# BAB II

# TINJAUAN PUSTAKA

## II.1. Pengertian Sistem

Sistem merupakan sekumpulan elemen - elemen yang saling terintegrasi serta melaksanakan fungsinya masing-masing untuk mencapai tujuan yang telah ditetapkan. Karakteristik sistem terdiri dari :

1. Komponen Sistem

Suatu sistem terdiri dari sejumlah komponen yang saling berinteraksi, yang artinya saling bekerja sama membentuk suatu kesatuan. Komponen - komponen sistem atau elemen - elemen sistem dapat berupa suatu subsistem atau bagian - bagian dari sistem.

2. Batasan Sistem

Batasan merupakan daerah yang membatasi antara suatu sistem dengan sistem yang lainnya atau dengan lingkungan luarnya. Batasan sistem ini memungkinkan suatu sistem dipandang suatu kesatuan. Batasan suatu sistem menunjukkan ruang lingkup (*scope*) dari sistem tersebut.

3. Lingkungan Luar Sistem

Lingkungan luar dari suatu sistem adalah apapun diluar batas dari sistem yang mempengaruhi operasi sistem. Lingkung luar sistem dapat bersifat menguntungkan dan dapat juga bersifat merugikan sistem tersebut.

4. Penghubung Sistem

Penghubung merupakan media penghubung antara satu subsistem dengan subsistem lainnya. Melalui penghubung ini memungkinkan sumber-sumber daya mengalir dari satu subsistem ke subsistem lainnya.

5. Masukan Sistem

Masukan sistem adalah energi yang dimasukkan ke dalam sistem. Masukan dapat berupa masukan perawatan (*maintance input*) dan masukan sinyal (*signal input*).

6. Keluaran Sistem

Keluaran sistem adalah hasil energi yang diolah dan diklasifikasikan menjadi keluaran yang berguna dan sisa pembuangan.

7. Pengolah Sistem

Suatu sistem dapat mempunyai suatu bagian pengolah atau sistem itu sendiri sebagai pengolahnya. Pengolah akan mengubah masukan menjadi keluaran.

8. Sasaran Sistem

Suatu sistem mempunyai tujuan (*goal*) atau sasaran (*objective*). Kalau suatu sistem tidak mempunyai sasaran, maka operasi sistem tidak ada gunanya (Sulindawati ; 2010 : 1-2).

## II.2. Sistem Pakar

Sistem pakar adalah suatu sistem yang dirancang untuk dapat menirukan keahlian seseorang pakar dalam menjawab pertanyaan dan memecahkan suatu masalah. Sistem pakar akan memberikan pemecahan suatu masalah yang didapat

dari dialog dengan pengguna. Dengan bantuan sistem pakar seseorang yang bukan pakar / ahli dapat menjawab pertanyaan, menyelesaikan masalah serta mengambil keputusan yang biasanya dilakukan oleh seorang pakar (T. Sutojo, S. Si., M.Kom. ; 2011 : 13).

### II.2.1. Konsep Dasar Sistem Pakar

Menurut Efraim Turban, Konsep dasar sistem pakar mengandung : keahlian, ahli, pengalihan keahlian, inferensi, aturan dan kemampuan menjelaskan. Keahlian adalah suatu kelebihan penguasaan pengetahuan di bidang tertentu yang diperoleh dari pelatihan, membaca atau pengalaman. Contoh bentuk pengetahuan yang termasuk keahlian adalah :

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) (Ari Fadli ; 2010 : 2).

Bentuk - bentuk ini memungkinkan para ahli untuk dapat mengambil keputusa lebih cepat dan lebih baik daripada seseorang yang bukan ahli. Seorang ahli adalah seseorang yang mampu menjelaskan suatu tanggapan, mempelajari hal-hal baru seputar topik permasalahan (*domain*), menyusun kembali pengetahuan jika dipandang perlu, memecah aturan-aturan jika dibutuhkan, dan menentukan relevan tidaknya keahlian mereka. Pengalihan keahlian dari para ahli

ke komputer untuk kemudian dialihkan lagi ke orang lain yang bukan ahli, merupakan tujuan utama dari sistem pakar. Proses ini membutuhkan 4 aktivitas yaitu :

1. Tambahan pengetahuan (dari para ahli atau sumber - sumber lainnya).
2. Representasi pengetahuan (ke komputer).
3. Inferensi pengetahuan.
4. Pengalihan pengetahuan ke *user*.

Pengetahuan yang disimpan di komputer disebut dengan nama basis pengetahuan. Ada 2 tipe pengetahuan, yaitu : fakta dan prosedur (biasanya berupa aturan). Salah satu fitur yang harus dimiliki oleh sistem pakar adalah kemampuan untuk menalar. Jika keahlian-keahlian sudah tersimpan sebagai basis pengetahuan dan sudah tersedia program yang mampu mengakses basisdata, maka komputer harus dapat diprogram untuk membuat inferensi. Proses inferensi ini dikemas dalam bentuk motor inferensi (*inference engine*). Sebagian besar sistem pakar komersial dibuat dalam bentuk *rule-based systems,* yang mana pengetahuannya disimpan dalam bentuk aturan-aturan. Aturan tersebut biasanya berbentuk *IF-THEN*. Fitur lainnya dari sistem pakar adalah kemampuan untuk merekomendasi. Kemampuan inilah yang membedakan sistem pakar dengan sistem konvensional.

Ada 4 bentuk sistem pakar, yaitu :

1. Berdiri sendiri. Sistem pakar jenis ini merupakan *software* yang berdiri sendiri tidak tergantung dengan *software* yang lainnya.

2. Tergabung. Sistem pakar jenis ini merupakan bagian program yang terkandung didalam suatu algoritma *(*konvensional*)*, atau merupakan program dimana didalamnya memanggil algoritma subrutin lain (konvensional).
3. Menghubungkan ke *software* lain. Bentuk ini biasanya merupakan sistem pakar yang menghubungkan ke suatu paket program tertentu, misalnya *DBMS.*
4. Sistem Mengabdi. Sistem pakar merupakan bagian dari komputer khusus yang dihubungkan dengan suatu fungsi tertentu. Misalnya sistem pakar yang digunakan untuk membantu menganalisis data radar.

Sistem pakar terdiri-dari 2 bagian pokok, yaitu : lingkungan pengembangan (*development environment*) dan lingkungan konsultasi (*consultation environment*). Lingkungan pengembangan digunakan sebagai pembangunan sistem pakar baik dari segi pembangunan komponen maupun basis pengetahuan. Lingkungan konsultasi digunakan oleh seorang yang bukan ahli untuk berkonsultasi.

Basis pengetahuan berisi pengetahuan-pengetahuan dalam penyelesaian masalah, tentu saja di dalam domain tertentu. Ada 2 bentuk pendekatan basis pengetahuan yang sangat umum digunakan, yaitu :

1. Penalaran berbasis aturan (*Rule-Based Reasoning*)

Pada penalaran berbasis aturan, pengetahuan direpresentasikan dengan menggunakan aturan berbentuk : *IF-THEN*. Bentuk ini digunakan apabila kita memiliki sejumlah pengetahuan pakar pada suatu permasalahan tertentu, dan si pakar dapat menelesaikan msalah tersebut secara berurutan. Disamping itu, bentuk

ini juga digunakan apabila dibutuhkan penjelasan tentang jejak (langkah - langkah) pencapaian solusi.

2. Penalaran berbasis kasus (*Case-Based Reasoning*).

Pada penalaran berbasis kasus, basis pengetahuan akan berisi solusi - solusi yang telah dicapai sebelumnya, kemudian akan diturunkan suatu solusi untuk keadaan yang terjadi sekarang (fakta yang ada). Bentuk ini digunakan apabila user menginginkan untuk tahu lebih banyak lagi pada kasus-kasus yang hampir sama (mirip). Selain itu, bentuk ini juga digunakan apabila kita telah memiliki sejumlah situasi atau kasus tertentu dalam basis pengetahuan.

Ada 2 cara yang dapat dikerjakan dalam melakukan inferensi, yaitu :

1. *Forward Chaining.* Pencocokan fakta atau pernyataan dimulai dari bagian sebelah kiri (*IF* dulu). Dengan kata lain, penalaran dimulai dari fakta terlebih dahulu untuk menguji kebenaran hipotesis.
2. *Backward Chaining.* Pencocokan fakta atau pernyataan di mulai dari bagian sebelah kanan (*THEN* dulu). Dengan kata lain, penalaran dimulai dari hipotesis terlebih dahulu, dan untuk menguji kebenara hipotesis tersebut dicari harus dicari fakta-fakta yang ada dalam basis pengetahuan.

Sistem pakar yang baik harus memenuhi ciri-ciri sebagai berikut :

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

Ada beberapa masalah yang menjadi area luas aplikasi sistem pakar, 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. Diagnosis. 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 Systems.*
7. *Debugging,* memberikan resep obat terhadap suatu kegagalan.
8. Perbaikan.
9. Instruksi. Melakukan instruksi untuk diagnosis, *debugging,* dan perbaikan kerja.
10. Kontrol. Melakukan kontrol terhadap interpretasi interpretasi, prediksi, perbaikan, dan *monitoring* kelakukan sistem.

Secara garis besar, banyak manfaat yang dapat diambil dengan adanya sistem pakar, antara lain :

1. Memungkinkan orang awam bisa mengerjakan pekerjaan para ahli.
2. Bisa melakukan proses secara berulang secara otomatis.
3. Menyimpan pengetahuan dan keahlian para pakar.
4. Meningkatkan *output* dan produktivitas.
5. Meningkatkan kualitas.
6. Mampu mengambil dan melestarikan keahlian para pakar (terutama yang termasuk keahlian langka).
7. Mampu beroperasi dalam lingkungan yang berbahaya.
8. Memiliki kemampuan untuk mengakses pengetahuan.
9. Memiliki reliabilitas.
10. Meningkatkan kepabilitas sistem komputer.
11. Memiliki kemampuan untuk bekerja dengan informasi yang tidak lengkap dan mengandung ketidakpastian.
12. Sebagai media pelengkap dalam penelitian.
13. Meningkatkan kapabilitas dalam penyelesaian masalah.
14. Menghemat waktu dalam pengambilan keputusan.

Disamping memiliki beberapa keuntungan, sistem pakar juga memiliki beberapa kelemahan, antara lain :

1. Biaya yang diperlukan untuk membuat dan memeliharanya sangat mahal.
2. Sulit dikembangkan. Hal ini tentu saja erat kaitannya dengan ketersediaan pakar dibidangnya.
3. Sistem Pakar tidak 100% bernilai benar (Ari Fadli ; 2010 : 2-8).

## II.3. Metode *Certainty Factor*

Dalam menghadapi suatu permasalahan sering ditemukan jawaban yang tidak memiliki kepastian penuh. Ketidakpastian ini dapat berupa probabilitas atau kebolehjadian yang tergantung dari hasil suatu kejadian. Hasil yang tidak pasti disebabkan oleh dua faktor, yaitu aturan yang tidak pasti dan jawaban pengguna yang tidak pasti atas suatu pertanyaan yang diajukan oleh sistem. Hal ini sangat mudah dilihat pada sistem diagnosis penyakit, dimana pakar tidak dapat mendefinisikan hubungan antara gejala dengan penyebabnya secara pasti, dan pasien tidak dapat merasakan suatu gejala dengan pasti pula. Pada akhirnya akan ditemukan banyak kemungkinan diagnosis. Sistem pakar harus mampu bekerja dalam ketidakpastian. Sejumlah teori telah ditemukan untuk menyelesaikan ketidakpastian, termasuk diantaranya probabilitas klasik, probabilitas *bayes,* teori *Hartley* berdasarkan himpunan klasik teori *shannon* berdasakan pada probabilitas, teori *Depmster-Shafer, teorifuzzy Zadeh,* dan faktor kepastian (*certainty factor*). Faktor kepastian (*Certanity Factor*) diperkenalkan oleh Shortliffe Buchanan dalam pembuatan *MYCIN*. *Certanity Factor* (*CF*) merupakan nilai parameter klinis yang diberikan *MYCIN* untuk menunjukkan besarnya kepercayaan. *Certanity Factor* (*CF*) menunjukkan ukuran kepastian terhadap suatu fakta atau aturan (Fitrah Rumaisa ; 2010 : 11-12).

*Certainty Factor* (*CF*) menunjukkan ukuran kepastian terhadap suatu fakta atau aturan. Faktor kepastian ini merupakan bentuk penggabungan kepercayaan dan ketidakpercayaan dalam suatu bilangan tunggal. Berikut notasi faktor kepastian :

$$CF[Pk, G] = MB[Pk, G] – MD[Pk, G]$$

Beberapa *evidence* dapat dikombinasikan untuk menentukan *CF* dari suatu hipotesis. Untuk sistem ini, tingkat kepastian sistem terhadap kesimpulan yang diperoleh dihitung berdasarkan nilai probabilitas penyakit karena adanya *evidence* / gejala tertentu. Jika ada gejala dan penyakit sebagai *hipothesis* maka tingkat kepastian diformulasikan sebagai *CF(Pk,G)*:

$$MB(Pk,G) = \begin{cases} 1 & ,P(Pk)=1 \\ \frac{\max[P(Pk|G),P(Pk)]-P(Pk)}{\max[1,0]-P(Pk)} & ,yanglain \end{cases}$$

$$MD(Pk,G) = \begin{cases} 1 & ,P(Pk)=1 \\ \frac{\min[P(Pk|G),P(Pk)]-P(Pk)}{\min[1,0]-P(Pk)} & ,yanglain \end{cases}$$

**Gambar II.1. Formula Tingkat Kepastian**

**Sumber : Bain Khusnul Khotimah ; 2010 : 13**

Di mana :

*P(Pk)* : probabilitas kerusakan Pk

*G* : Gejala

*CF* : *Certainty Factor* (Faktor Kepastian) dalam hipotesis H yang dipengaruhi oleh *evidence* (fakta) E.

*MB* : *Measure of Belief* (Tingkat Keyakinan), merupakan ukuran kepercayaan dari hipotesis H dipengaruhi oleh *evidence* (fakta) E.

*MD* : *Measure of Disbelief* (Tingkat Ketidakyakinan) merupakan ukuran ketidakpercayaan hipotesis H dipengaruhi oleh fakta E.

*H* : Hipotesa atau konklusi yang dihasilkan

Jika ada kaidah lain termasuk dalam hipotesis yang sama tetapi berbeda dalam faktor kepastian, maka perhitungan faktor kepastian dari kaidah yang sama dihitung dari penggabungan fungsi untuk faktor kepastian yang didefinisikan sebagai berikut :

$$CF_{combine}(CF_1, CF_2) = \begin{cases} CF_1 + CF_2(1 - CF_1) & kedua - duanya > 0 \\ \dfrac{CF_1 + CF_2}{1 - \min(|CF_1|, |CF_2|)} & salah\ satu < 0 \\ CF_1 + CF_2(1 - CF_1) & kedua - duanya < 0 \end{cases}$$

**Gambar II.2. Perhitungan Faktor Kepastian Dengan Kaidah Yang Sama**

**Sumber : Bain Khusnul Khotimah ; 2010 : 13**

Di mana, $CF_{combine}$ digunakan bergantung pada apakah faktor kepastian positif atau negatif (Bain Khusnul Khotimah ; 2010 : 13).

**Tabel II.1 Tabel Nilai *Certainty Factor***

| Uncertanty Term | CF |
|---|---|
| 1. Pasti Tidak | - 1.0 |
| 2. Hampir Pasti Tidak | - 0.8 |
| 3. Kemungkinan Besar Tidak | - 0.6 |
| 4. Mungkin Tidak | 0.4 |
| 5. Tidak Tahu | - 0.2 to 0.2 |
| 6. Mungkin | 0.4 |
| 7. Kemungkinan Besar | 0.6 |
| 8. Hampir pasti | 0.8 |
| 9. Pasti | 1.0 |

### II.3.1. Mesin Inferensi (*Inference Engine*)

Ada 2 cara yang dapat dikerjakan dalam melakukan inferensi, yaitu:

1. *Forward Chaining* (Pelacakan ke depan)

Pencocokan fakta atau pernyataan dimulai dari bagian sebelah kiri (*IF* dulu). Dengan kata lain, penalaran dimulai dari fakta terlebih dahulu untuk menguji hipotesis.

2. *Backward Chaining* (Pelacakan ke belakang)

Pencocokan fakta atau pernyataan dimulai dari bagian sebelah kanan (*THEN* dulu). Dengan kata lain, penalaran dimulai dari hipotesis terlebih dahulu, dan untuk menguji kebenaran hipotesis tersebut, harus dicari fakta-fakta yang ada dalam basis pengetahuan (Yasidah Nur Istiqomah ; 2013 : 35).

## II.4. Pengertian *Java*

*Java* adalah sebuah bahasa pemrograman yang diciptakan oleh *James Gosling*, seorang developer dari *Sun Microsystem* pada tahun 1991. Selanjutnya *Java* dikembangkan *Sun Microsystem* dan banyak digunakan untuk menciptakan *Executable Content* yang dapat didistribusikan melalui *network*.

*Java* adalah bahasa pemrograman *Object-Oriented* dengan unsur - unsur seperti bahasa *C++* dan bahasa - bahasa lainnya yang memiliki *libraries* yang cocok untuk lingkungan internet. *Java* dapat melakukan banyak hal dalam melakukan pemrograman, seperti membuat animasi halaman *web*, pemrograman *Java* untuk ponsel dan aplikasi interaktif. *Java* juga dapat digunakan untuk handphone, internet dan lain-lain.

Karakteristik - karakteristik *Java* :

1. Sederhana

Bahasa pemrograman *Java* menggunakan sintaks yang mirip dengan bahasa C++ namun sintaks pada Java telah banyak diperbaiki, terutama dengan menghilangkan pointer yang rumit dan *multiple inheritance*. *Java* juga menggunakan *automatic memory allocation* dan *garbage collection*.

2. Berorientasi Objek

*Java* merupakan bahasa pemrograman berorientasi objek yang memungkinkan program untuk dibuat secara modular dan digunakan kembali.

3. Terdistribusi

*Java* dibuat untuk memudahkan distribusi aplikasi dengan adanya *networking libraries* yang terintegrasi dalam *Java*.

4. *Interpreted*

Program *Java* dijalankan menggunakan program *Interpreted*, yaitu *Java Virtual Machine* (*JVM*). Hal ini menyebabkan *source code Java* yang telah dikompilasi menjadi *byte codes* dapat dijalankan pada berbagai *platform*.

5. *Robust*

*Java* mempunyai reliabilitas yang tinggi. Kompiler pada *Java* mempunyai kemampuan mendeteksi *error* yang lebih baik dibandingkan bahasa pemrograman yang lain. *Java* mempunyai *Runtime Exception Handling* untuk membantu mengatasi *error* pada pemrograman.

6. *Secure*

Sebagai bahasa pemrograman aplikasi internet dan terdistribusi, *Java* memiliki beberapa mekanisme keamanan untuk menjaga agar aplikasi tidak digunakan untuk merusak sistem komputer yang menjalankan aplikasi tersebut.

7. *Architecture Neutral*

Program *Java* tidak bergantung pada platform dimana program akan dijalankan. Cukup dibuat satu program yang dapat dijalankan pada berbagai platform dengan *Java Virtual Machine.*

8. *Portable*

*Source code* maupun program *Java* dapat dengan mudah dibawa ke berbagai *platform* berbeda tanpa harus dikompilasi ulang.

9. *Performance*

Kinerja *Java* sering kali dikatakan kurang, namun kinerja *Java* dapat ditingkatkan menggunakan compiler *Java* lain seperti buatan *Inprise, Microsoft* maupun *Symantec* yang menggunakan *Just In Time Compilers* (*JIT*).

10. *Multithreaded*

Java dapat membuat suatu program yang mampu melakukan beberapa pekerjaan secara sekaligus dan simultan.

11. *Dynamic*

*Java* dapat didesain untuk dapat dijalankan pada lingkungan yang dinamis. Perubahan suatu class dengan menambahkan *properties* ataupun metode dapat dilakukan tanpa menggangu program yang menggunakan class tersebut.

Cara kerja java :

Lingkungan pemrograman *Java* menggunakan kompiler sekaligus interpreter agar dapat berjalan pada *platform* yang berbeda. Kompiler *Java* akan mentransformasikan kode - kode dalam bahasa *Java* ke dalam suatu *bytecode*. *Bytecode* adalah sekumpulan perintah hasil kompilasi yang kemudian dapat dieksekusi melalui sebuah mesin komputer abstrak yang disebut dengan *JVM*. *JVM* juga sering dinamakan sebagai *interpreted*, karena sifatnya yang selalu menerjemahkan kode - kode yang tersimpan dalam *bytecode* dengan cara baris demi baris (Yusni Nyura ; 2010 : 18-19).

## II.5. Pengertian *Database*

*Database* adalah sekumpulan tabel - tabel yang saling berelesasi, relasi tersebut bisa ditunjukkan dengan kunci dari tiap tabel yang ada. Satu *database* menunjukkan satu kumpulan data yang dipakai dalam satu lingkup perusahaan atau instansi.

*Database* mempunyai kegunaan dalam mengatasi penyusunan dan penyimpanan data, maka seringkali masalah yang dihadapi adalah:

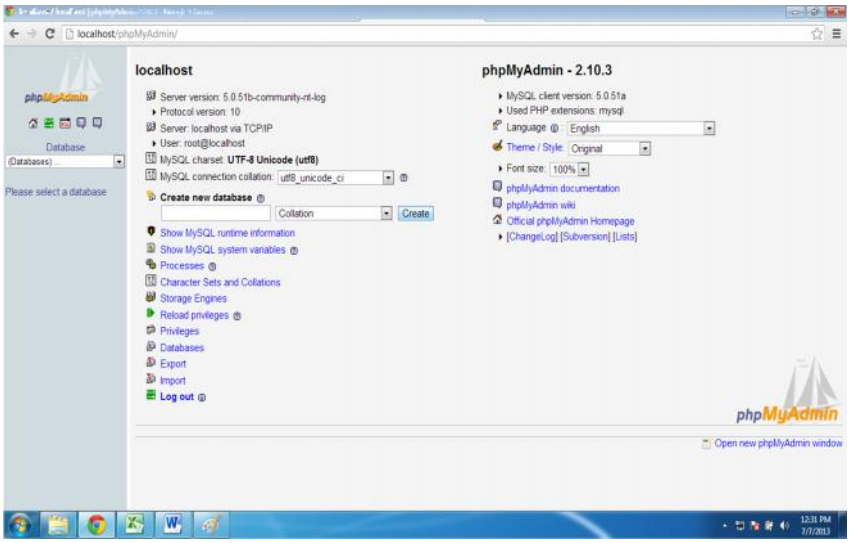
1. Redundansi dan Inkonsistensi data
2. Kesulitan dalam pengaksesan data
3. Isolasi data untuk standarisasi
4. *Multi user*
5. Keamanan data
6. Integritas data

7. Kebebasan data (Asrianda ; 2008 : 1-2).

## II.6. Pengertian *MySQL*

*Mysql* pertama kali dirintis oleh seorang *programmer database* bernama *Michael Widenius*, yang dapat anda hubungi di emailnya monty@analytikerna. *Mysql database server* adalah *RDBMS* (*Relasional Database Management System*) yang dapat menangani data yang bervolume besar. meskipun begitu, tidak menuntut *resource* yang besar. *Mysql* adalah *database* yang paling populer di antara *database* yang lain.

*MySQL* adalah program *database* yang mampu mengirim dan menerima data dengan sangat cepat dan *multi user*. *MySQL* memiliki dua bentuk lisensi, yaitu *free software* dan *shareware*. penulis sendiri dalam menjelaskan buku ini menggunakan *database* ini untuk keperluan pribadi atau usaha tanpa harus membeli atau membayar lisensi, yang berada di bawah lisensi *GNU/GPL* (*general public license*) (Wahana Komputer ; 2010 : 5).




**Gambar II.3. Tampilan MySQL**
**(Sumber : Wahana Komputer ; 2010 : 5)**

## II.7. *Entity Relationship Diagram* (*ERD*)

*Entity Relationship Diagram* atau *ERD* adalah alat pemodelan data utama dan akan membantu mengorganisasi data dalam suatu proyek ke dalam entitas - entitas dan menentukan hubungan antarentitas. Proses memungkinkan analis menghasilkan struktur basisdata yang baik sehingga data dapat disimpan dan diambil secara efisien.

1. Entitas (*Entity*)

Entitas adalah sesuatu yang nyata atau abstrak dimana kita akan menyimpan data. Ada 4 kelas entitas, yaitu misalnya pegawai, pembayaran, kampus dan buku.

2. Relasi (*Relationship*)

Relasi adalah hubungan alamiah yang terjadi antara satu atau lebih entitas, misalnya proses pembayaran pegawai. Kardinalitas menentukan kejadian suatu entitas untuk satu kejadian pada entitas yang berhubungan. Misalnaya, mahasiswa bisa mengambil banyak mata kuliah.

3. Atribut (*Atrtribute*)

Atribut adalah cirri umum semua atau sebagian besar instansi pada entitas tertentu. Sebutan lain atribut adalah properti, elemen data, dan field. Misalnya, nama, alamat, nomor pegawai dan gaji adalah atribut entitas pegawai. Sebuah atribut atau kombinasi atribut yang mengidentifikasikan satu dan hanya satu instansi suatu entitas disebut kunci utama atau pengenal. Misalnya, nama pegawai adalah kunci utama untuk pegawai (Janner Simarmata ; 2010 : 67).

## II.8. Normalisasi

Normalisasi adalah teknik perancangan yang banyak digunakan sebagai pemandu dalam merancang basis data 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 rasional.

Teori normalisasi didasarkan pada konsep bentuk normal. Sebuah tabel relasional dikatakan berada pada bentuk normal tertentu jika tabel memenuhi himpunan batasan tertentu. Ada lima bentuk normal yang tekah ditemukan.

### II.8.1. Bentuk-bentuk Normalisasi

1. **Bentuk normal pertama (1 Normal Form)**

   Contoh yang kita gunakan di sini adalah sebuah perusahaan yang mendapatkan barang dari sejumlah pemasok. Masing-masing pemasok berada pada satu kota. Sebuah kota dapat mempunyai lebih dari satu pemasok dan masing-masing kota mempunyai kode status tersendiri.

   Contoh normalisasi 1NF adalah seperti pada table berikut :

**Tabel 1. Tabel Bentuk Normal Pertama (1NF)**

**Pemasok**

| p# | status | Kota | b# | qty |
|---|---|---|---|---|
| p1 | 20 | Yogyakarta | b1 | 300 |
| p1 | 20 | Yogyakarta | b2 | 200 |
| p1 | 20 | Yogyakarta | b3 | 400 |
| p1 | 20 | Yogyakarta | b4 | 200 |
| p1 | 20 | Yogyakarta | b5 | 100 |
| p1 | 20 | Yogyakarta | b6 | 100 |
| p2 | 10 | Medan | b1 | 300 |

| | | | | |
|---|---|---|---|---|
| p2 | 10 | Medan | b2 | 400 |
| p3 | 10 | Medan | b2 | 200 |
| p4 | 20 | Yogyakarta | b2 | 200 |
| p4 | 20 | Yogyakarta | b4 | 300 |

**Gambar II.4. Tabel Bentuk Normal Pertama (1NF)**

**Sumber : Janner simarmata ; 2010 : 80**

**2. Bentuk normal kedua (2NF)**

Definisi 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 bentuk normal kedua jika dia berada pada 1NF dan setiap kolom bukan kunci yang sepenuhnya tergantung pada seluruh kolom yang membentuk kunci utama.

**Tabel 2. Tabel Bentuk Normal Kedua (2NF)**

**Pemasok2**

| p# | status | Kota |
|---|---|---|
| P1 | 20 | Yogyakarta |
| P2 | 10 | Medan |
| P3 | 10 | Medan |
| P4 | 20 | Yogyakarta |
| P5 | 30 | Bandung |

**Barang**

| p# | b# | qty |
|---|---|---|
| p1 | b1 | 300 |
| p1 | b2 | 200 |
| p1 | b3 | 400 |
| p1 | b4 | 200 |
| p1 | b5 | 100 |
| p1 | b6 | 100 |
| p2 | b1 | 300 |
| p2 | b2 | 400 |
| p3 | b2 | 200 |
| p4 | b2 | 200 |
| p4 | b4 | 300 |
| p4 | b5 | 400 |

**Gambar II.5. Tabel Bentuk Normal Kedua (2NF)**

**Sumber : Janner simarmata ; 2010 : 82**

### 3. Bentuk normal tahap ketiga (3rd normal form)

Bentuk normal ketiga mengharuskan semua kolom pada table relasional tergantung hanya pada kunci utama. Secara definisi, 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.

**Tabel 3. Tabel Bentuk Normal Ketiga (3NF)**

Pemasok Kota

| p# | Kota |
|---|---|
| P1 | Yogyakarta |
| P2 | Medan |
| P3 | Medan |
| P4 | Yogyakarta |
| P5 | Bandung |

Kota Status

| Kota | status |
|---|---|
| Yogyakarta | 20 |
| Medan | 10 |
| Yogyakarta | 20 |
| Bandung | 30 |

**Gambar II.6. Tabel Bentuk Normal Ketiga (3NF)**

**Sumber : Janner simarmata ; 2010 : 82**

### 4. Bentuk Normal Keempat (4NF)

Sebuah tabel rasional berada pada bentuk normal keempat (4NF) jika dia dalam *BCNF* dan semua ketergantungan *multivalue* merupakan ketergantungan fungsional. Bentuk normal keempat (4NF) didasarkan pada konsep ketergantungan *multivalue* (*MVD*). Sebuah ketergantungan multivalue tiga kolom, satu kolom mempunyai banyak baris bernilai sama, tetapi kolom lain bernilai berbeda.

**Tabel 4. Tabel Bentuk Normal Keempat (4NF)**

Pegawai Proyek

| peg# | Pry# |
|---|---|
| 1211 | P1 |
| 1211 | P3 |

Pegawai Ahli

| Peg# | ahli |
|---|---|
| 1211 | Analisis |
| 1211 | Perancangan |
| 1211 | Pemrograman |

**Gambar II.7. Tabel Bentuk Normal Keempat (4NF)**
**Sumber : Janner simarmata ; 2010 : 86**

**5. Bentuk Normal Kelima (5NF)**

Sebuah tabel berada pada bentuk normal kelima (5NF) jika ia tidak dapat mempunyai dekomposisi lossless menjadi sejumlah tabel lebih kecil. Empat bentuk normal pertama berdasarkan pada konsep ketergantungan fungsional, sedangkan bentuk normal kelima berdasarkan pada konsep ketergantungan gabungan (*join dependence*) (Janner Simarmata ; 2010 : 77-86).

**Tabel 6. Tabel Bentuk Normal Kelima (5NF)**

| peg# | Pry# | Ahli |
|---|---|---|
| 1211 | 11 | Perancangan |
| 1211 | 28 | Pemrograman |

**Gambar II.8. Tabel Bentuk Normal Kelima (5NF)**

**Sumber : Janner simarmata ; 2010 : 82**

## II.9. *UML* (*Unified Modeling Language*)

*Unified Modelling Language* (*UML*) adalah sebuah “bahasa” yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan system piranti lunak. *UML* menawarkan sebuah standar untuk merancang model sebuah sistem. Dengan menggunakan *UML* dapat

dibuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena *UML* juga menggunakan class dan operation dalam konsep dasarnya, maka lebih cocok untuk penulisan piranti lunak dalam bahasa berorientasi objek seperti C++, Java, atau *VB. NET*. *DIAGRAM UML* Setiap sistem yang kompleks seharusnya bisa dipandang dari sudut yang berbeda – beda sehingga bisa mendapatkan pemahaman secara menyeluruh . Untuk upaya tersebut *UML* menyediakan 9 jenis diagram yang dapat dikelompokkan berdasarkan sifatnya statis atau dinamis. Ke 9 diagram dalam *UML* itu adalah :

1. Diagram Kelas

Diagram kelas bersifat statis. Diagram ini memperlihatkan himpunan kelas - kelas, antarmuka - antarmuka, kolaborasi - kolaborasi serta relasi.

2. Diagram Objek

Diagram objek bersifat statis. Diagram ini memperlihatkan objek-objek serta relasi antar objek. Diagram objek memperlihatkan instansiasi statis dari segala sesuatu yang dijumpai pada diagram kelas.

3. *Use case* Diagram

Diagram ini bersifat statis. Diagram ini memperlihatkan himpunan *use case* dan actor - aktor (suatu jenis khusus dari kelas). Diagram ini terutama sangat penting untuk mengorganisasi dan memodelkan perilaku dari suatu sistem yang dibutuhkan serta diharapkan pengguna.

4. *Sequence* Diagram (Diagram Urutan)

Diagram ini bersifat dinamis. Diagram *sequence* merupakan diagram interaksi yang menekankan pada pengiriman pesan (*message*) dalam suatu waktu tertentu.

5. *Collaboration* Diagram

Diagram ini bersifat dinamis. Diagram kolaborasi adalah diagram interaksi yang menekankan organisasi struktural dari objek - objek yang menerima serta mengirim pesan (*message*).

6. *Statechart* Diagram

Diagram ini bersifat dinamis. Diagram ini memperlihatkan state – state pada sistem, memuat state, transisi, event, serta aktifitas. Diagram ini terutama penting untuk memperlihatkan sifat dinamis dari antarmuka, kelas, kolaborasi dan terutama penting pada pemodelan sistem – sistem yang reaktif.

Ke 6 diagram ini tidak mutlak harus digunakan dalam pengembangan perangkat lunak, semua dibuat sesuai dengan kebutuhan. (Prastuti Sulistyorini ; 2009 : 23-25).