BAB II

TINJAUAN PUSTAKA

II.1. Sistem

Sistem adalah kumpulan dari elemen-elemen yang berinteraksi untuk mencapai suatu tujuan tertentu (Jogiyanto; 2005: 2).

II.2. Pakar

Pakar adalah seseorang yang mempunyai pengetahuan dan metode khusus serta mampu menerapkannya untuk memecahkan masalah atau memberi nasehat (T.Sutojo; 2011: 163).

II.3. Sistem Pakar

II.3.1. 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; 2011: 13).

Istilah sistem pakar berasal dari istilah *knowledge-based expertt system.* Istilah ini muncul karena untuk memecahkan masalah, sistem pakar menggunakan pengetahuan seorang pakar yang dimasukkan ke dalam komputer. Seorang yang bukan pakar menggunakan sistem pakar untuk meningkatkan kemampuan pemecahan masalah, sedangkan seorang pakar menggunakan sistem pakar untuk *knowledge assistant.* Berikut adalah beberapa pengertian sistem pakar:

1. Turban (2001, p402)

   “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”.

2. Jackson (1999, p3)

   “Sistem pakar adalah program komputer yang merepresentasikan dan melakukan penalaran dengan pengetahuan beberapa pakar untuk memecahkan masalah atau memberikan saran”.

3. Luger dan Stubblefield (1993, p308)

   “Sistem pakar adalah program yang berbasiskan pengetahuan yang menyediakan solusi ‘kualitas pakar’ kepada masalah-masalah dalam bidang (domain) yang spesifik” (T.Sutojo; 2011: 160).

II.3.2. Manfaat Sistem Pakar

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 aplikasi.
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 merespon dengan: "tidak tahu" atau "tidak yakin" pada satu atau lebih pertanyaan selama konsultasi dan sistem pakar tetap akan 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 (T.Sutojo; 2011: 160-161).

II.3.3. 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 (T.Sutojo; 2011: 161).

II.3.4. 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; 2011: 162).

II.3.5. Konsep Dasar Sistem Pakar

Konsep dasar sistem pakar meliputi enam hal berikut ini:

II.3.5.1. 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.3.5.2. Pakar (*Expert*)

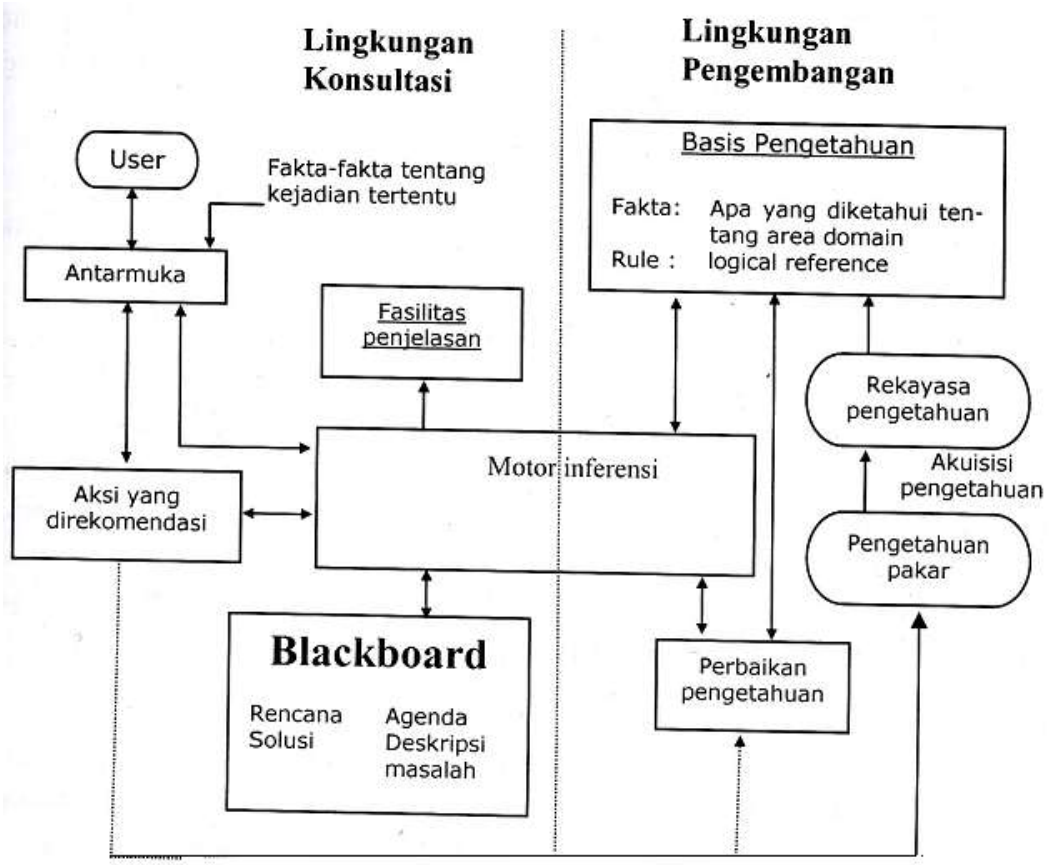
Pakar adalah seseorang yang mempunyai pengetahuan, pengalaman dan metode khusus serta mampu menerapkannya untuk memecahkan masalah atau memberi nasehat. Seorang pakar harus mampu menjelaskan dan mempelajari hal-hal baru yang berkaitan dengan topik permasalahan, jika perlu harus mampu menyusun kembali 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 (T.Sutojo; 2011: 163-164).

II.3.5.3. 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 ke dalam *knowledge base* (basis pengetahuan). Lingkungan konsultasi digunakan oleh pengguna untuk berkonsultasi sehingga pengguna mendapatkan pengetahuan dan nasihat dari Sistem Pakar layaknya berkonsultasi dengan seorang 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. Sutejo; 2011: 167)

Keterangan:

1. Akuisisi Pengetahuan

   Subsistem ini digunakan untuk memasukkan pengetahuan dari seorang pakar dengan cara merekayasa pengetahuan agar bisa diproses oleh komputer dan manaruhnya ke dalam 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 di web.

2. Basis Pengetahuan (*Knowledge Base*)

   Basis pengetahuan mengandung pengetahuan yang diperlukan untuk memahami, memformulasikan, 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 menggunakan strategi pengendalian, yaitu strategi yang berfungsi sebagai panduan arah dalam melakukan proses

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

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 formulir elektronik. Pada bagian ini akan terjadi dialog antara sistem pakar dan pengguna.

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

   Berfungsi memberi 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*)

   Kemampuan memperbaiki pengetahuan (*knowledge refining system*) dari seorang pakar diperlukan untuk menganalisis pengetahuan, belajar dari

kesalahan 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.

8. Pengguna (*User)*

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

II.3.5.4. 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.3.5.5. 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 (T.Sutojo; 2011: 164).

II.3.5.6. 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.3.5.7. 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 (T.Sutojo; 2011: 165).

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	Mudah berurusan dengan data

kuantutatif.	kulaitatif.
Menagkap, menambah dan mendistribusikan akses ke data numerik atau informasi.	Menangkap, menambah dan mendistribusikan akses ke pertimbangan dan pengetahuan.

Sumber: (T. Sutejo; 2011: 165)

II.4. Permasalahan Pengecatan Finishing Pada Mobil

Adapun jenis-jenis permasalahan pengecatan *finishing* pada mobil yaitu:

1. *Bleeding*

   Yaitu suatu kondisi yang disebabkan *Pigment* dari cat warna asli larut kepermukaan atau cat larut dalam *thinner* dari cat warna yang baru dan melunturinya. Tanda – tanda *bleeding* adalah permukaan cat berubah warna dan pada warna *metalic*.

2. *Blistering*

   Yaitu suatu kondisi yang disebabkan pembersihan kotoran minyak ataupun bahan kimia yang tidak sempurna dari permukaan. Akibatnya permukaan yang kotor tersebut berfungsi sebagai *spon* air yang setiap saat bisa meletus disebabkan suhu udara, penggunaan *thinner* atau *reducer* yang tidak sesuai terutama apabila cat di*spray*kan terlalu kering atau dengan tekanan yang terlalu tinggi mengakibatkan udara maupun air terjebak di cat, serta aplikasi cat yang terlalu tebal dan waktu pengeringan belum cukup untuk pengecatan

selanjutnya. Tanda–tanda *blistering* adalah permukaan gelembung-gelembung dan terdapat udara yang terperangkap.

3. *Blushing*

   Yaitu suatu kondisi yang disebabkan karena pengecatan dilakukan pada saat udara lembab menyebabkan penguapan air terjebak di dalam cat basah, tekanan udara yang sangat tinggi dari *spray*, *thinner* yang tidak sesuai, serta pengecatan dengan *spray* di *temperature* yang rendah mengakibatkan udara *spray* jatuh dibawah titik embun. Tanda-tanda dari *blushing* adalah permukaan memutih dan seperti berkabut.

4. *Chalking*

   Yaitu suatu kondisi yang disebabkan karena pengaruh cuaca, penyebab utama ialah "sinar *ultra violet*". Biasanya terjadi pada lapisan cat yang berwarna terang (keputih-putihan) akan mengapur dan membuat penglihatan pada permukaan seperti buram dan bubukan. Tanda-tanda *chalking* adalah permukaan cat kusam dan terdapat serbuk-serbuk halus.

5. *Checking*

   Yaitu suatu kondisi yang disebabkan karena pengecatan yang terlalu tebal serta persiapan dan permbersihan permukaan yang tidak sempurna. Tanda-tanda *checking* adalah permukaan cat retak-retak dan Bentuknya lebih paralel dan panjangnya sampai 18 *inchi*.

6. *Cracking*

   Yaitu suatu kondisi yang disebabkan karena cat yang terlalu tebal, pengadukan yang tidak merata, pengeringan yang tidak sempurna antar

pengecatan serta penggunaan *additive* yang tidak sesuai. Tanda-tanda *cracking* adalah permukaan cat retak-retak.

7. *Crazing*

   Yaitu suatu kondisi yang disebabkan karena tempat pengecatan terlalu dingin yang mana menyebabkan *solvent* bereaksi melunakan. Tanda-tanda *crazing* adalah permukaan cat retak-retak dan ada cabang kecil seperti cakar ayam tampil di bagian yang tidak menentu.

8. *Featheredge*

   Yaitu suatu kondisi yang disebabkan karena pengeringan yang tidak sempurna antar *under coat* sehingga *solvent* terjebak di cat berikutnya, penggunaan *thinner* yang tidak sesuai, pembersihan permukaan yang tidak sempurna mengakibatkan *primer surfacer coat* tidak menutup disebabkan kurangnya daya rekat serta pengeringan *primer* yang tidak sempurna dengan menggunakan *spray gun compressor* mengakibatkan kekeringan sebelum *solvent* bereaksi. Tanda–tanda *featheredge* adalah permukaan cat retak-retak dan retakan di ujung-ujung permukaan.

9. *Fish Eyes*

   Yaitu suatu kondisi efek sampingan dari pengecatan terakhir yang tidak bersih dari *silicone* dan *wax*, mengakibatkan kondisi permukaan cat meletup. Tanda-tanda *fish eyes* adalah permukaan gelembung-gelembung dan seperti bentuk mata ikan.

10. *Lifting*

    Yaitu suatu kondisi yang disebabkan karena *solvent* cat yang baru diaplikasi menyerang *under coat* disebabkan bahan yang tidak cocok, pengering *coat* yang tidak sempurna anatar pengecatan, pembersihan permukaan yang tidak sempurna serta penggunaan *thinner* atau *reducer* tidak sesuai. Tanda-tanda *lifting* adalah permukaan berkerut-kerut dan cat tidak bisa kering.

11. *Mottling*

    Yaitu suatu kondisi yang disebabkan karena pecahan cat yang terapung dalam pengecatan. Tanda-tanda *motling* adalah permukaan tampil titik-titik dan permukaan terkelupas.

12. *Orange Peel*

    Yaitu suatu kondisi yang disebabkan karena penyetelan *spray gun* kurang tepat, tekanan angin terlalu rendah, pengeringan yang tidak sempurna dengan *spray gun* mengakibatkan titisan cat kering sebelum merata, penggunaan *thinner* atau *reducer* yang tidak sesuai serta pengadukan cat yang tidak merata. Tanda-tanda *orange peel* adalah permukaan berkerut-kerut dan seperti kulit jeruk.

13. *Peeling*

    Yaitu suatu kondisi yang disebabkan karena bahan tidak diaduk dengan baik, penggunaan *thinner* kurang sesuai, lapisan dasar menggunakan bahan yang berkualitas jelek serta tidak di amplas antar lapisan. Tanda-tanda *peeling* adalah cat kehilangan daya lekat dan bahan cat tidak bisa menyatu dipermukaan.

14. *Pinholing*

    Yaitu suatu kondisi yang disebabkan karena pengeringan yang tidak sempurna, pengeringan menggunakan *compressor* mengakibatkan udara tampil dan *solvent* dipermukaan cat, *compressor* yang terkontaminasi dengan air dan minyak, serta pengaturan dan teknik *spay gun* yang salah. Tanda-tanda *pinholing* adalah lobang-lobang kecil dipermukaan dan seperti udara dan air yang terperangkap..

15. *Runs Or Sags*

    Yaitu suatu kondisi yang disebabkan karena *spray gun* terlalu dekat dengan permukaan dan kurang cepat dalam *spraying* menyebabkan penimbunan, suhu terlalu rendah di tempat pengecatan serta terlalu banyak menggunakan *thinner*. Tanda-tanda *runs or sags* adalah warna luntur dan terlalu banyak bahan cat di sekitar tempat tersebut.

16. *Sand Scratch Swelling*

    Yaitu suatu kondisi yang disebabkan karena tidak menggunakan *sealer* yang mengakibatkan *solvent top coat* menembus dan membengkakan *enzyme*, pembersihan permukaan yang kurang sempurna serta penggunaan *thinner* ataupun *reducer* yang tidak cocok mengakibatkan *undercoat* menyebabkan goresan-goresan dipermukaan. Tanda-tanda *sand scratch swelling* adalah permukaan cat retak-retak dan seperti goresan yang dalam.

17. *Solvent Popping*

    Yaitu suatu kondisi yang disebabkan karena pengecatan terlalu tebal menyebabkan *solvent* terperngkap di *undercoat*, penggunaan *thinner* atau

reducer tidak cocok serta menggunakan cat jenis *water base*. Tanda-tanda *solvent popping* adalah lobang-lobang kecil dipermukaan dan cat berbuih.

18. *Water Marks*

    Yaitu suatu konsidi yang disebabkan karena kesalahan thinner, kesalahan sistem lapisan dan kesalahan takaran. Tanda-tanda *water marks* adalah terdapat bekas cap air dan bentuknya melingkar.

19. *Wet Spot*

    Yaitu suatu kondisi yang disebabkan karena pembersihan yang tidak sempurna, pengeringan yang tidak sempurna atau *undercoat* yang berlebihan, mengamplas basah dengan *solvent* yang terkontaminasi. Tanda-tanda *wet spot* adalah permukaan seperti bekawah dan permukaan berminyak.

20. *Wrinkling*

    Yaitu suatu kondisi yang disebabkan karena pengeringan yang tidak sempurna menyebabkan pengecatan selanjutnya menyebabkan *undercoat* melunak, penggunaan *reducer* ataupun bahan yang tidak sesuai, suhu tempat pengecatan yang tidak sesuai menyebabakan permukaan *enamel* terbentuk tidak rata. Tanda-tanda *wrinkling* adalah permukaan berkerut-kerut dan bentuknya vertikal.

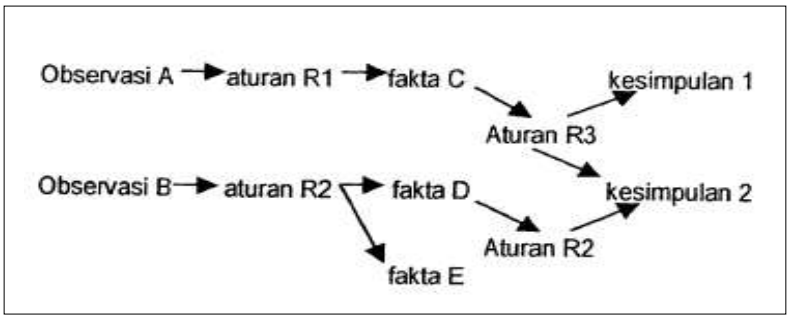
    .

II.5. *Forward Chaining*

Forward chaining adalah teknik pencarian yang dimulai dengan fakta yang diketahui, kemudian mencocokkan 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 pencocokkan, dimulai dari rule teratas. Setiap rule hanya boleh dieksekusi sekali saja. Proses pencocokkan berhenti bila tidak ada lagi rule yang bisa dieksekusi (T.Sutojo; 2011: 171).

Adapun diagram *forward chaining* ditunjukkan pada gambar II.2 sebagai berikut:



Gambar II.2 Diagram *Forward Chaining*

Sumber: (Fetty Tri Anggraeny, I Gede Susrama, Lina Surtika ; SCAN VOL. II NOMOR 1 : 32)

II.6. *PHP*

PHP adalah kependekan dari *PHP Hypertext Preprocessor*, bahasa *interpreter* yang mirip dengan bahasa *C* dan *Perl* yang memiliki kesederhanaan dalam perintah. *PHP* dapat digunakan bersama dengan *HTML* sehingga memudahkan dalam pembangunan aplikasi *web* dengan cepat. *PHP* dapat digunakan untuk meng-*update* basis data dan menciptakan basis data. *Interpreter* adalah sebuah program yang digunakan untuk membaca *file* yang berisi kode program yang akan dijalankan kemudian *interpreter* tersebut akan meminta *CPU* untuk melakukan perintah yang diterimanya (Iswanto; 2007: 2).

II.7. *MySql*

MySql adalah salah satu perangkat lunak sistem manajemen basis data (*database*) *SQL* atau sering disebut dengan DBMS (*Database Management System*). Berbeda dengan basis data konvensional seperti .dat, .dbf, .mdb, *MySql* memiliki kelebihan yaitu bersifat *multithread* dan *multi-user* serta mendukung sistem jaringan. *MySql* didistribusikan secara grafis dibawah lisensi GNU *General Public Licence* (GPL), namun ada juga versi komersial bagi kalangan tertentu yang menginginkannya (A.M Hirin & Virgi ; 2011 : 27-28).

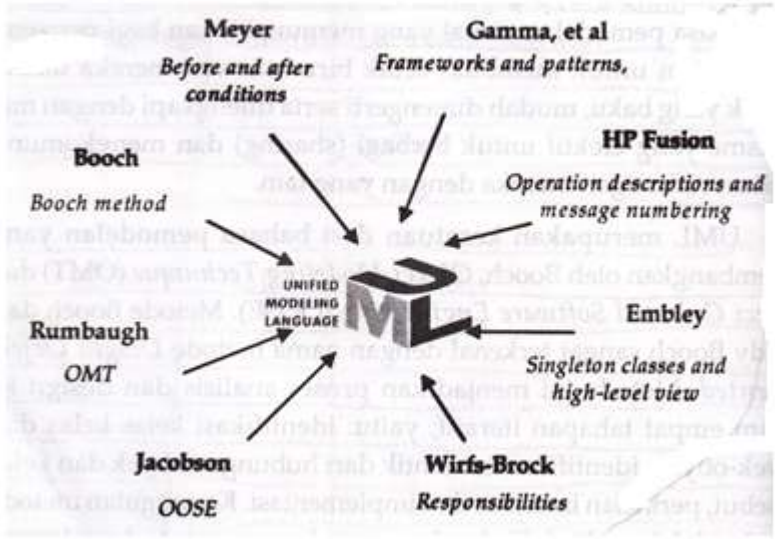
II.8. UML (*Unified Modelling Language*)

UML (*Unified Modelling Language*) adalah salah satu alat bantu yang sangat handal di dunia pengembangan sistem yang berorientasi objek. Hal ini disebabkan karena UML menyediakan bahasa pemodelan visual yang memungkinkan bagi pengembangan sistem untuk membuat cetak biru atas visi mereka dalam bentuk yang baku, mudah dimengerti serta dilengkapi dengan mekanisme yang efektif untuk berbagi (Sharing) dan mengkomunikasikan rancangan dengan lain (Munawar; 2005: 17).

UML merupakan kesatuan bahasa pemodelan yang dikembangkan oleh Booch, *Object Modeling Technique* (OMT) dan *object Oriented Engineering* (OOSE). Metode Booch dari Grady Booch sangat terkenal dengan nama metode *Design Object Oriented.* Metode ini menjadikan proses analisis dan design ke dalam empat tahapan iteratif, yaitu: identifikasi kelas-kelas dan objek-objek, identifikasi semantik dari hubungan objek dan kelas tersebut, perincian interface

dan implementasi. Keunggulan metode Booch adalah pada detil dan kayanya dengan notasi dan elemen. Pemodelan OMT yang dikembangkan oleh Rumbaugh didasarkan pada analisis terstruktur dan pemodelan *entity-relationship*. Tahapan utama dalam metodologi in adalah analisis, desain sistem, desain objek dan impelmentasi. Keunggulan motode ini adalah dalam penotasian yang mendukung semua konsep OO. Metode OOSE dari Jacobson lebih memberi penekanan dan *use case.* OOSE memiliki tiga tahapan yaitu membuat model *requirement* dan analisis, desain dan implementasi dan model pengujian (test Model). Keunggulan metode ini adalah mudah dipelajari karena memiliki notasi yang sederhananamun mencangkup seluruh tahapan dalam rekayasa perangkat lunak.

Dengan UML, metode Booch, OMT dan OOSE digabungkan dengan membuang elemen-elemen yang tidak praktis ditambah dengan elemen-elemen dari metode lain yang lebih efektif dan elemen-elemen baruyang belum ada pada metode terdahulu sehingga UML lebih ekspresif dan seragam daripada metode lainnya. Unsur-unsur yang membentuk UML ditunjukkan dalam Gambar II.3.



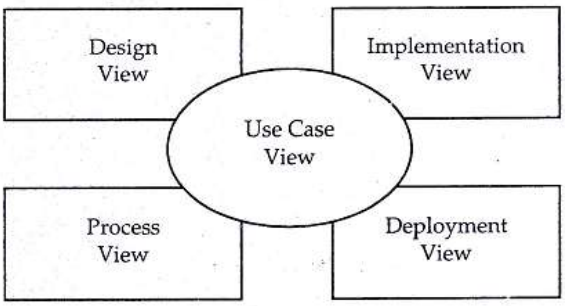
Gambar II.3 Unsur-unsur pembentuk UML

Sumber: (Munawar; 2005: 18)

UML adalah hasil kerja dari konsorsium berbagai organisasi yang berhasil dijadikan sebagai standar baku dalam OOAD (*Object Oriented Analysis dan Design*). UML tidak hanya dominan dalam penotasian di lingkungan OO tetapi juga populer di luar lingkungan OO. Ada tiga karakter penting yang melekat di UML yaitu sketsa, cetak biru dan bahasa pemrograman. Sebagai sebuah sketsa UML bisa berfungsi sebagai sebuah cetak biru kerena sangat lengkap dan detil. Dengan cetak biru ini maka akan bisa diketahui informasi detil tentang coding program (*Forward Engineering*) atau bahkan membaca program dan mengiterpretasikannya kembali ke dalam diagram (*reverse engineering*). *Reverse engineering* sangat berguna pada situasi dimana kode program yang tidak terdokumentasi asli hilang atau bahkan belum dibuat sama sekali. Sebagai bahasa

pemrogaraman, UML dapat menterjemahkan diagaram yang ada di UML menjadi kode program siap untuk dijalankan.

UML dibangun atas model 4+1 *view*. Model ini didasarkan pada fakta bahwa struktur sebuah sistem dideskripsikan dalam *view* dimana salah satu diantaranya *use case view*. *Use case view* ini memegang peran khusus untuk mengintegrasikan *content* ke *view* yang lain. Model 4+1 *view* ditunjukkan pada gambar II.4.



Gambar II.4 Model 4+1 *View*

Sumber: (Munawar; 2005: 20)

Kelima *view* tersebut tidak berhubungan dengan diagram yang dideskripsikan di UML. Setiap *view* berhubungan dengan perspektif tertentu dimana sistem akan diuji. *View* yang berbeda akan menekankan pada aspek yang berbeda dari sistem yang mewakili tentang sistem bisa dibentuk dengan menggabungkan informasi-informasi yang ada pada kelima *view* tersebut.

Use case view mendefinisikan perilaku eksternal sistem. Hal ini menjadi daya tari bagi *end user,* analis dan tester. Pandangan ini mendefinisikan kebutuhan sistem karena mengandung semua *view* yang lain yang

mendeskripsikan aspek-aspek tertentu dari peran dan sering dikatakan yang mendrive proses pengembangan perangkat lunak.

Design view mendeskripsikan struktur logika yang mendukung fungsi-fungsi yang dibutuhkan di *use case*. *Design view* ini berisi definisi komponen program, class-class utama bersama-sama dengan spesifikasi data, perilkau dan interaksinya. Informasi yang terkandung di *view* ini menjadi perjatian para programer karena menjelaskan secara detil bagaimana fungsionalitas sistem akan diimplementasikan.

Implemantasi *view* menjelaskan komponen-komponen fisi dari sistem yang akan dibangun. Hal ini berbeda dengan komponen logic yang dideskripsikan pada *design view.* Termasuk disini diantaranya *file exe*, *library* dan *database.* Informasi yang ada di *view* dan integrasi sistem.

Proses *view* berhubungan dengan hal-hal yang berkaitan dengan *concurrency* do dalam sistem. Sedangkan *deployment view* menjelaskan bagaimana komponen-komponen fisik didistribusikan ke lingkungna fisik seperti jaringan komputer dimana sistem akan dijalankan. Kedua *view* ini menunjukkan kebutuhan non fungsional dari sistem seperti toleransi kesalahan dan hal-hal yang berhubungan dengan kinerja (Munawar; 2005: 17-21).

II.8.1. Use Case Diagram

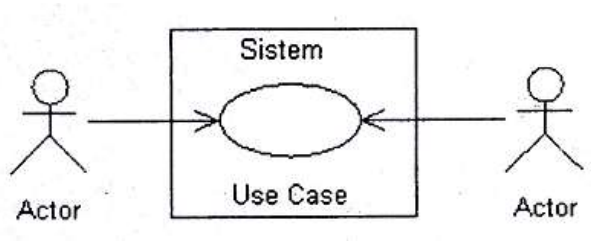
Use case adalah deskripsi fungsi dari sebuah sistem dari perspektif pengguna. *Use case* bekerja dengan cara deskripsikan tipikal interaksi antara *user* (pengguna) sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai. Urutan langkah-langkah yang menerangkan

antara pengguna dan sistem disebut *scenario.* Setiap *scenario* mendeskripsikan urutan kejadian. Setiap urutan diinisialisasi oleh orang, sistem yang lain, perangkat keras atau urutan waktu. Dengan demikian secara singkat bisa dikatakan *use case* adalah serangkaian *scenario* yang digabungkan bersama-sama oleh tujuan umum pengguna.

Dalam pembicaraan tentang *use case,* pengguna biasanya disebut dengan *actor. Actor* adalah sebuah peran yang bisa dimainkan oleh pengguna dalam interaksinya dengan sistem.

Model *use case* adalah bagai dari model *requirement.* Termasuk disini adalah problem domain object dan penjelasan tentang *user interface*. *Use case* memberikan spesifikasi fungsi-fungsi yang ditawarkan oleh sistem dari *perspectif user.*

Notasi *use case* menunjukkan 3 aspek dari sistem yaitu *actor use case* dan *system/sub system boundary. Actor* mewakili peran orang, *system* yang lain atau alat ketika berkomunikasi dengan *use case.* Ilustrasi *actor, usecase* dan *system* ditunjukkan pada gambar II.5.



Gambar II.5 *Use case* Model

Sumber: (Munawar; 2005: 64)

Untuk mengidentifikasi *actor,* harus ditentukan pembagian tenaga kerja dan tugas-tugas yang berkaitan dengan peran pada konteks targer sistem. *Actor* adalah *abstraction* dari orang dan sistem yang lain yang mengaktifkan fungsi dari target sistem. Orang atau sistem bisa muncul dalam beberapa peran. Perlu dicatat bahwa *actor* berinteraksi dengan *use case*, tetapi tidak memiliki kontrol atas *use case.*

Use case adalah abstraksi dari interaksi antara sistem dan *actor*. Oleh karena itu sangat penting untuk memilih abstraksi yang cocok. *Use case* dibuat berdasarkan keperluan *actor. Use case* harus merupakan 'apa' yang dikerjakan *software* aplikasi, bukan 'bagaimana' *software* aplikasi mengerjakannya. Setiap *use case* harus diberi nama yang menyatakan apa hal yang dicapai dari hasil interaaksinya dengan *actor*. Namun *use case* boleh terdiri dari beberapa kata dan tidak boleh ada dua *use case* yang memiliki nama yang sama (Munawar; 2005: 63-66).

II.8.2. Class Diagram

Class adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class* menggambarkan keadaan (*atribut*/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class, package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain.

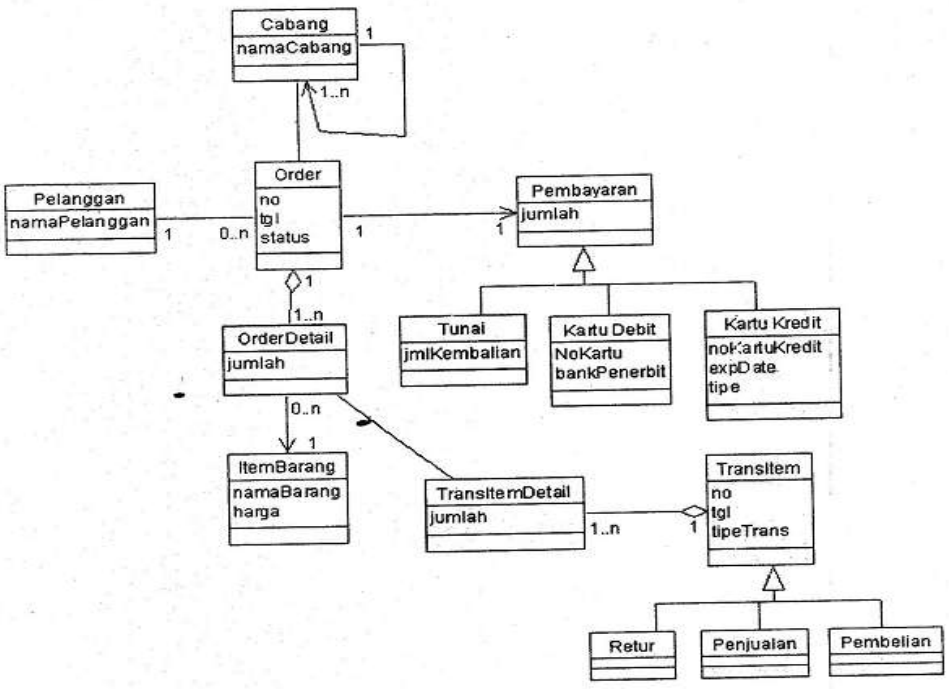
Class memiliki tiga area pokok:

1. Nama kelas
2. Atribut
3. Metode

Atribut dan metode dapat memiliki salah satu sifat berikut :

1. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan.
2. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan.
3. *Public*, dapat dipanggil oleh siapa saja.

Class dapat merupakan implementasi dari sebuah *interface*, yaitu *class* abstrak yang hanya memiliki metode. *Interface* tidak dapat langsung diinstansiasikan, tetapi harus diimplementasikan dahulu menjadi sebuah *class*. Contoh diagram *class* dapat dilihat pada gambar II.6 dibawah ini:



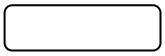
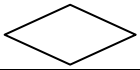
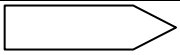
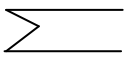
Gambar II.6 *Class* Diagram

Sumber: (Munawar; 2005: 220)

II.8.3. Activity Diagram

Activity Diagram adalah teknik untuk mendiskripsikan logika prosedural, proses bisnis dan aliran kerja dalam banyak kasus. *Activity Diagram* mempunyai peran seperti halnya *flowchart*, akan tetapi perbedaannya dengan *flowchart* adalah *activity* diagram bisa mendukung perilaku paralel sedangkan flowchart tidak bisa. Adapun simbol *activity diagram* dapat dilihat pada table II.2.

Tabel II.2. Simbol *Activity Diagram*

Notasi	Keterangan
	Titik Awal
	Titik Akhir
	*Activity*
	Pilihan untuk pengambilan keputusan
	*Fork* digunakan untuk menunjukkan kegiatan yang dilakukan secara paralel atau untuk menggabungkan dua kegiatan paralel menjadi satu
	*Rake* menunjukkan adanya dekomposisi
	Tanda waktu
	Tanda pengiriman
	Tanda penerimaan
	Aliran Akhir (*Flow Final*)

Sumber: (Munawar; 2005: 110)

II.8.4. Sequence Diagram

Sequence diagram digunakan untuk menggambarkan perilaku pada sebuah sekenario. Diagram ini menunjukkan sejumlah contoh objek dan pesan yang diletakkan diantaran objek-objek ini di dalam *use case.*

Komponen utama *sequence diagram* terdiri atas objek yang ditulisakan dengan kotak segiempat bernama. *Messege* diwakili oleh garis dengan tanda panah dan waktu yang ditunjukkan dengan *progress vertical.*

1. Objek /*participant*

Objek diletakkan di dekat bagian atas diagram dengan urutan dari kiri ke kanan. Mereka diatur dalam urutan guna menyederhanakan diagram. Setiap *participant* dihubungkan dengan garis titik-titik yang disebut *lifeline.* Sepanjang *lifeline* ada kotak yang disebut *activation. Activation* mewakili sebuah eksekusi operasi dari *participant.* Panjang kotak ini berbanding lurus dengan durasi *activation.* Bentuk *participant* dapat dilihat pada gambar II.7.

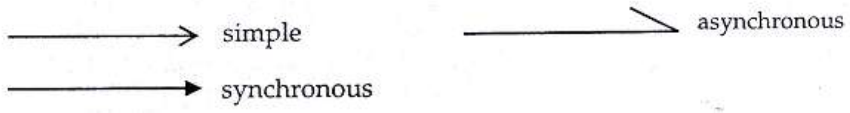
Gambar II.7 Bentuk *Participant*

Sumber: (Munawar; 2005: 88)

2. *Messege*

Sebuah *messsage* bergerak dari satu *participant* ke *participant* yang lain dan dari satu *lifeline* ke *lifeline* yang lain. Sebuah *participant* bisa mengirim sebuah *message* kepada dirinya sendiri.

Sebuah *message* bisa jadi *simple, synchronous* atau *asynchoronous. Message* yang *simple* adalah sebuah perpindahan (transfer), contoh dari satu *participant* ke *participant* yang lainnya. Jika sebuah *participant* mengirimkan sebuah *messagae* tersebut akan ditunggu sebelum diproses dengan urusannya. Namun jika *message asynchoronous* yang dikirimkan, maka jawabannya atas *message* tersebut tidak perlu ditunggu. Simbol *message* pada *squnence diagram* dapat dilihat pada gambar II.8.



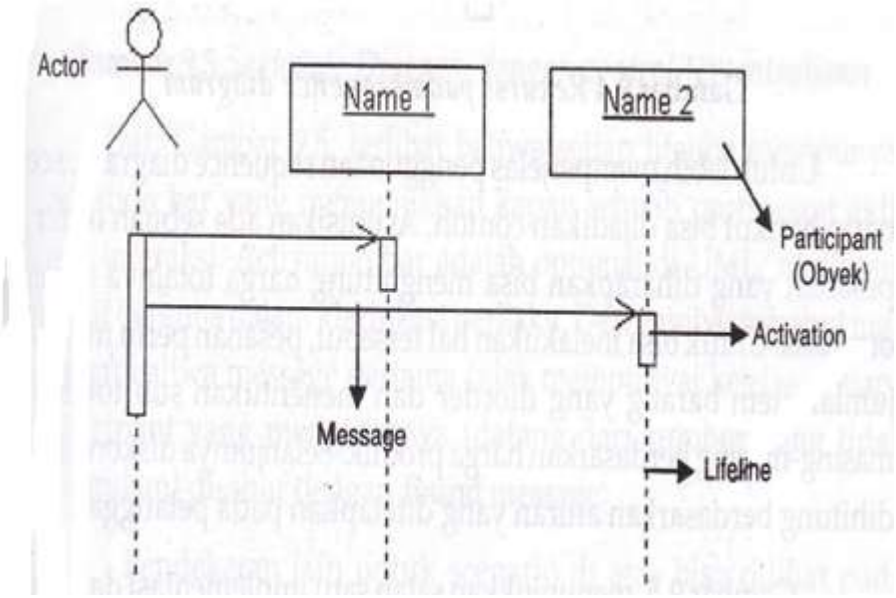
Gambar II.8 Simbol-simbol *Messege*

Sumber: (Munawar; 2005: 88)

3. Time

Time adalah diagram yang mewakili waktu pada arah vertikal. Waktu dimulai dari ata ke bawah. *Message* yang lebih dekat dari atas akan dijalankan terlebih dahulu dibanding *message* yang lebih dekat ke bawah.

Terdapat dua dimensi pada *Sequence diagram* yaitu dimensi dari kiri ke kanan menunjukkan tata letak *participant* dan dimensi dari atas ke bawah menunjukkan lintasan waktu. Simbol-simbol yang ada pada *Sequence diagram* ditunjukkan pada gambar II.9.



Gambar II.9 *Sequence Diagaram*

Sumber: (Munawar; 2005: 89)

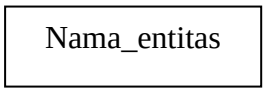
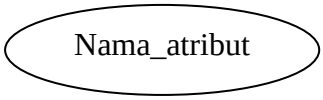
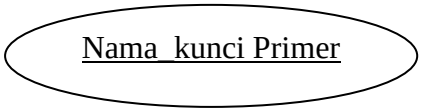
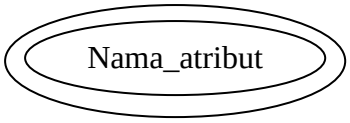
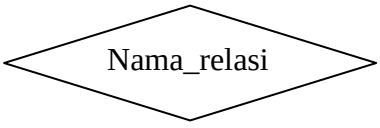
II.9. ERD (*Entity Relationship Diagram*)

ERD (*Entity Relationship Diagram*) dikembangkan berdasarkan teori himpunan dalam bidang matematika. ERD (*Entity Relationship Diagram*) digunakan untuk pemodelan basis data relasional. Sehingga jika penyimpanan basis data menggunakan OODBMS maka perancangan basis data tidak perlu menggunakan ERD (*Entity Relationship Diagram*) (Rosa A.S, M. Shalahuddin; 2011: 49).

II.9.1. Simbol-simbol ERD (*Entity Relationship Diagram*)

Adapaun simbol-simbol ERD (*Entity Relationship Diagram*) ditunjukkan pada tabel II.3.

Tabel II.3. Simbol-simbol ERD (*Entity Relationship Diagram*)

Simbol	Deskripsi
Entitas/ *entity* 	Entitas merupakan data inti yang akan disimpan; bakal tabel pada basis data
Atribut 	*Field* atau kolom data yang perlu 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
Atribut multinal/ *multivalue* 	*Field* atau kolom data yang butuh disimpan dalam suatu entitas yang dapat memiliki nilai lebih dari satu
Relasi 	Relasi yang menghungkan antarentitas; relasi biasanya diawali dengan kata kerja
Asosiasi/ *Association*	Penghubung antar relasi dan entitas dimana di kedua ujungnya memiliki *multiplicity*

1 0..*	kemungkinan jumlah pemakaian

Sumber: (Rossa A.S-M. Shalahuddin; 2011: 50-51)

II.10. Normalisasi

Normalisasi adalah teknik perancangan yang banyak digunakan sebagai pemandu dalam merancang basisdata relasional. Pada dasarnya, normalisasi adalah proses dua langkah yang meletakkan data dalam bentuk tabulasi dengan menghilangkan kelompok berulang lalu menghilangkan data yang terduplikasi dari tabel relasional (Janner Simarmata, Iman Paryudi; 2006: 77).

Tahapan normalisasi terdiri dari beberapa bentuk yaitu sebagai berikut:

1. Bentuk Normal Pertama (1NF/*First Normal Form*)

Bentuk normal pertama adalah suatu bentuk relasi yang atribut bernilai banyaknya (*multivalues attribute*) telah dihilangkan sehingga kita akan menjumpai nilai tunggal (mungkin saja nilai *null*) pada perpotongan setiap baris dan kolom pada tabel (Nugroho; 2009: 44).

2. Bentuk Normal Kedua (2NF/*Second Normal Form*)

Defenisi bentuk normal kedua menyatakan bahwa tabel dengan kunci utama gabungan hanya dapat berada pada 1NF, tetapi tidak pada 2NF. Sebuah tabel relasional berada pada bentuk normal kedua jika dia berada pada 1NF dan setiap kolom bukan kunci yang sepenuhnya tergantung pada kunci utama. Ini berarti bahwa setiap kolom bukan kunci harus tergantung pada seluruh kolom yang membentuk kunci utama.

3. Bentuk Normal Ketiga (3NF/*Third Normal Form*)

Bentuk normal ketiga mengharuskan semua kolom pada tabel relasional tergantung pada kunci utama. Secara defenisi, sebuah tabel berada pada bentuk normal ketiga (3NF) jika tabel sudah berada pada 2NF dan setiap kolom yang bukan kunci tidak tergantung secara transitif pada kunci utamanya. Dengan kata lain, semua atribut bukan kunci tergantung secara fungsional hanya pada kunci utama.

4. Bentuk Normal B*oyce Code* (BCNF/*Boyce Code Normal Form*)

Bentuk normal B*oyce Code* (BCNF) adalah versi 3NF yang lebih teliti dan berhubungan dengan tabel relasional yang mempunyai (a) banyak kunci kandidat, (b) kunci kandidat gabungan, dan (c) kunci kandidat yang saling tumpang tindih.

BCNF didasarkan pada konsep penentu. Sebuah kolom penentu adalah kolom di mana kolom-kolom lain sepenuhnya tergantung secara fungsional. Sebuah tabel relasional berapa pada BCNF jika dan hanya jika setiap penentu adalah kunci kandidat (Janner Simarmata, Iman Paryudi; 2006: 81-85).