual Basic 2010 (sering disingkat sebagai *VB* saja) merupakan sebuah bahasa pemrograman yang bersifat *event driven* dan menawarkan *Integrated Development Environment (IDE)* visual untuk membuat program aplikasi berbasis sistem operasi *Microsoft Windows* dengan menggunakan model pemrograman *Common Object Model (COM). Visual*

Basic merupakan turunan bahasa *BASIC* dan menawarkan pengembangan aplikasi komputer berbasis grafik dengan cepat, akses ke basis data menggunakan Data Access Objects (DAO), Remote Data Objects (RDO), atau ActiveX Data Object (ADO), serta menawarkan pembuatan kontrol ActiveX dan objek ActiveX.

Visual Basic merupakan turunan bahasa *BASIC* dan menawarkan pengembangan aplikasi komputer berbasis grafik dengan cepat, akses ke basis data menggunakan *Data Access Objects* (DAO), *Remote Data Objects* (RDO), atau *ActiveX Data Object* (ADO), serta menawarkan pembuatan kontrol *ActiveX* dan objek *ActiveX*. Beberapa bahasa skrip seperti *Visual Basic for Applications* (VBA) dan *Visual Basic Scripting Edition (VBScript),* mirip seperti halnya *Visual Basic*, tetapi cara kerjanya yang berbeda. (Jurnal Sistem Informasi, Vol. 6 No. 2; Adelia, Jimmy Setiawan ; 2011 : 114-115)

II.6. SQL Server 2008

II.6.1 Pengertian SQL Server 2008

SQL (*Structured Query Language*) adalah sebuah bahasa yang dipergunakan untuk mengakses data dalam basis data relasional. Bahasa ini secara *de facto* merupakan bahasa standar yang digunakan dalam manajemen basis data relasional. Saat ini hampir semua *server* basis data yang ada mendukung bahasa ini untuk melakukan manajemen datanya. (Jurnal Sistem Informasi, Vol. 6 No. 2; Adelia, Jimmy Setiawan ; 2011 : 115)

SQL Server 2008 adalah sebuah terobosan baru dari Microsoft dalam bidang *database* dibuat pada saat kemajuan dalam bidang *hardware* sedemikian pesat.

SQL Server adalah DBMS (Database Management *System*) yang dibuat oleh Microsoft untuk ikut berkecimpung dalam persaingan dunia pengolahan data menyusul pendahulunya seperti IBM dan Oracle.

II.7. *Unified Modeling Language* (UML)

II.7.1 Pengertian *Unified Modeling Language* (UML)

Unified Modeling Language (UML) adalah suatu alat untuk memvisualisasikan dan mendokumentasikan hasil analisa dan desain yang berisi sintak dalam memodelkan sistem secara visual (Braun, *et. al*.2011). Juga merupakan satu kumpulan konvensi pemodelan yang digunakan untuk menentukan atau menggambarkan sebuah sistem *software* yang terkait dengan objek (Whitten, *et. al*. 2004)

UML merupakan salah satu alat bantu yang sangat handal dalam bidang pengembangan sistem berorientasi objek karena UML menyediakan bahasa pemodelan visual yang memungkinkan pengembang sistem membuat *blue print* atas visinya dalam bentuk yang baku. UML berfungsi sebagai jembatan dalam berkomunikasikan beberapa aspek dalam sistem melalui sejumlah elemen grafis yang bisa dikombinasikan menjadi diagram. UML mempunyai banyak diagram yang dapat mengakomodasi berbagai sudut pandang dari suatu perangkat lunak yang akan dibangun.(Yuni Sugiarti; 2013:36-37)

Dengan menggunakan UML, kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemograman apapun. Tetapi karena UML juga menggunakan dan operation dalam konssep

dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa-bahasa berorientasi objek seperti C++, Java, C# atau VB.NET. Walaupun demikian UML tetap dapat digunakan untuk modeling aplikasi prosedural dalam VB atau C.

Seperti bahasa-bahasa lainnya, UML mengidentifikasikan notasi dan syntax/semantik. Notasi UML merupakan sekumpulan bentuk khusus untuk menggambarkan berbagai diagram piranti lunak. Setiap bentuk memiliki makna tertentu, dan UML syntax mendefinisikan bagaimana bentuk-bentuk tersebut dapat dikombinasikan. Notasi UML terutama diturunkan dari 3 notasi yang telah ada sebelumnya: Grady Booch OOD (*Object-Oriented Software Design*), Jim Rumbaugh OMT (*Object Modeling Technique*), dan Ivar Jacobson OOSE (*Object-Oriented Software Engineering*) (Yuni Sugiarti;2013:34).

Blok pembangunan utama UML adalah diagram. Beberapa diagram ada yang rinci (jenis *timming diagram*) dan lainnya ada yang bersifat umum (misalnya diagram kelas). Para pengembang sistem berorientasi objek menggunakan bahasa model untuk menggambarkan, membangundan mendokumentasikan sistem yang mereka rancang. UML memungkinkan para anggota team untuk bekerja sama dengan bahasa model yang sama dengan mengaplikasikan beragam sistem. Intinya UML merupakan alat komunikasi yang konsisten dalam mendukung para pengembang sistem saat ini.

II.7.2. *Use Case* Diagram

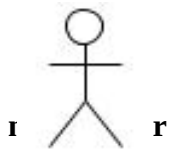
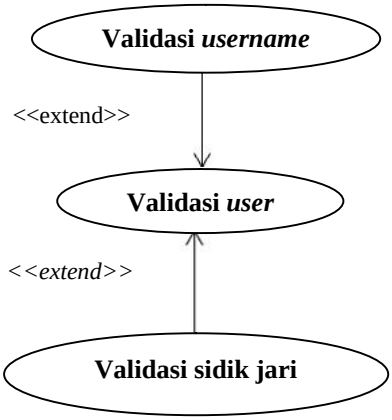
Dalam membuat sebuah sistem, langkah awal yang perlu dilakukan adalah menentukan kebutuhan. Terdapat dua jenis kebutuhan, yaitu kebutuhan

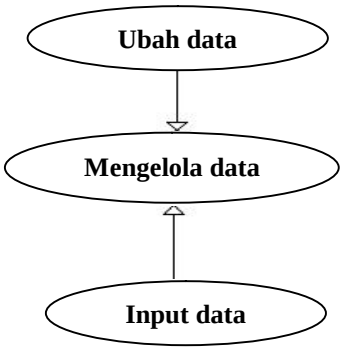
fungsional dan kebutuhan nonfungsional. Kebutuhan fungsional adalah kebutuhan pengguna dan stakeholder sehari-hari yang akan dimiliki oleh sistem, dimana kebutuhan ini akan digunakan oleh pengguna *stakeholder*. Sedangkan kebutuhan nonfungsional adalah kebutuhan yang memperhatikan hal-hal berikut yaitu performansi, kemudahan dalam menggunakan sistem, kehandalan sistem, keamanan sistem, keuangan, legalitas, dan operasional.

Kebutuhan fungsi akan digambarkan melalui sebuah diagram yang dinamakan diagram use case. *Use case* diagram atau diagram use case merupakan pemodelan untuk menggambarkan kelakuan (*behavior*) sistem yang akan dibuat. Diagram *use case* mendeskripsikan sebuah interaksi antar satu atau lebih *actor* dengan sistem yang akan dibuat. Dengan pengertian yang cepat, diagram *use case* digunakan untuk mengetahui fungsi apa saja yang ada didalam sebuah sistem dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut. Terdapat beberapa simbol dalam menggambarkan diagram *use case*, yaitu *use case*, *actor* dan relasi. Hal perlu diingat mengenai diagram *use case* adalah diagram *use case* bukan menggambarkan tampilan antarmuka (*user interface*), arsitektur dari sistem, kebutuhan nonfungsional, dan tujuan performansi. Sedangkan untuk penamaan *use case* adalah nama didefinisikan sesimpel mungkin, dapat dipahami dan menggunakan kata kerja (Yuni Sugiarti; 2013:41).

Berikut ini pada tabel II.3. adalah simbol-simbol yang ada pada diagram *use case*:

Tabel II.3. Simbol – Simbol *Use Case* Diagram

Simbol	Deskripsi
Use case Nama *use case*	Fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor, biasanya dinyatakan dengan menggunakan kata kerja diawal diawal frase nama *use case*
Aktor / *actor* 	orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari *actor* adalah gambar orang, tapi *aktor* belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal nama *aktor*.
Asosiasi/ *association* 	Komunikasi antara *actor* dan *use case* yang berpartisipasi pada *use case* atau *use case* memiliki interaksi dengan aktor
Ekstensi / *extend* <<*extend*>> 	Relasi *use case* tambahan kesebuah *use case* dimana *use case* yang ditambahkan dapat berdiri sendiri walau tanpa *use case* tambahan itu: mirip dengan prinsip *inheritance* pada pemograman berorientasi objek; biasanya *use*
	case tambahan memiliki nama depan yang sama dengan *use case* yang ditambahkan, misal  Arah panah mengarah pada *use case* yang ditambahkan; biasanya *use case* yang menjadi *extend*-nya merupakan jenis yang sama dengan *use case* yang menjadi induknya.

Generalisasi/*generalization* 	Hubungan generalisasi dan spesialisasi (umum – khusus) antara dua buah *use case* dimana fungsi yang satu adalah fungsi yang lebih umum dari lainnya, misalnya:  Arah panah mengarah pada *use case* yang menjadi generalisasinya (umum).
Menggunakan / *include* / *uses* <<*include*>>  <<*uses*>> 	Relasi *use case* tambahan ke sebuah *use case* dimana *use case* yang ditambahkan memerlukan *use case* ini untuk menjalankan fungsinya atau sebagai syarat *use case* ini. Ada dua sudut pandang yang cukup besar mengenai *include* di *use case* : *include* berarti *use case* yang ditambahkan akan selalu dipanggil saat *use case* tambahan dijalankan. *include* berarti *use case* yang tambahan akan selalu melakukan pengecekan apakah *use case* yang ditambahkan telah dijalankan sebelum *use case* tambahan dijalankan.

Sumber : (Rosa A.S, M.Shalahuddin; 2013; 155-158).

II.7.3. *Class Diagram* (Diagram Kelas)

Diagram kelas atau *class diagram* menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. kelas memiliki apa yang disebut atribut dan metode atau operasi.

1. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas.
2. Atribut mendeskripsikan property dengan sebaris teks didalam kotak kelas tersebut.
3. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

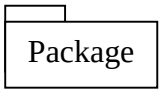
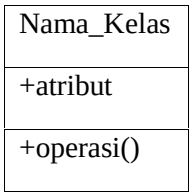
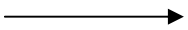
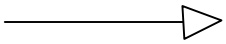
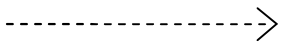
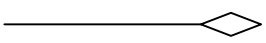
Diagram kelas mendeskripsikan jenis-jenis objek dalam sistem dan berbagai hubungan statis yang terdapat diantara mereka. Diagram kelas juga menunjukkan properti dan operasi sebuah kelas dan batasan-batasan yang terdapat dalam hubungan-hubungan objek. Diagram kelas menggambarkan struktur dan deskripsi class, package dan objek beserta hubungan satu sama lain seperti containment, perwarisan, asosiasi, dan lain-lain (Yuni Sugiarti;2013:57).

Kelas memiliki tiga area pokok :

1. Nama
2. Atribut
3. Operasi

Berikut adalah simbol-simbol yang ada pada diagram kelas :

Tabel II.4. Simbol-Simbol Diagram Kelas

Simbol	Deskripsi
Package 	Package merupakan bungkusan dari satu kelas atau lebih
Operasi 	Kelas pada struktur system
Antarmuka / *interface*  Nama *interface*	Sama dengan konsep *interface* dalam pemrograman berorientasi objek
Asosiasi / *association* 	Relasi antar kelas dengan makna umum, asosiasi biasanya juga disertai dengan *multiplicity*
Asosiasi berarah / *Directed association* 	Relasi antar kelas dengan makna kelas yang satu digunakan oleh kelas yang lain, asosiasi biasanya juga disertai dengan *multiplicity*
Generalisasi 	Relasi antar kelas dengan makna generalisasi-spesialisasi (umum khusus)
Kebergantungan 	Relasi antar kelas dengan makna Kebergantungan antar kelas
Agregasi / *aggregation* 	Relasi antar kelas dengan makna Semua bagian (*whole part*)

Sumber : (Yuni Sugiarti;2013:59).

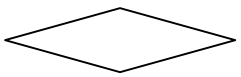
II.7.4. *Activity Diagram* (Aktivitas)

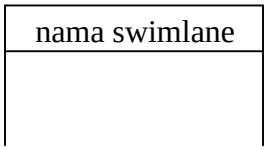
Diagram aktivitas atau *activity diagram* menggambarkan *workflow*(aliran kerja) atau aktivias dari sebuah sistem atau proses bisnis. *Activity diagram* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, decision yang mungkin terjadi, dan bagaimana mereka berakhir.

Activity Diagram merupakan *state* diagram khusus, dimana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tdak menggambarkan behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarka proses- proses dan jalur-jalur aktivitas dari level atas secara umum (Yuni Sugiarti;2013:75).

Berikut adalah tabel II.5. yang menggambarkan simbol-simbol yang digunakan pada diagram aktivitas :

Tabel II.5. Simbol *Activity Diagram*

Simbol	Deskripsi
Status awal 	Status awal aktivitas sistem, sebuah diagram aktivitas memiliki sebuah status awal.
Aktivitas 	Aktivitas yang dilakukan sistem, aktivitas biasanya diawali dengan kata kerja.
Percabangan / *decision* 	Asosiasi percabangan dimana jika ada pilihan aktivitas lebih dari satu.
Penggabungan / *join*	Asosiasi penggabungan dimana lebih dari satu aktivitas digabungkan menjadi satu.

Status akhir 	Status akhir yang dilakukan sistem, sebuah diagram aktivitas memiliki sebuah status akhir.
Swimlane 	Memisahkan organisasi bisnis yang bertanggungjawab terhadap aktivitas yang terjadi.

Sumber : (Rosa A.S, M.Shalahuddin; 2013; 161-163).

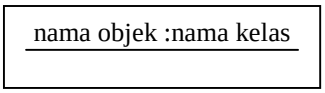
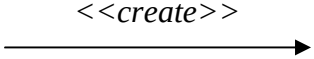
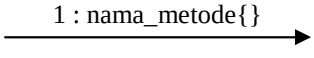
II.7.5. *Sequence* Diagram

Diagram sekuen menggambarkan kelakuan/perilaku objek pada *use case* dengan mendeskripsikan waktu hidup objek dan message yang dikirimkan dan diterima antar objek.

Banyaknya diagram *sequence* yang digambarkan adalah sebanyak pendefenisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefenisikan interaksi jalanya pesan sudah dicakup pada diagram sekuen sehingga semakin banyak *use case* didefenisikan maka diagram sekuen yang harus dibuat juga semakin banyak (Yuni Sugiarti;2013:69).

Berikut adalah tabel II.6. yang menerangkan simbol-simbol yang ada pada diagram sekuen:

Tabel II.6. Diagram *Sequence*

Simbol	Deskripsi
Aktor  **nama aktor**	orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat di luar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari *actor* adalah gambar orang, tapi *actor* belum tentu merupakan orang; biasanya dinyatakan menggunakan kata benda di awal nama *actor*.
Garis hidup / *lifeline*	Menyatakan kehidupan suatu objek.
Objek 	Menyatakan objek yang berinteraksi pesan.
Waktu aktif 	Menyatakan objek dalam keadaan aktif dan berinteraksi, semua yang terhubung dengan waktu aktif ini adalah tahapan yang dilakukan didalamnya.
Pesan tipe create 	Menyatakan suatu objek membuat objek yang lain, arah panah mengarah pada objek yang dibuat
Pesan tipe call 	Menyatakan suatu objek memanggil operasi / metode yang ada pada objek lain atau dirinya sendiri.

Pesan tipe send 1 : masukan 	Menyatakan bahwa suatu objek mengirimkan data / masukan / informasi ke objek lainnya, arah panah mengarah pada objek yang dikirimi.
Pesan tipe return 1 : keluaran 	Menyatakan bahwa suatu objek yang telah menjalankan suatu operasi atau metode menghasilkan suatu kembalian ke objek tertentu, arah panah mengarah pada objek yang menerima kembalian.
Pesan tipe destroy <<*destroy*>> 	Menyatakan suatu objek mengakhiri hidup objek yang lain, arah panah mengarah pada objek yang diakhiri, sebaiknya jika ada create maka ada *destroy*.

Sumber : (Rosa A.S, M.Shalahuddin; 2013; 165-167).

II.8. Basis Data

Basis data adalah sekumpulan fakta berupa representasi tabel yang saling berhubungan dan disimpan dalam media penyimpanan secara digital. Suatu basis data terdiri dari sekumpulan tabel yang saling berelasi ataupun tidak berelasi. semua tabel tersebut merupakan representasi tempat untuk penyimpanan data yang mendukung fungsi dari basis data tersebut pada suatu sistem (Yudi Priadi ; 2014:1-2).

II.8.1. Basis Data Relational

Basis data *relational* diciptakan oleh seorang peneliti dari IBM yang bernama Dr. E. F. Codd, kemudian dikembangkan terus oleh peneliti lainnya

hingga saat ini. Prinsipnya, model basis data *relational* digunakan sebagai suatu cara untuk mengelompokan data dari sebuah kumpulan data yang besar. hal ini dapat dilakukan dengan cara menghapus duplikasi dari suatu data melalui proses yang disebut normalisasi. Proses ini terdiri dari beberapa langkah yang akan menjadi suatu bentuk normal. Hasilnya berupa bahasa umum untuk melakukan akses terhadap suatu basis data, yang disebut dengan *Strucutured Query Language* (SQL), sehingga dimungkinkan untuk melakukan *query* terhadp organisasi struktur datanya. SQL sebagai bahasa pada basis data telah menjadi standar untuk semua perusahaaan pengembang basis data hingga saat ini (Yudi Priadi ; 2014: 14).

II.9. Normalisasi Data

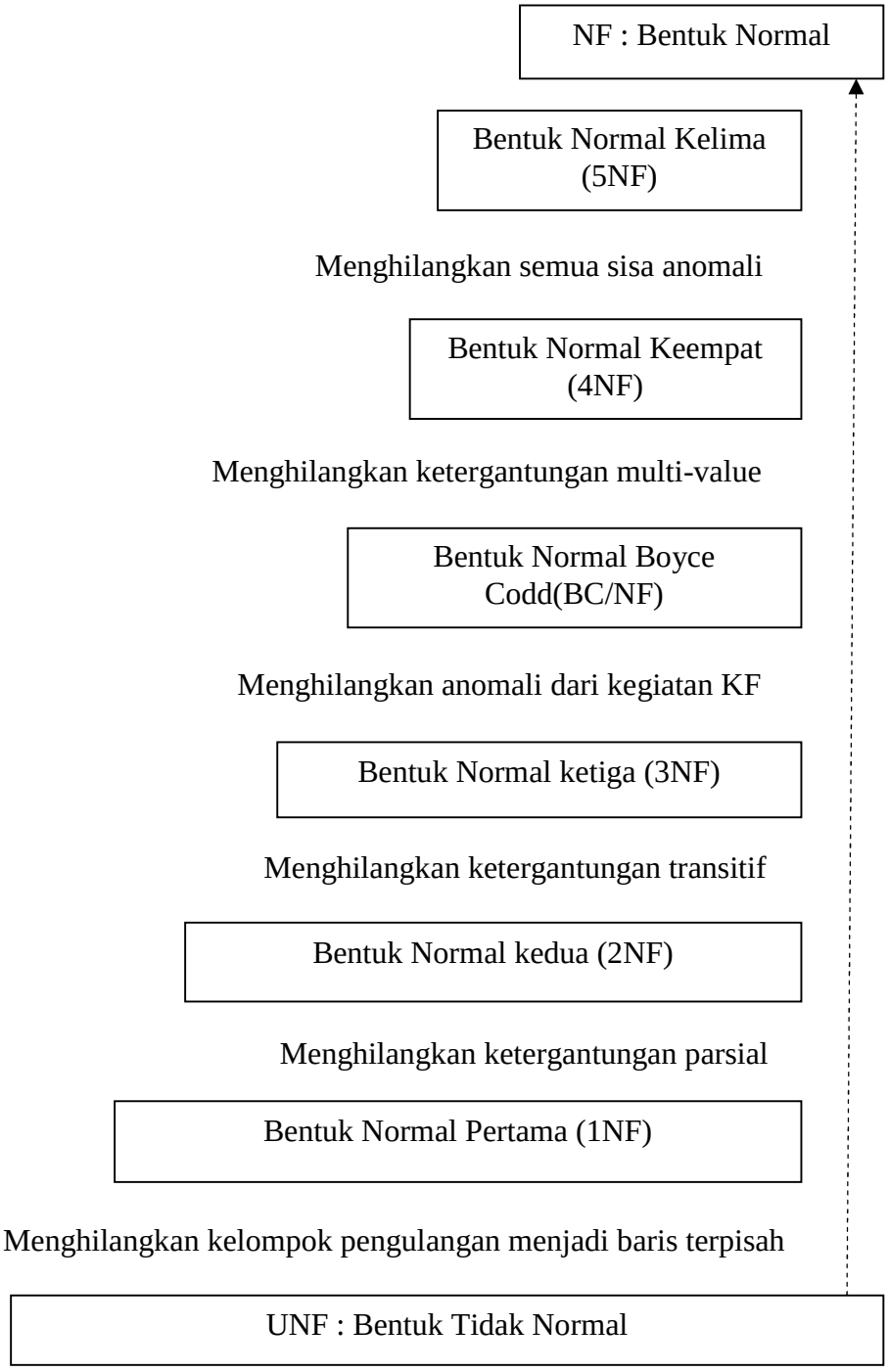
Normalisasi merupakan tahapan proses sistematis yang dilakukan pada struktur tabel basis data menjadi struktur tabel yang memiliki integritas data, sehingga tidak memiliki data anomali pada saat melakukan insert, delete, dan *update.* Kegiatan normalisasi adalah melakukan dekomposisi atau penguraian tabel beserta datanya, menjadi tabel yang normal menurut konsep RDBMS. Dekomposisi diawali dengan melakukan analisis pada suatu tabel atau beberapa contoh formulir yang sudah memiliki data lengkap utuk suatu basis data, tetapi masih dalam bentuk yang tidak normal (UNF). Oleh karena itu, agar dapat memenuhi syarat bentuk normal pertama (1NF), pada setiap barisnya diisikan suatu *value* dengan kelompok data yang sama, berdasarkan suatu atribut key. Dengan demikian, kelompok pengulangan dalam suatu baris dapat dihilangkan,

karena sudah tidak terdapat *value* yang kosong untuk setiap *field* dan *record*-nya (Yudi Priadi ; 2014: 67- 68).

Setelah memenuhi syarat bentuk normal pertama (1NF), proses berikutnya adalah menghilangkan ketergantungan secara persial, yaitu dengan cara melakukan dekomposisi tabel menjadi beberapa kelompok berdasarkan *field* yang memiliki status sebagai *key*. Hal in dapat dilakukan oleh salah satu *field* saja, dengan tetap tidak mengubah arti relasi dan ketergantungannya. oleh sebab itu, disebut ketergantungan fungsional sebagian (*partially functional*), sehingga syarat bentuk normal kedeu (2NF) sudah tercapai (Yudi Priadi ; 2014:68).

Bentuk normal kedua (2NF) merupakan syarat yang harus dimiliki untuk menuju bentuk normal ketiga (3NF). Pada proses ini , dilakukan dengan menghilangkan ketergantungan secara transistif, yaitu konsep untuk tabel dari hasil relasi yang didalamnya terdapat ketergantungan secara tidak langsung pada beberapa atributnya. Pada umunya, proses normalisai suda dapat tercapai pada bentuk normal ketiga (3NF), yaitu dengan menghasilkan tabel yang tidak mengalami anomali basis data pada saat proses *insert*, *delete*, dan *update* (Yudi Priadi ; 2014:68)

Berikut ini pada gambar II.2. adalah tahapan aturan proses normalisasi :



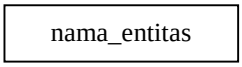
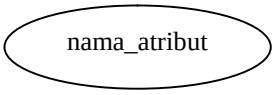
Gambar II.3. Tahapan Aturan Proses Normalisasi

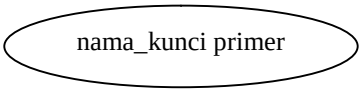
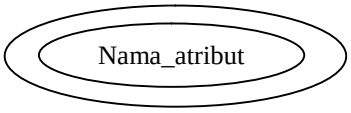
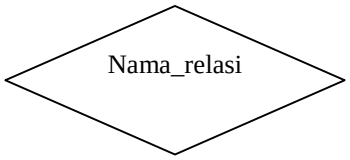
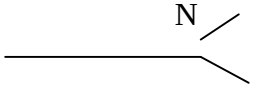
Sumber : (Yudi Priyadi;2014:72).

II.10. ERD (*Entity Relationship Diagram*)

Pemodelan awal basis data yang paling banyak digunakan adalah menggunakan ERD (*Entity Relationship Diagram*). ERD dikembangkan berdasarkan teori himpunan dalam bidang matematika. ERD digunakan untuk pemodelan basis data relasional. Sehingga jika penyimpanan basis data menggunakan OODBMS maka perancangan basis data perlu menggunakan ERD. ERD memiliki beberapa aliran notasi seperti notasi Chen (dikembangkan oleh Peter Chen), Barker (dikembangkan oleh Richard Barker, Ian Palmer, Harry Ellis), notasi Crow's Foot, dan beberapa notasi lain. Namun yang banyak digunakan adalah notasi Chen. Berikut pada Tabel II.7. adalah simbol – simbol yang digunakan pada ERD dengan notasi Chen : (Rosa A.S, M.Shalahuddin; 2013; 50-51).

Tabel II.7. Simbol-Simbol ERD

Simbol	Deskripsi
Entitas / *entity* 	Entitas merupakan data inti yang akan disimpan; bakal tabel pada basis data; benda yang memiliki data dan harus disimpan datanya agar dapat diakses oleh aplikasi komputer; penamaan entitas biasanya lebih ke kata benda dan belum merupakan nama tabel.
Atribut 	*Field* atau kolom data yang butuh disimpan dalam suatu entitas.

Atribut kunci primer 	*Field* atau kolom data yang butuh disimpan dalam suatu entitas dan digunakan sebagai kunci akses *record* yang diinginkan; biasanya berupa id; kunci primer dapat lebih dari satu kolom, asalkan kombinasi dari beberapa kolom tersebut dapat bersifat unik (berbeda tanpa ada yang sama).
Atribut multinilai / *multivalue* 	*Field* atau kolom data yang butuh disimpan dalam suatu entitas yang dapat memiliki nilai lebih dari satu.
Relasi 	Relasi yang menghubungkan antar entitas; biasanya diawali dengan kata kerja.
Asosiasi / *association* 	Penghubung antara relasi dan entitas dimana dikedua ujungnya memiliki *multiplicity* kemungkinan jumlah pemakaian, kemungkinan jumlah maksimum keterhubungan antara entitas satu dengan yang lain disebut kardinalitas. Misalkan ada kardinalitas 1 ke N atau sering disebut dengan *one to many* menghubungkan entitas A dan B maka.

Sumber : (Rosa A.S, M.Shalahuddin; 2013; 50-51).

Ada beberapa relasi yang dapat terjadi, yaitu :

1. Satu ke satu (1:1), mempunyai arti bahwa setiap satu entitas pada himpunan entitas A hanya dapat berelasi dengan satu saja pada himpunan entitas B.

2. Satu ke banyak (1:N), mempunyai arti bahwa setiap satu entitas pada himpunan entitas A dapat berelasi dengan banyak entitas pada himpunan entitas B.
3. Banyak ke satu (N:1), mempunyai arti bahwa beberapa entitas pada himpunan entitas A hanya dapat berelasi dengan satu entitas saja pada himpunan entitas B.
4. Banyak ke banyak (N:N), mempunyai arti bahwa beberapa entitas pada himpunan entitas A dapat berelasi dengan banyak entitas pada himpunan entitas B.