BAB II

TINJAUAN PUSTAKA

II.1. Sistem Pakar

II.1.1. Pengertian Sistem Pakar

Sistem Pakar (*Expert System*) merupakan suatu sistem yang menggunakan pengetahuan manusia dalam komputer untuk memecahkan masalah yang biasanya dikerjakan oleh seorang pakar, misalnya : Dokter, *Lawyer*, Analis Keuangan, *Tax Advisor*. Sistem pakar dapat mendorong perhatian besar diantara ahli komputer dan spesialis informasi untuk mengembangkan sistem yang membantu manajer dan non manajer memecahkan masalah. Sistem pakar adalah sistem yang berusaha mengadopsi pengetahuan manusia ke komputer yang dirancang untuk memodelkan kemampuan untuk menyelesaikan masalah seperti layaknya seorang pakar (Daniel, Virginia, 2010:27).

Definisi sistem pakar yang lain yaitu suatu sistem yang dirancang untuk dapat menirukan keahlian seorang pakar dalam menjawab pertanyaan dan memecahkan suatu masalah (Sutojo, et al., 2011:13). Sistem pakar yang merupakan bagian dari kecerdasan buatan dapat dilihat sebagai upaya untuk aspek model pemikiran manusia pada komputer. Hal ini juga kadang-kadang didefinisikan sebagai sebuah usaha komputer untuk memecahkan setiap masalah manusia yang dapat selesaikan dengan lebih cepat.

Dari pengertian-pengertian sistem pakar tersebut diatas dapat ditarik sebuah kesimpulan bahwa sistem pakar adalah sistem yang didesain dan

diimplementasikan dengan bantuan bahasa pemrograman tertentu untuk dapat menyelesaikan masalah seperti yang dilakukan oleh para ahli (pakar). Diharapkan dengan sistem ini, orang awam dapat menyelesaikan masalah tertentu baik yang mudah ataupun rumit sekalipun tanpa bantuan para ahli dalam bidang tersebut.

Konsep dasar sistem pakar mengandung keahlian, ahli, pengalihan keahlian, inferensi, aturan, dan kemampuan menjelaskan atau memberikan solsusi dari permasalahan. Keahlian adalah suatu kelebihan penguasaan pengetahuan di bidang tertentu yang diperoleh dari pelatihan, membaca, atau pengalaman. Contoh bentuk pengetahuan yang termasuk keahlian (Luther, Motolalu, 2011:132), yaitu :

1. Fakta-fakta pada lingkup permasalahan tertentu.
2. Teori-teori pada lingkup permasalahan tertentu.
3. Prosedur-prosedur dan aturan-aturan berkenaan dengan lingkup permasalahan tertentu.
4. Strategi-strategi global untuk menyelesaikan masalah.
5. *Meta Knowledge* (pengetahuan tentang pengetahuan).

II.1.2. Pemakai Sistem Pakar

Dalam sistem pakar pemakai merupakan aktor utama terbentuknya sistem pakar tersebut. Sebagai contoh ketika ingin membangun sistem pakar yang mendiagnosa kerusakan laptop, maka anda butuh seorang pakar. Hal ini tidak lepas dari pengertian sistem pakar tersebut, yang menyatakan bahwa sistem pakar merupakan program yang dapat menggantikan keberadaan seorang pakar. Untuk itu sistem pakar dapat digunakan oleh (Kusrini, 2006:14) :

1. Orang awam yang bukan pakar untuk meningkatkan kemampuan mereka dalam memecahkan masalah.
2. Pakar sebagai asisten yang berpengetahuan.
3. Memperbanyak atau menyebarkan sumber pengetahuan yang semakin langka.

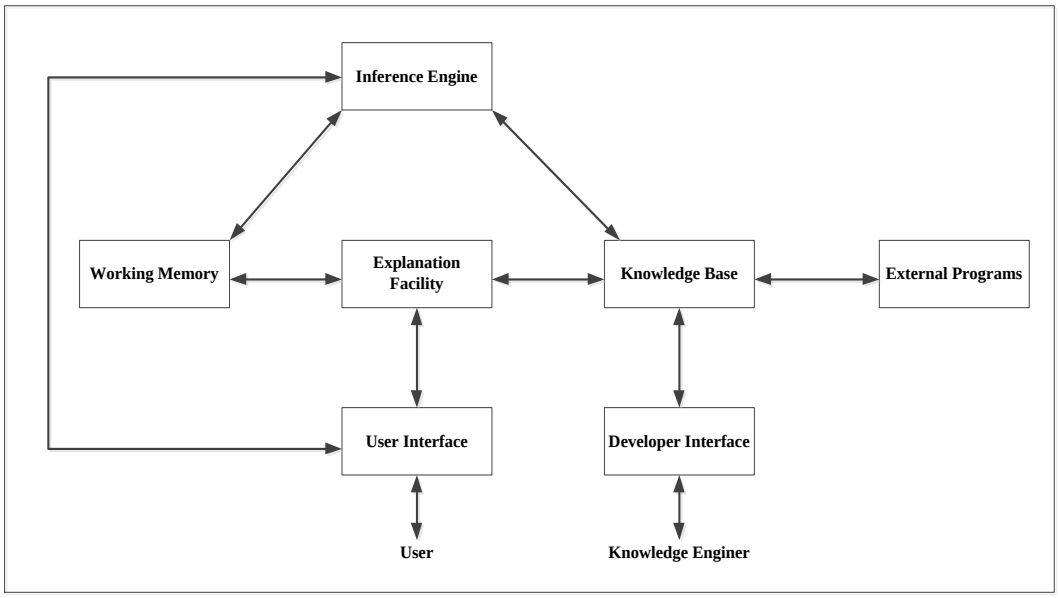
II.1.3. Ciri-Ciri Sistem Pakar

Menurut Sutojo, et al. (2011:162), ciri-ciri sistem pakar adalah 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.

II.1.4. Arsitektur Sistem Pakar

Sistem pakar memiliki beberapa komponen utama, yaitu antarmuka pengguna (*user interface*), basis data sistem pakar, fasilitas akuisisi pengetahuan, dam mekanisme inferensi.



Gambar II.1. Arsitektur Sistem Pakar

(Sumber : Handojo, Irawan, 2004:34)

Adapun keterangan dari gambar tersebut ada sebagai berikut :

1. *Knowledge base* adalah representasi pengetahuan dari seorang atau beberapa pakar yang diperlukan untuk memahami, memformulasikan dan memecahkan masalah. Dalam hal ini digunakan untuk memecahkan masalah-masalah yang terjadi pada sistem. *Knowledge base* ini terdiri dari dua elemen dasar, yaitu fakta dan *rule*.
2. *Inference engine* merupakan otak dari sistem pakar yang mengandung mekanisme fungsi berpikir dan pola-pola penalaran sistem yang digunakan oleh seorang pakar. Mekanisme ini yang menganalisis suatu masalah tertentu dan kemudian mencari solusi atau kesimpulan yang terbaik.
3. *Working Memory* merupakan tempat penyimpanan fakta-fakta yang diketahui dari hasil menjawab pertanyaan.

4. *User/developer interface*. Semua *software* pengembangan sistem pakar memberikan *interface* yang berbeda bagi *user* dan *developer*. *User* akan berhadapan dengan tampilan yang sederhana dan mudah sedangkan *developer* akan berhadapan dengan editor dan *source code* waktu mengembangkan program.
5. *Explanation facility* memberikan penjelasan saat mana *user* mengetahui apakah sistem yang diberikan sebuah solusi.
6. *External programs*. Berbagai program seperti *database, spreadsheets, algorithms*, dan lainnya yang berfungsi untuk mendukung sistem.

II.1.5. Permasalahan Sistem Pakar

Ada beberapa permasalahan yang menjadi area luas aplikasi sistem pakar (Anita, Arhami, 2006:232-233), antara lain :

1. Interpretasi. Pengambilan keputusan dari hasil observasi, termasuk diantaranya: pengawasan, pengenalan ucapan, analisis citra, interpretasi sinyal, dan beberapa analisis kecerdasan.
2. Prediksi. Termasuk diantaranya: peramalan, prediksi demografis, peramalan ekonomi, prediksi lalu lintas, estimasi hasil, militer, pemasaran, atau peramalan keuangan.
3. Dianosis. Termasuk diantaranya: medis, elektronis, mekanis, dan diagnosis perangkat lunak.
4. Perancangan. Termasuk diantaranya: layout sirkuit dan perancangan bangunan.

5. Perencanaan. Termasuk diantaranya: perencanaan keuangan, komunikasi, militer, pengembangan produk, *routing* dan manajemen proyek.
6. *Monitoring*. Misalnya: *Computer-Aided Monitoring System*.
7. *Debugging*, memberikan resep obat terhadap suatu kegagalan.
8. Perbaikan.
9. Instruksi. Melakukan instruksi untuk diagnosis, *debugging*, dan perbaikan kerja.
10. Kontrol. Melakukan kotrol terhadap interpretasi-interpretasi prediksi, perbaikan, dan *monitoring* kelakuan sistem.

II.1.6. Kelebihan Sistem Pakar

Sistem pakar merupakan suatu perangkat lunak atau suatu program komputer yang ditujukan sebagai penyedia nasehat dan saran bantu dalam memecahkan masalah di bidang-bidang spesialisasi tertentu seperti sains, perekayasaan, matematika, kedokteran, pendidikan, dan lain-lain. Menurut Kusrini (2006:15), ada beberapa kelebihan pemakaian sistem pakar, antara lain :

1. Membuat orang yang awam dapat bekerja seperti layaknya seorang pakar.
2. Dapat bekerja dengan informasi yang tidak lengkap atau tidak pasti.
3. Meningkatkan *output* dan produktivitas.
4. Meningkatkan kualitas.
5. Sistem pakar menyediakan nasehat yang konsisten dan dapat mengurangi tingkat kesalahan.
6. Membuat peralatan yang kompleks lebih mudah dioperasikan karena sistem pakar dapat melatih pekerja yang tidak berpengalaman.

7. Handal.
8. Sistem pakar tidak lelah atau bosan.
9. Memiliki kemampuan untuk memecahkan masalah yang kompleks.
10. Memungkinkan pemindahan pengetahuan ke lokasi yang jauh serta memperluas jangkauan seorang pakar, dapat diperoleh dan diapakai di mana saja.

II.1.7. Kelemahan Sistem Pakar

Sistem pakar dirancang dengan mengikuti aturan-aturan yang telah ditentukan sebelumnya. Dengan begitu sistem pakar mutlak memberikan hasil seperti apa yang telah ditentukan juga. Hal ini menegaskan bahwa sistem pakar tidak hanya memiliki kelebihan, tetapi juga memiliki kelemahan. Beberapa kelemahan sistem pakar (Sutojo, er al., 2011:161), antara lain :

1. Biaya yang sangat mahal untuk membuat dan memeliharanya.
2. Sulit dikembangkan karena keterbatasan keahlian dan ketersediaan pakar.
3. Sistem pakar tidak 100% bernilai benar.

II.2. *Certainty Factor*

Dalam menghadapi suatu permasalahan sering ditemukan jawaban yang tidak memiliki kepastian penuh. Ketidakpastian ini dapat berupa probabilitas yang tergantung dari hasil suatu kejadian. Hasil yang tidak pasti disebabkan oleh kedua faktor, yaitu aturan yang tidak pasti dan jawaban pengguna yang tidak pasti atas suatu pernyataan yang diajukan oleh sistem. Hal ini sangat mudah dilihat pada sistem diagnosis kerusakan monitor laptop, dimana pakar tidak dapat

mendefinisikan hubungan antara gejala dengan penyebabnya secara pasti, sementara kerusakan monitor tidak dapat ditentukan dengan pasti pula. Pada akhirnya akan ditemukan banyak kemungkinan diagnosis.

Sistem pakar harus mampu bekerja dalam ketidakpastian dalam memberikan suatu solusi dari permasalahan. Sejumlah teori telah ditemukan untuk menyelesaikan ketidakpastian, termasuk diantaranya probabilitas klasik, Probabilitas *Bayes*, Teori *Hartley* berdasarkan himpunan klasik, Teori *Shannon* berdasarkan probabilitas, Teori *Depmster-Shafer*, Teori *Fuzzy Zadeh*, dan faktor kepastian (*certainty factor*).

Certainty factor atau faktor kepastian menyatakan kepercayaan dalam sebuah kejadian (atau fakta atau hipotesis) berdasarkan bukti atau penilaian pakar (Daniel, Virginia, 2010:27).). *Certainty factor* merupakan nilai parameter klinis yang diberikan MYCIN untuk menunjukkan besarnya kepercayaan. Faktor kepastian memberikan seorang pakar untuk menyatakan kepercayaan tanpa menyatakan nilai kepercayaan. *Certainty factor* memperkenalkan konsep keyakinan dan ketidakyakinan yang kemudian diformulakan dalam rumusan dasar sebagai berikut :

$$CF[P,E] = MB[P,E] - MD[P,E]$$

Keterangan :

CF : *Certainty Factor*

MB : *Measure of Belief*

MD : *Measure of Disbelief*

P : *Probability*

E : *Evidence* (Peristiwa/Fakta)

Gabungan kepercayaan (*belief*) dan ketidakpercayaan (*unbelief*) dalam bidang yang tunggal memiliki dua kegunaan, yaitu pertama faktor kepastian digunakan untuk tingkatan hipotesis di dalam urutan kepentingan. Faktor kepastian (CF) menunjukkan jaringan kepercayaan berdasarkan beberapa fakta atau gejala dalam bidang kedokteran CF positif bermakna fakta mendukung hipotesis karena MB > MD. CF = 1 mengandung arti bahwa fakta secara definisi membuktikan suatu hipotesis. CF = 0 berarti slah satu dari dua kemungkinan, yaitu pertama CF = MB – MD = 0 keduanya MB dan MD adalah nol yang berarti tidak ada fakta. Kemungkinan kedua adalah bahwa MD = MB dan keduanya tidak sama dengan nol yang berarti bahwa kepercayaan dihapus atau ditiadakan oleh ketidakpercayaan.

Faktor kepastian memberikan seorang pakar untuk menyatakan kepercayaan tanpa menyatakan nilai kepercayaan. Dengan demikian rumusannya dapat diyatakan : CF(H,E) + CF(H',E) = 0. Yang berarti bahwa jika fakta memperkuat hipotesis yang diyatakan oleh nilai CF(H | E), di mana penegasan dari negasi hipotesisnya bukan 1-CF(H | E) seperti yang diyatakan oleh teori probabilitas, yaitu:

$$CF(H, E) + CF(H', E) \neq 1$$

Fakta CF(H, E) + CF(H', E) = 0 berarti bahwa *evidence* mendukung suatu hipotesis dan mengurangi dukungan terhadap negasi dari hipotesis dengan jumlah yang sama, sehingga jumlahnya selalu nol.

Rumus dasar untuk CH dari kaidah : IF E THEN H

Diberikan dengan rumus di bawah ini: CF(H,e) = CF(E,e) CF(H,E)

Di mana:

- CF(E, e) : Adalah faktor kepastian dari fakta E membuat *antecedent* dari kaidah berdasarkan pada ketidakpastian fakta e.
- CF(H, E): Adalah Faktor kepastian dalam hipotasa dengan asumsi bahwa fakta diketahui dengan pasti, bila CF(E, e) = 1.
- CF(H, e): Adalah faktor kepastian hipotesis yang didasarkan pada ketidakpastian fakta e.

Sehingga jika semua fakta dalam *antecedent* diketahui dengan pasti, rumus untuk faktor kepastian adalah:

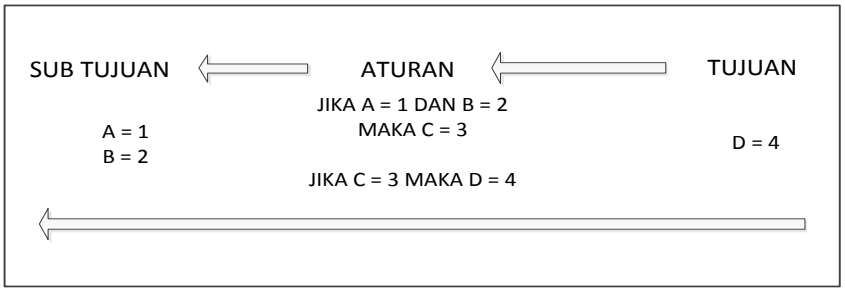
CF(H,e) = CF(H.E) karena CF(E, e) = 1.

Dalam diagnosis suatu penyakit, hubungan antara gejala dan hipotesis sering tidak pasti. Sangat dimungkinkan beberapa aturan menghasilkan satu hipotesis dan satu hipotesis menjadi *evidence* bagi aturan yang lain.

II.3. *Backward Chaining*

Backward Chaining adalah pendekatan yang dimotori oleh tujuan (*goal driven*). Dalam pendekatan ini pelacakan dimulai dari tujuan, selanjutnya dicari aturan yang memiliki tujuan tersebut untuk kesimpulannya. Selanjutnya proses pelacakan menggunakan aturan tersebut sebagai tujuan baru dan mencari aturan lain dengan tujuan baru sebagai kesimpulannya. Proses berlanjut sampai semua kemungkinan ditemukan. Menurut Sutojo, et al. (2011:178), *backward chaining* adalah metode inferensi yang berkerja mundur kearah kondisi awal. Proses diawali dari tujuan (bagian THEN), yang kemudian pencarian mulai dijalankan

untuk mencocokkan apakah fakta-fakta yang ada cocok dengan premis-premis di bagian IF.



Gambar II.2. Inferensi *Backward Chaining*

(Sumber : Kusrini, 2006:37)

II.4. *Hypertext Preprocessor* (PHP)

Hypertext preprocessor atau PHP adalah adalah salah satu jenis bahasa pemograman *web* yang *open source*, sehingga dapat digunakan oleh siapa saja secara cuma-cuma (Komang, 2009:12). PHP juga merupakan bahasa *server-side scripting* yang bisa menyatu dengan tag-tag HTML. *Server-side scripting* adalah sintaks dan perintah-perintah yang dijalankan pada *server* dan disertakan pada dokumen HTML. PHP berfungsi sebagai bahasa pemrograman yang menjalankan suatu perintah tertentu. PHP mempu mengelolah data pada berbagai *platform database*, namun yang paling ideal dan banyak digunakan adalah menggunakan *database MySQL*. PHP+*MySQL* menjadi standar bagi pembuatan *web* dinamis saat ini, hal ini dikarenakan keduanya *opensource*, sehingga bisa digunakan siapa saja dengan bebas.

Dalam penulisan *script* PHP, terdapat *tag* pembukan dan *tag* penutup. *Tag* ini merupakan sebagai identitas PHP itu sendiri. Sehingga ketika PHP digabungkan dengan *scripting* yang lain, seperti HTML dan *Javascript* maka *tag*

pembukan dan *tag* penutup inilah yang akan memisahkan PHP dengan bahasa pemograman yang lainnya. Selain *tag* tersebut, PHP juga mempunyai variabel. Variabel dalam PHP tidak seperti variabel dalam bahasa pemograman lain. Variabel PHP tidak membutuhkan deklarasi sebelum digunakan (Muklis, Nugroho, 2009:314). Dengan begitu nilai dapat dimasukkan kapanpun untuk digunakan. Penulisan variabel PHP diawali dengan simbol ”**$**”.

Pemograman PHP tidak terlepas dari *control flow*. Hal ini dikarenakan PHP mampu bermain dengan logika. Pada Tabel II.1. akan menjelaskan jenis-jenis *control flow* pada PHP.

Tabel II.1. Jenis-jenis *Control Flow* Pada PHP

Statement Type	***Keywords***
Looping	*While, do-while, for.*
Decision making	*If-else, switch-case*
Exception handling	*Try-catch, finally, throw*
Branching	*Break, continue, label, return.*

(Sumber : Muklis, Nugroho, 2009:314)

II.5. *MySQL*

MySQL adalah *database server* relasional yang gratis di bawah lisensi GNU (*General Public License*). Dengan sifatnya *Open Source*, memungkinkan juga *user* untuk melakukan modifikasi pada *source code*-nya untuk memenuhi kebutuhan spesifik. *MySQL* merupakan *database server multi-user* dan *multi-threaded* yang tangguh (*robust*). Dengan memiliki banyak fitur *MySQL* bisa bersaing dengan *database* komersil lainnya. *MySQL* dikembangkan, disebarluaskan, dan didukung oleh *MySQL* AB yang merupakan perusahaan komersial yang didirikan oleh para pengembang *MySQL* (Wahana, 2010:26).

MySQL merupakan sebuah perangkat lunak sistem manajemen basis data *SQL* (*database management system*) atau DBMS yang memiliki konsep *multithread* dan *multi-user* serta menggunakan perintah SQL (*Structured Query Language*). *MySQL* merupakan sebuah program *database server* yang mampu menerima dan mengirimkan datanya dengan sangat cepat ke dalam sebuah sistem aplikasi (*Client*).

II.5.1. *Internal* dan *Protabilitas*

Salah satu fitur yang terdapat pada *MySQL* adalah fitur *internal* dan *portabilitas*. Menurut Wahana Komputer (2010:27-28), ada beberapa fitur *internal* dan *portabilitas* sebagai salah satu fitur utama pada *MySQL*, antara lain :

1. *MySQL Server* ditulis dengan bahasa C dan C++.
2. *MySQL Server* diuji secara luas menggunakan *compiler* yang berbeda.
3. *MySQL Server* mampu bekerja pada banyak *platform* berbeda.
4. *MySQL Server* menggunakan GNU *Automake*, *Autoconf*, dan *Libtool* untuk *portabilitas.*
5. Tersedia API untuk C, C++, *Eifel*, *Java*, *Perl*, PHP, *Python, Ruby*, dan Tcl untuk mengakses *MySQL Server*.
6. *MySQL Server* secara penuh mendukung *multi-threaded* menggunakan *thread* kernel sehingga mudah menggunakan banyak CPU, jika tersedia.
7. *MySQL Server* menyediakan mesin penyimpanan transaksi dan non-transaksi.
8. *Thread MySQL Server* sangat cepat berdasarkan sistem alokasi memori.
9. Penggabungan tabel *MySQL Server* dapat dilakukan sangat cepat menggunakan optimasi *one sweep multi-join*.

II.5.2. Tipe Data *MySQL*

MySQL Server mendukung banyak tipe data yang dapat disimpan pada sebuah kolom. Terdapat tiga kategori tipe data yang didukung oleh *MySQL Server*, yaitu :

1. Tipe Data Numerik

   Data numerik adalah salah satu bentuk data berupa angka, baik berupa bilangan bulat maupun bilangan *real*.

Tabel II.2. Tipe Data Numerik Bilangan Bulat

Tipe Data	*Byte*	Nilai Minimal	Nilai Maksimal
Tinyint	1	-128	127
		0	255
Smallint	2	-32768	32767
		0	65535
Mediumint	3	-8388608	8388607
		0	16777215
Int/Integer	4	-2147483648	21455483648
		0	4294967295
Bigint	8	-9223372036854775808	9223372036854775807
		0	184467440373709551615

(Sumber : Wahana, 2010:31)

Tabel II.3. Tipe Data Numerik Bilangan *Real*

Tipe Data	*Byte*	Keterangan
Float (p)	4 jika0<=p<=24	P mempresentasikan presisi bit.
Float	4	Angka *floating point* kecil
Double	8	Ukuran normal angka *floating point*
Decimal(M,D), *Numeric*(M,D)	Variasi	M adalah jumlah angka digit *decimal* dan D adalah angka dibelakang *decimal*
Bit (M)	(M+7)/8	M adalah banyaknya bit setiap nilai.

(Sumber : Wahana, 2010:31)

2. Tipe Data String

   Tipe data string dapat menyimpan semua data, baik berupa karakter, angka, waktu maupun tanggal. Data dapat pula merupakan kombinasi karakter dan angka.

Tabel II.4. Tipe Data String

Tipe Data	*Byte*	Keterangan
Varchar	255	Tipe *varchar* menyimpan data sebanyak karakter yang diinputkan.
Char	255	Tipe *char* sama dengan *varchar*, hanya saja tempat penyimpanan selalu tetap.
Binary	255	*Binary* mirip dengan *char* hanya yang disimpan adalah nilai biner dari data yang disimpan.
Varbinary	255	*Varbinary* sama dengan *binary*, tetapi keduanya berbeda sebagaimana perbedaaan *char* dengan *varchar*.
Enum	N	Tipe data ini disebut juga dengan tipe data validasi. Pada tipe ini data inputan telah dideklarasikan terlebih dahulu.
Set	N	*Set* memeiliki fungsi yang sama dengan *enum*.

(Sumber : Wahana, 2010:33)

3. Tipe Data Penanggalan dan Waktu

   Dalam menangani data tanggal dan waktu (jam), *MySQL* memiliki tipe data tersendiri. Dengan begitu, masalah penanggalan dan waktu dapat diselesaikan dengan cepat daripada menggunakan tipe data string.

Tabel II.5. Tipe Data Tanggal dan Waktu

Tipe Data	*Byte*	Keterangan
Datetime	8	Bentuk ini merupakan tipe data yang menyimpan dua tipe, yaitu tanggal dan jam.
Date	3	Tipe ini hanya menyimpan data tanggal dengan format “0000-00-00”.
Timestamp	4	Tipe ini ditulis berjajar tanpa ada pembatas. Tipe ini dapat menyimpan tanggal dan jam.
Time	3	Tipe ini hanya dapat menyimpan data jam.
Year	1	Tipe ini hanya menyimpan data tahun saja.

(Sumber : Wahana, 2010:33-34)

II.6. Sistem Basis Data

Basis data adalah sekumpulan file. Definisi umum dari basis data adalah bahwa basis data merupakan kumpulan dari seluruh data berbasis komputer sebuah organisasi/perusahaan. Definisi basis data yang lebih sempit adalah bahwa basis data merupakan kumpulan data yang berada di bawah kendali peranti

perangkat lunak sistem menajemen basis data (Raymond dan George, 2008:158). Menurut Marlinda (2004:1), sistem basis data adalah suatu sistem menyusun dan mengelola *record-record* menggunakan komputer untuk menyimpan dan merekam serta memelihara data operasional lengkap sebuah organisasi/perusahaan sehingga pemakai mampu menyediakan informasi yang optimal yang diperlukan pemakai untuk proses mengambil keputusan.

Sistem basis data terdiri dari beberapa komponen yang saling berhungan satu sama lain. Adapun komponen-komponen dasar pada sistem basis data adalah sebagai berikut (Marlinda, 2004:2) :

1. Data

   Data di dalam sebuah basis data dapat disimpan secara terintegrasi dan dapat pula dipakai secara bersama-sama. Terdapat tiga jenis data, yaitu :

   a. Data Operasional
   b. Data Masukkan
   c. Data Keluaran

2. *Hardware* (Perangkat Keras)

   Terdiri dari semua peralatan komputer yang digunakan untuk pengelolaan sistem basis data yang berupa: peralatan untuk penyimpanan, peralatan masukkan dan keluaran, dan peralatan komunikasi data.

3. *Software* (Perangkat Lunak)

   Berfungsi sebagai perantara antara pemakai dengan data fisik pada basis data. Perangkat lunak dalam basis data berupa *Database Management System* (DBMS) atau program aplikasi prosedur.

4. *User* atau Pemakai

   Pemakai basis data dibagi atas tiga klasifikasi, yaitu :

   a. *Database Administrator* (DBA)

      Database Administartor (DBA) adalah orang atau tim yang bertugas untuk mengelola sistem basis data secara keseluruhan. Adapun tugas dari DBA adalah sebagai berikut :

      1) Mengontrol DBMS dan *software-software*.
      2) Memonitor siapa saja yang mengakses basis data.
      3) Mengatur pemakaian basis data.
      4) Memeriksa keamanan, *integrity, recovery* dan *concurrency*.

   b. *Programmer*

      Programmer adalah orang atau tim yang bertugas membuat program aplikasi, misalnya untuk perbankan, administrasi dan lain-lain.

   c. *End User*

      End User adalah orang yang mengakses basis data melalui terminal dengan menggunakan *query language* atau program aplikasi yang telah dibuat oleh *programmer*.

II.6.1. *Entity Relationship Diagram*

Entity relationship diagram merupakan suatu model untuk menjelaskan hubungan antar data dalam basis data. Menurut Marlinda (2004:17), model *entity relationship* adalah suatu penyajian data dengan menggunakan *entity* dan *relationship*. Diperkenalkan pada tahun 1976 oleh P.P. Chen. Ada beberapa alasan diperlukannya model *entity relationship,* yaitu :

1. Dapat menggambarkan hubungan antar entitas dengan jelas.
2. Dapat menggambarkan batasan jumlah entitas dan partisipasi antar entitas.
3. Mudah dimengerti oleh pemakai.
4. Mudah disajikan oleh perancang basis data.

Model *entity relationship* terbentuk karena didukung oleh beberapa komponen yang saling berhubungan satu sama lain. Komponen-komponen yang terdapat pada model *entity relationship* adalah sebagai berikut :

1. *Entity*

   Entity adalah suatu yang dapat dibedakan dalam dunia nyata dimana informasi yang berkaitan dengannya dikumpulkan. Simbol *entity* adalah persegi panjang.

2. *Relationship*

   Merupakan hubungan yang terjadi antara satu atau lebih *entity*.

3. *Attribute*

   Attribute merupakan karakteristik dari *entity* atau *relationship* yang menyediakan penjelasan detail tentang hal tersebut. Nilai *attribute* adalah suatu data yang aktual.

4. Indikator Tipe

   Indikator tipe ada dua, yakni : indikator tipe *asscociative object* dan indikator tipe supertipe.

5. *Cardinality Rasio*

   Menjelaskan hubungan batasan jumlah keterhubungan satu *entity* dengan *entity* lainnya atau banyaknya *entity* yang bersesuaian dengan *entity* yang lain melalui *relationship*.

6. Derajat *Relationship*

   Derajat *Relationship* menyatakan jumlah entity yang berpatisipasi di dalam suatu *relationship*.

7. *Participantion Contraint*

   Participantion Contraint, menjelaskan apakah keberadaan suatu *entity* tergantung pada hubungannya dengan *entity* yang lain.

8. Representasi dari *Entity Set*

   Entity set dipresentasikan dalam bentuk tabel dan nama yang *unique*. Setiap tabel terdiri dari sejumlah kolom dan diberi nama yang *unique*.

II.6.2. Normalisasi

Normalisasi merupakan proses pengelompokan elemen data menjadi tabel-tabel yang menunjukkan entitas dan relasinya. Teori normalisasi secara umum merupakan satu *set* peraturan yang membenarkan perkara pangkalan data menganai perkumpulan data yang tidak memuaskan dan menentukan hubungan yang boleh ditukar menjadi bentuk yang lebih baik. Untuk menggunakan normalisasi yang baik, pangkalan data mestilah mengetahui maksud data-data. Terdapat enam bentuk normal (Marlinda, 2004:122-123). Namun demikian, dalam pemaparan berikut hanya akan dijelaskan bentuk normal pertama, kedua dan ketiga.

1. Bentuk Tidak Normal (*Unnormalized Form*)

   Bentuk ini merupakan kumpulan data yang akan direkam, tidak ada keharusan untuk mengikuti suatu format tertentu. Dapat saja data tidak lengkap atau terduplikasi. Data tersebut dikumpulkan apa adanya.

2. Bentuk Normal Pertama (1NF)

   Suatu relasi 1NF jika dan hanya jika sifat dan setiap relasi atributnya bersifat atomik. Atomik bermaksud tidak berkepunyaan untuk berada dalam keadaan satu bagian. Ciri-ciri bentuk normal pertama, yaitu :

   a. Setiap data dibentuk dalam *flat file*.
   b. Tidak ada *set* atribut yang berulang atau bernilai ganda.
   c. Tiap *field* hanya satu pengertian.

3. Bentuk Normal Kedua (2NF)

   Bentuk normal kedua mempunyai syarat yaitu bentuk data telah memenuhi kriteria bentuk normal pertama. Atribut bukan kunci haruslah bergantung secara fungsi pada *primary key*. Peraturan ini menentukan kebergantungan sepenuhnya. Beberapa sumber teks menjelaskan sebagai kebergantungan secara fungsi dan transitif.

4. Bentuk Normal Ketiga (3NF)

   Satu hubungan dikatakan dalam bentuk 3NF jika dan hanya jika ia dalam bentuk 2NF dan setiap atribut tanpa kekunci pula bergantungan secara tidak transitif dengan kunci primer.

II.6.3. Kamus Data

Kamus data (data dictionary) mencakup definisi-definisi dari data yang disimpan di dalam basis data dan dikendalikan oleh sistem manajemen basis data (Raymond dan George, 2008:171). Struktur basis data yang dimuat dalam kamus data adalah kumpulan dari seluruh definisi *field*, definisi tabel, relasi tabel dan hal-hal lainnya.

Kamus data dibuat dan digunakan baik pada tahap analisis maupun pada tahap perancangan sistem. Pada tahap analisis kamus data digunakan sebagai alat komunikasi antara sistem analis dengan *user* tentang data yang mengalir pada sistem tersebut serta informasi yang dibutuhkan oleh pemakai sistem. Karena kamus data merupakan suatu *file* yang terpisah dalam suatu basis data, sehingga kamus data menyimpan informasi seperti :

1. Nama setiap *item*/jenis atau kolom data.
2. Struktur data untuk setiap *item.*
3. Program yang menggunakan tiap *item.*
4. Ukuran *item* pada setiap tipe data.

Kamus data berguna khusus bagi perlindungan timbulnya kelebihan data. Tanpa kamus data, pemakai data lain bagian mungkin menyimpan versi identik dari *item* data yang sama pada beberapa lokasi, di mana masing-masing *item* data mempunyai nama yang berbeda.

.

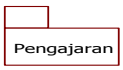
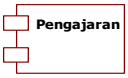
II.7. *Unfied Modeling Languange* (UML)

Unified Modeling Language (UML) adalah bahasa pemodelan untuk sistem atau perangkat lunak yang berparadigma berorientasi objek (Adi Nugroho, 2010:6). Hal ini disebabkan karena UML menyediakan bahasa pemodelan visual yang memungkinkan bagi pengembangan sistem untuk membuat cetak biru atas mereka dalam bentuk yang baku, mudah dimengerti serta dilengkapi dengan mekanisme yang efektif untuk berbagi (*sharing*) dan mengkomunikasikan rancangan mereka dengan yang lain.

UML juga merupakan salah satu alat yang sangat handal di dunia pengembangan system yang berorientasi objek. Hal ini disebabkan karena UML menyediakan bahasa pemodelan visual yang memungkinkan bagi pengembangan sistem untuk membuat cetak biru atas mereka dalam bentuk yang baku, mudah dimengerti serta dilengkapi dengan mekanisme yang efektif untuk berbagi (*sharing*) dan mengkomunikasikan rancangan mereka dengan yang lain.

Ada beberapa pengklasifikasi (*Classifer*) dan notasi dalam UML (Adi Nugroho, 2010:16) yang ditunjukkan pada Tabel II.6. dibawah ini.

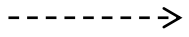
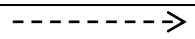
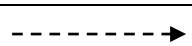
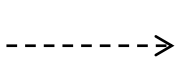
Tabel II.6. Pangklaisfikasi dan Notasi UML

Pengklasifikasi	Kegunaan	Notasi
Actor	Menggambarkan semua objek diluar sistem yang berinteraksi dengan sistem yang dikembangkan.	
Use Case	Menggambarkan fungsionalistis yang dimiliki sistem.	
Kelas (*Class*)	Menggambarkan konsep dasar pemodelan sistem.	
Subsistem (*Subsystem*)	Menggambarkan paket spesifikasi serta implementasi.	
Komponen (*Componen*t)	Menggambarkan bagian-bagian fisik sistem/perangkat lunak yang dikembangkan.	
Antarmuka (*Interface*)	Menggambarkan antarmuka pengiriman pesan (*message*) antar pengklasifikasi.	
Simpul (*Node*)	Menggambarkan sumber daya komputasional yang digunakan oleh sistem.	

(Sumber : Adi Nugroho, 2010:16)

Relasi-relasi antar pengkalsifikasi yang dikenali UML adalah asosiasi, generalisasi, aliran, dan berbagai jenis kebergantungan termasuk didalamnya realisasi dan penggunaan. Untuk lebih jelasnya dapat dilihat pada Tabel II.7. berikut ini :

Tabel II.7. Relasi-relasi dalam UML

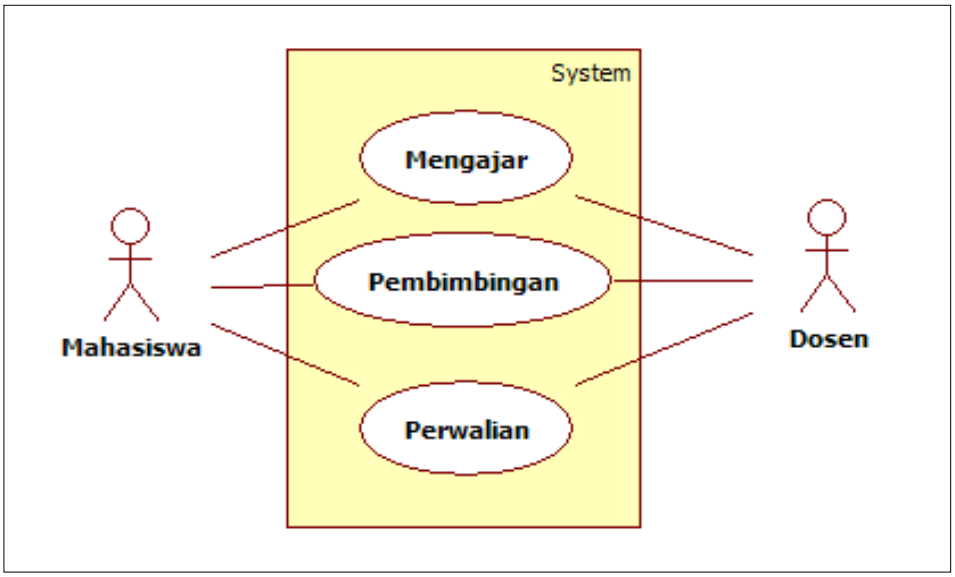
Relasi	Fungsi	Notasi
Asosiasi (*Association*)	Mendeskripsikan hubugan antar instance suatu kelas.	
Kerbergantungan (*Dependency*)	Relasi antar dua elemen model.	
Aliran (*Flow*)	Relasi antar dua versi suatu model.	
Generalisasi (*Generalization*)	Relasi antar pengkalisifikasi yang memiliki deskripsi yang bersifat lebih umum dengan berbagai pengklasifikasi yang lebih spesifik, digunakan dalam struktur pewarisan.	
Realisasi (*Realization*)	Relasi antara spesifikasi dan implementasinya.	
Penggunaan (*Usage*)	Situasi di mana salah satu elemen membutuhkan elemen yang lainnya agar dapat berfungsi dengan baik.	

(Sumber : Adi Nugroho, 2010:23)

II.7.1. *Use Case Diagram*

Diagram ini bersifat statis. Diagram ini memperlihatkan himpunan *use case* dan aktor-aktor (suatu jenis khusus dari kelas). Diagram ini terutama sangat penting untuk mengorganisasi dan memodelkan perilaku dari suatu sistem yang dibutuhkan serta diharapkan pengguna. *Use Case Diagram* adalah fungsionalitas atau persyaratan – persyaratan sistem yang harus dipenuhi oleh sistem yang akan dikembangkan tersebut menurut pandangan pemakai sistem (Sholiq, 2010:21). *Use case* diagram juga dapat diartikan sebagai urutan langkah-langkah yang secara tindakan saling terkait (skenario), baik terotomatisasi maupun secara

manual, untuk tujuan melengkapi satu tugas bisnis tunggal. *Use case* digambarkan dalam bentuk *ellips/oval*.

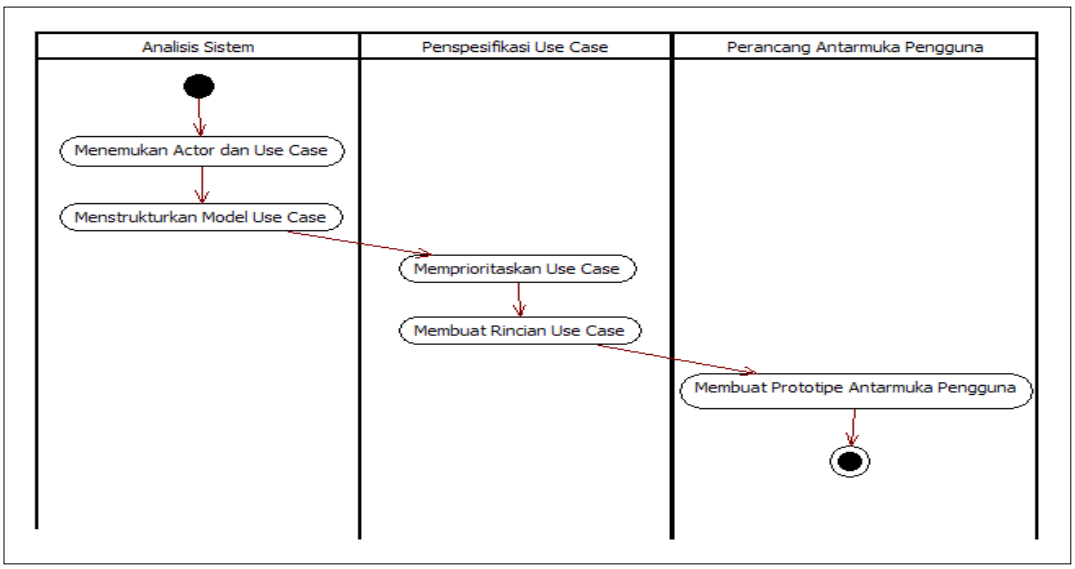


Gambar II.3. Contoh *Use Case Diagram*

(Sumber : Adi Nugroho, 2010:34)

II.7.2. *Activity Diagram*

Acitivity diagram memodelkan alur kerja (*workflow*) sebuah proses bisnis dan urutan aktivitas dalam suatu proses (Sulistyorini, 2009:27). Diagram ini sangat mirip dengan sebuah *flowchart* karena dapat dimodelkan sebuah alur kerja dari satu aktivitas ke aktivitas lainnya atau dari satu aktivitas ke dalam keadaan sesaat (*state*). Seringkali bermanfaat bila dibuat sebuah *activity* terlebih dahulu dalam memodelkan sebuah proses untuk membantu memahami proses secara keseluruhan. *Activity diagram* juga sangat berguna ketika ingin menggambarkan perilaku paralel atau menjelaskan bagaimana perilaku dalam berbagai *use case* berinteraksi.



Gambar II.4. Contoh *Activity Diagram*

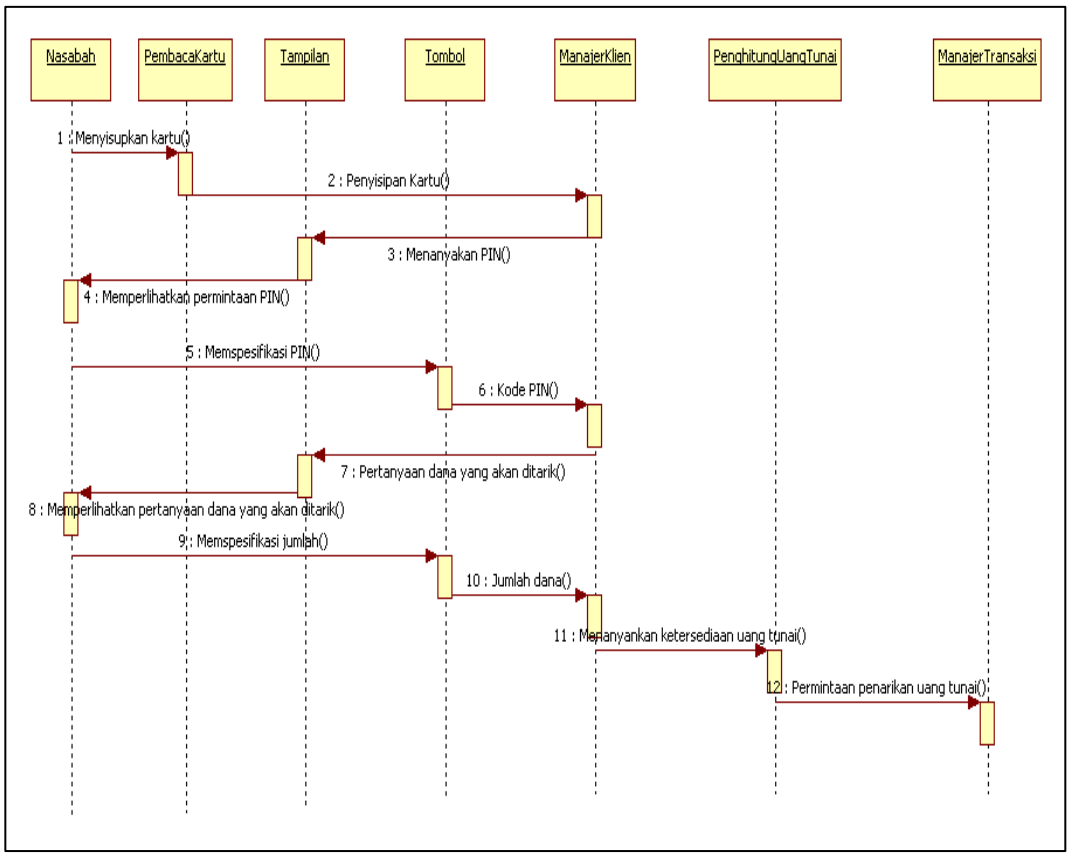
(Sumber : Adi Nugroho, 2010:141)

Dapat digunakan *statechart* diagram untuk memodelkan perilaku dinamis satu kelas atau objek. *Statechart* diagram memperlihatkan urutan keadaan sesaat (*state*) yang dilalui sebuah objek, kejadian yang menyebabkan sebuah transisi dari satu state atau aktivitas ke *state* atau aktivitas lainnya, dan aksi yang menyebabkan perubahan satu state lainnya, dan aksi yang menyebabkan perubahan satu state atau aktivitas. Diagram aktivitas paling cocok digunakan untuk memodelkan aktivitas dalam suatu proses.

II.7.3. *Sequence Diagram*

Sequence Diagram memeperlihatkan interaksi sebagai diagram dua mantra (dimensi). Mantra vertikal adalah sumbu waktu; waktu bertambah dari atas ke bawah sedangkan mantra horizontal memperlihatkan pengklasifikasi yang mempresentasikan objek-objek mandiri yang terlibat dalam kolaborasi. *Diagram sequence* merupakan diagram interaksi yang menekankan pada pengiriman pesan

(*message*) dalam suatu waktu tertentu (Sulistyorini, 2009:24). *Sequence Diagram* digunakan untuk menggambarkan perilaku pada sebuah skenario. Diagram ini menunjukkan sejumlah contoh objek dan *messege* (pesan) yang diletakkan diantara objek-objek ini dalam *use case*.



Gambar II. 5. Contoh *Sequence Diagram*

(Sumber : Adi Nugroh, 2010:109)

Ada beberapa komponen yang terdapat pada *sequence diagram*, yaitu :

1. Objek/*Participant*

   Objek diletakkan di dekat bagian atas diagram dengan urutan dari kiri ke kanan. Objek ini diatur dalam urutan guna menyederhanakan diagram. Setiap *participant* terhubung dengan garis titik-titik yang disebut dengan *lifeline*. Sepanjang *lifeline* ada kotak yang disebut *activation*. *Activation* mewakili

sebuah eksekusi operasi dari *participant*. Panjang kotak ini berbanding lurus dengan durasi *activation*.

2. *Messege*

   Sebuah *messege* bergerak dari satu *participan*t ke *participant* yang lain dan dari satu *lifeline* ke *lifeline*. Sebuah *participant* bisa mengirim sebuah *messege* kepada dirinya sendiri. Jika sebuah *participant* mengirimkan sebuah *messege synchronous*, maka jawaban atas *messege* tersebut akan ditunggu sebelum diproses dengan urusannya. Namun jika *messege synchronous* yang dikirimkan, maka jawaban atas *messege* tersebut tidak perlu ditunggu.

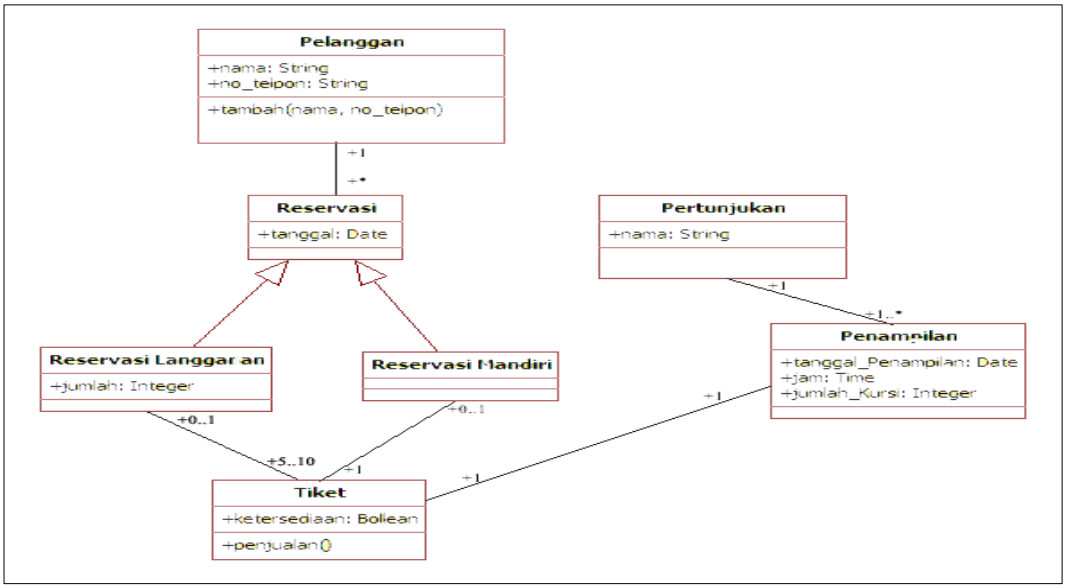
3. *Time*

   Time adalah diagram yang mewakili waktu pada arah *vertical*. Waktu dimulai dari atas ke bawah. *Messege* yang lebih dekat dari atas akan dijalankan terlebih dahulu dibanding *messege* yang lebih dekat ke bawah.

II.7.4. *Class Diagram*

Class Diagram menunjukkan interaksi antar kelas – kelas dalam sistem. Sebuah kelas mengandung informasi dan tingkah laku (*behavior*) yang berkaitan dengan informasi tersebut. *Class diagram* sesungguhnya merupakan deskripsi dari konsep yang datang dari ranah aplikasi atau solusi aplikasi (Adi Nugroho : 2010 :11). Oleh karena itu pengertian kelas sangat penting sebelum merancang diagram kelas. Kelas diagram sebagai satu *set* objek yang memiliki atribut dan prilaku yang sama. *Class diagram* membantu dalam visualisasi struktur kelas – kelas dari suatu sistem dan merupakan tipe diagram yang paling banyak. *Class diagram* memperlihatkan hubungan antar kelas dan penjelasan detail tiap – tiap kelas di

dalam model desain (dalam *logical view*) dari suatu sistem. Selama proses analisi, *class diagram* memperlihatkan aturan – aturan dan tanggung jawab entitas yang menentukan perilaku sistem.



Gambar II.6. ***Contoh Class Diagram***

(Sumber : Adi Nugroho, 2010:12)