**BAB II**

**LANDASAN TEORI**

* 1. **Sistem**
     1. **Definisi Sistem**

Sistem adalah kumpulan dari elemen-elemen yang berinteraksi untuk mencapai suatu tujuan tertentu (Jogiyanto, 2005). Kumpulan elemen-elemen tersebut dapat terdiri dari manusia, mesin, prosedur, dokumen, data atau elemen lain yang terorganisir dari elemen-elemen tersebut. Elemen sistem disamping berhubungan satu sama lain, juga berhubungan dengan lingkungannya untuk mencapai tujuan yang telah ditentukan sebelumnya.

Secara sederhana, suatu sistem dapat diartikan sebagai suatu kumpulan atau himpunan dari unsur komponen atau variable yang terorganisir, saling berinteraksi, saling ketergantungan satu sama lain dan terpadu (Sutarbi, 2012).

* + 1. **Pengertian Perancangan Sistem**

Hal yang paling dominan ketika perancangan suatu aplikasi dilakukan adalah memodelkan kebutuhan pemakai. Ada banyak cara untuk memodelkan aplikasi sebagaimana banyak cara yang digunakan oleh seorang arsitek untuk membangun sebuah rumah. Pada dasarnya pemodelan tersebut merupakan kombinasi antara perangkat lunak dan perangkat keras yang digunakan (Whitten et al, 2005).

Perancangan suatu aplikasi termasuk dalam kegiatan rekayasa perangkat lunak. Proses rekayasa perangkat lunak dimulai jauh sebelum *coding* dilakukan dan berlanjut sampai tercapainya sebuah aplikasi yang diinginkan (Pohan, 1997). Pada dasarnya rekayasa perangkat lunak dilakukan untuk merancang suatu aplikasi atau *software* dengan mengurutkan transformasi masalah menjadi solusi perangkat lunak yang dapat bekerja dengan baik.

* + 1. **Perinsip Dasar Perancangan Sistem**

Proses perancangan perangkat lunak merupakan serangkaian kegiatan dan hasil yang berhubungan dengan perangkat lunak, yang bertujuan untuk dihasilkannya suatu produk perangkat lunak. Walaupun ada banyak proses dalam perancangan suatu perangkat lunak, ada kegiatan-kegiatan mendasar yang umum bagi semua proses perancangan perangkat lunak (Sommerville, 2003), antara lain:

1. Penspesifikasian perangkat lunak, fungsionalitas perangkat punak dan batasan operasinya harus didefenisikan.
2. Perancangan dan implementasi perangkat lunak yang memenuhi persyaratan harus dibuat.
3. Validasi perangkat lunak tersebut harus divalidasi untuk menjamin bahwa perangkat lunak bekerja sesuai dengan apa yang diinginkan.
4. Pengevolusian perangkat lunak harus dapat berkembang untuk menghadapi kebutuhan yang dapat berubah sewaktu-waktu

Dalam menciptakan sebuah aplikasi, terdapat beberapa hal yang perlu diperhatikan guna perolehan hasil yang maksimal (Whitten et al, 2005), antara lain sebagai berikut :

1. *Produktivitas*

Saat ini hampir segala bidang memerlukan aplikasi yang dapat digunakan sesuai dengan keperluan dalam bidangnya. Hal ini menyebabkan permintaan terhadap pengadaan aplikasi lebih banyak. Dan tuntutan terhadap kualitas aplikasi yang lebih bagus dan handal. Tentunya hal ini membutuhkan lebih banyak *programmer* dan penganalisa sistem yang berkualitas, kondisi kerja ekstra, kemampuan pemakai untuk mengembangkan sendiri, bahasa pemrograman yang lebih baik, perawatan sistem yang lebih baik, disiplin teknis pemakaian perangkat lunak dan perangkat pengembangan sistem yang terotomasi.

1. *Reliabilitas*

*Reliabilitas* suatu perangkat lunak tidak seperti faktor kualitas lain yang dapat diukur, diarahkan dan diestimasi dengan menggunakan data pengembangan historis. *Reliabilitas* perangkat lunak didefenisikan dalam bentuk statistik sebagai kemungkinan operasi program komputer bebas kegagalan didalam suatu lingkungan dalam kurun waktu tertentu.

1. *Maintabilitas*

*Maintabilitas* mencakup perawatan aplikasi, seperti :

- Koreksi jika ditemukan kesalahan pada program.

- Pengadaptasian jika lingkungan berubah.

- Modifikasi jika pengguna membutukan perubahan kebutuhan.

1. *Integritas*

*Integritas* adalah mengukur kemampuan sistem suatu aplikasi untuk menahan serangan terhadap sekuritasnya. Dalam hal ini kekuatan sistem akan diuji terhadap serangan dari tipe tertentu yang dapat terjadi suatu waktu.

1. *Usabilitas*

*Usabilitas* merupakan ukuran terhadap kualitas interaksi yang terjadi antara aplikasi dengan pengguna. Ukuran usabilitas dapat diketahui melalui tampilan fisik suatu aplikasi (*user friendly*), penggunaan waktu yang efisien dan lain sebagainya.

* 1. **Pengertian Pendaftaran**

Pengertian pendaftaran disini pada dasarnya hanya untuk memperlancar dan mempermudah dalam proses pendaftaran siswa siswi baru, pendataan dan pembagian kelas seorang siswa siswi. Sehingga dapat terorganisir, teratur dengan cepat dan tepat dengan beberapa persyaratan yang telah ditentukan oleh sekolah.

* 1. **Data dan Informasi**

Pengertian Data adalah apapun dan atau semua fakta yang dikumpulkan, disimpan, dan diproses oleh suatu sistem informasi, sedangkan informasi adalah data yang telah diatur dan diproses sehingga dapat memiliki arti (Romney, 2000). Informasi dapat berupa dokumen laporan, atau jawaban suatu pertanyaan. Dokumen merupakan catatan transaksi atau data suatu perusahaan/instansi.

Terdapat enam karakteristik yang membuat informasi menjadi berguna dan berarti :

1. **Relevan**, informasi adalah relevant bila dapat mengurangi ketidakpastian, meningkatkan kemampuan pengambilan keputusan dalam membuat prediksi, atau memastikan, membenarkan pikiran mereka.
2. **Reliable**, informasi adalah reliable bila bebas dari kesalahan atau bisa dan secara tepat menampilkan kejadian yang atau aktifitas organisasi.
3. **Complete**, informasi adalah complete bila dapat mencakup aspek-aspek penting dari kejadian atau aktifitas yang diukurnya.
4. **Timely**, informasi adalah timely bila dapat menyediakan tepat waktu bagi para pembuat keputusan untuk mengunakannya dalam membuat keputusan.
5. **Understandable**, informasi adalah understandable bila informasi yang ditampilkan dengan format yang dapat dibaca dan dimengerti oleh user
6. **Verifiable**, informasi adalah verifiable bila dua orang yang berpengetahauan menghasilkan informasi yang sama.
   1. **Analisa Perancangan Sistem**
      1. **Definisi Analisa Sistem**

Analisa merupakan penguraian suatu persoalan atau permasalahan serta menjelaskan mengenai hubungan antara bagian-bagian yang ada yang kemudian untuk selanjutnya diperoleh suatu penyelesaian secara keseluruhan dari permasalahan yang ada (Maith, 2013). Sedangkan perancangan merupakan sebuah serangkaian prosedur yang dapat digunakan untuk menerjemahkan hasil analisa dari sebuah sistem yang ada ke dalam bahasa pemrograman untuk mendeskripsikan dengan detail bagaimana komponen-komponen sistem diimplementasikan sesuai dengan tujuan yang ditentukan (Amin, Zulfiandri, & Waspodo, 2013).

Analisa sistem dapat diartikan sebagai suatu proses untuk memahami sistem yang ada, dengan menganalisa uraian tugas (*business* *user*), proses bisnis (*business* *prosess*), ketentuan atau aturan (*business* *rule*), masalah dan mencari solusinya (*business* *problem* *and* *solution*) dan rencana-rencana perusahaan (*business* *plan*) (Yakub, 2012).

* + 1. **Fungsi Analisa Sistem**

Adapun fungsi analisa sistem adalah sebagai berikut :

1. Mengidentifikasi masalah-masalah kebutuhan pemakai atau *user*.
2. Menyatakan secara spesifik saran yang harus dicapai untuk memnuhi keutuhan pemakai.
3. Memilih alternatif-alternatif metode pemecahan masalah yang paling tepat.
4. Merencanakan dan menerapkan rancangan sistem yang telah disetujui pemakai user *user*
5. Analisa masukan (*input*), data yang diterima dan akan diolah oleh sistem.
6. Analisa proses, kegiatan memproses masukan yang sudah diterima.
7. Analisa keluaran (*output*), data yang dihasilkan oleh suatu proses setelah dilakukannya proses masukan (*input*).
   1. **Rekam Medis**

Menurut Menurut Permenkes No. 269 Tahun 2008, rekam medis adalah berkas yang berisikan catatan dan dokumen tentang identitas pasien, pemeriksaan, pengobatan, tindakan, dan pelayanan yang telah diberikan kepada pasien. Rekam medis adalah keterangan baik yang tertulis maupun yang terekam tentang identitas, anamneses penentuan fisik laboratorium, diagnosa segala pelayanan dan tindakan medis yang diberikan kepada pasien dan pengobatan fisik yang dirawat inap, rawat jalan maupun yang mendapatkan pelayanan gawat darurat. Selain itu, rekam medis juga berisikan berkas yang berisi catatan dan dokumen tentang identitas pasien, pemeriksaan, pengobatan, tindakan, dan pelayanan lain kepada pasien di sarana pelayanan kesehatan. Rekam medis merupakan dokumen fakta yang berkaitan dengan keadaan pasien, riwayat penyakit dan pengobatan masa lalu, serta saat ini yang tertulis oleh profesi kesehatan yang memberikan pelayanan kepada pasien tersebut (Huffman, 1999: Dalam Permenkes No. 749 Tahun 1989 tentang Rekam Medis disebutkan bahwa rekam medis adalah berkas yang berisikan catatan dan dokumen tentang identitas pasien, pemeriksaan, pengobatan, tindakan dan pelayanan lain kepada pasien pada sarana pelayanan kesehatan. Dijelaskan lebih lanjut dalam Surat Keputusan Direktotar Jenderal Pelayanan medis No. 78 Tahun 1991 tentang Penyelenggaraan Rekam Medis di Rumah Sakit, bahwa rekam medis adalah berkas yang berisikan catatan dan dokumen tentang identitas, anamnesis, pemeriksaan, diagnosis, pengobatan, tindakan dan pelayanan lain yang diberikan kepada seorang pasien selama dirawat di rumah sakit yang dilakukan di unit-unit rawat jalan termasuk unit gawat darurat dan rawat inap.

Tujuan rekam medis adalah menunjang tercapainya tertib administrasi dalam rangka upaya peningkatan pelayanan kesehatan. Tanpa didukung suatu sistem pengelolaan rekam medis yang baik dan benar, tidak mungkin tertib administrasi di tempat pelayanan kesehatan akan berhasil sebagaimana yang diharapkan. Sedangkan tertib administrasi merupakan salah satu faktor yang menentukan di dalam upaya pelayanan kesehatan.

* 1. **Pelayanan**

Menurut Rekam medis mempunyai dua bagian yaitu bagian pertama adalah tentang individu: suatu informasi tentang kondisi kesehatan dan penyakit pasien yang bersangkutan dan sering disebut patient record, bagian kedua adalah tentang manajemen: suatu informasi tentang pertanggungjawaban apakah dari segi manajemen maupun keuangan dari kondisi kesehatan dan penyakit pasien yang bersangkutan. Secara umum, informasi yang tercantum dalam rekam medis seorang pasien adalah sebagai berikut. yaitu :

1. Siapa (who) pasien tersebut dan Siapa (who) yang memberikan pelayanan kesehatan/medis.
2. Apa (what), Kapan (when) , Kenapa (why) dan Bagaimana (how) pelayanan kesehatan/medis diberikan.
3. Hasil akhir atau dampak (outcome) dari pelayanan kesehatan dan pengobatan.
4. Rekam medis merupakan salah satu sumber data penting yang nantinya akan diolah menjadi informasi.

Berdasarkan proses pelayanan rekam medis yang ada pada rumah sakit dapat terlihat bahwa pasien yang datang ke rumah sakit dapat datang sendiri atau membawa surat rujukan. Di unit pendaftaran, identitas pasien dicatat di kartu atau status rekam medis dan selanjutnya pasien beserta kartu atau status rekam medisnya dibawa ke ruang pemeriksaan. Oleh tenaga kesehatan, pasien akan dianamnesia dan diperiksa serta membutuhkan pemeriksaan penunjang. Akhirnya dilakukan penegakkan diagnosa sesuai dengan kebutuhan, pasien tersebut diberi obat atau tindakan medis lainnya. Semua pelayanan kesehatan ini dicatat dalam status rekam medis. Setiap tenaga kesehatan yang melakukan pelayanan kesehatan dan atau tindakan medis harus menuliskan nama dan membubuhi tanda tangannya di status rekam medis tersebut. Semua kegiatan ini merupakan kegiatan bagian pertama rekam medis (patient Record).

Setelah melalui ini semua, pasien dapat pulang atau dirujuk. Kegiatan pengelolaan rekam medis tidak berhenti. Status rekam medis dikumpulkan, biasanya kembali ke ruang rekam medis untuk dilakukan ICD-10 penyakit dan dilakukan pendataan di buku-buku register harian yang telah disediakan. Setelah diolah, status rekam medis disimpan pada tempatnya di ruang arsip agar lain kali pasien yang sama datang, maka status rekam medisnya dapat dipergunakan kembali.

* 1. **Agile Software Development**

Metode *agile* merupakan metode pengembangan *incremental* yang fokus pada perkembangan yang cepat, perangkat lunak yang dirilis bertahap, mengurangi *overhead* proses, dan menghasilkan kode berkualitas tinggi dan pada proses pengembangannya melibatkan pelanggan secara langsung (Sommerville, 2011). Metode *agile* ditemukan oleh tujuh belas pengembang yang sedang berdiskusi untuk menciptakan *lightweight* *development* *methods*. Tujuh belas pengembang mendiskusikan dan merumuskan beberapa tujuan metode *agile* yaitu:

1. *High-value & working App system*, diharapkan dengan memakai *agile* *development* *methods* dapat dihasilkan perangkat lunak yang mempunyai nilai jual yang tinggi, biaya pembuatan bisa di tekan dan perangkat lunak bisa berjalan dengan baik.
2. *Iterative, incremental, evolutionary, agile* adalah metode pengembangan perangkat lunak yang iteraktif, selalu mengalami perubahan, dan revolusioner. Tim harus bekerja dalam waktu yang singkat (biasanya 1-3 minggu) dan juga selalu menambah fungsionalitas dari perangkat lunak sesuai dengan kebutuhan klien. *Agile* dapat dianalogikan ketika seseorang ingin pergi ke suatu kota dan dia tidak tahu jalannya. Lalu agar dapat sampai pada tujuan yaitu dengan sering bertanya kepada orang yang dia temui dijalan hingga dia sampai di tempat tujuan.
3. *Cost control & value-driven development*, salah satu tujuan dari agile yaitu pengembangan perangkat lunak disesuaikan dengan kebutuhan pengguna, tim bisa dengan cepat merespon kebutuhan yang diinginkan pengguna sehingga waktu dan biaya pembuatan perangkat lunak bisa dikontrol.
4. *High-quality production*, walaupun biaya pembuatan perangkat lunak bisa ditekan dan proses pembuatan bisa dipercepat tetapi kualitas dari perangkat lunak yang dibuat harus tetap dijaga. Dengan melakukan tes setiap fungsionalitas perangkat lunak setelah selesai dibuat berarti *agile* juga mengakomodir kebutuhan ini.
5. *Flexible & risk management*, jika menggunakan metode pembuatan yang biasanya dipakai, jika ingin mengubah fungsionalitas dari *wireframe* yang telah dibuat di butuhkan proses yang rumit. Mulai dari pertemuan dengan sistem analis untuk mengubah sistem perangkat lunak, perubahan rencana rilis produk hingga perubahan biaya produksi. Pertemuan dengan klien untuk melakukan tes perangkat lunak juga sering dilakukan sehingga fungsionalitas perangkat lunak mudah diubah dan akhirnya kegagalan perangkat lunak pun bisa diminimalisir.
6. *Collaboration*, dengan menggunakan *agile*, tim pengembang diharuskan sering bertemu untuk membahas perkembangan proyek dan *feedback* dari klien yang nantinya akan ditambahkan dalam perangkat lunak, sehingga tim bisa berkolaborasi dengan maksimal.
7. *Self-organizing, self-managing teams*, rekrut orang terbaik, beri dan dukung kebutuhan mereka lalu biarkan mereka bekerja. Itulah perbedaan *agile* dan metode lainnya. Dengan *agile*, *developer* dapat memanajemen dirinya sendiri, sedangkan manajer tim hanya bertugas mengkolaborasikan *developer* perangkat lunak dengan klien. Sehingga terciptalah tim yang solid.

Semua metode yang dibandingkan dalam memberikan solusi terhadap bisnis. Karena kelima metode tersebut menyediakan iterasi, inkremental, umpan balik, pembelajaran, keterlibatan pengguna, kerjasama tim, dan pemberdayaan tim dalam membuat suatu keputusan. Metode *Agile* dikerjakan oleh tim yang relatif kecil dan menekankan adanya komunikasi informal antar tim. Metode *agile* meminimalkan dokumentasi dan produksi sistematis. Karena metode agile lebih menekankan coding dan testing

Karakteristik dari metode *agile* berbeda-beda. *DSDM* dan *ASD* adalah kerangka kerja yang mengadopsi teknik dari metodologi lainnya, *XP* mendefinisikan satu set eksplisit untuk beradaptasi dengan cepat terhadap perubahan lingkungan dan berfokus pada komunikasi dengan pelanggan. *Crystal* adalah metodologi untuk memilih teknik yang efektif dalam setiap proyek pengembangan perangkat lunak, sedangkan *scrum* berfokus pada manajemen proyek.

*DSDM* dan *ASD* menawarkan detail pada banyak teknik yang digunakan, *XP* mengedepankan teknik dalam mengembangkan perangkat lunak, sedangkan Scrum mengedepankan kejelasan dalam mengelola proyek. Hasil tersebut dikemukakan berdasarkan kerangka kerja analisis yang ia gunakan dalam membandingkan kelima metode *agile*.

* 1. **Extreme Programming (XP)**

*XP* merupakan salah satu pendekatan dalam *agile* *development*. Semua sifat-sifat dan karakteristik yang ada pada *agile* *development* juga terdapat dalam *XP* (Pressman, 2010). Dalam *XP*, terdapat 5 (lima) nilai yang menjadi dasar dalam semua kegiatan *XP*, yaitu :

1. Komunikasi (*communication*)

Untuk mencapai komunikasi yang baik antara *software* *engineers* dan para *stakeholder*, seperti fitur apa yang dibutuhkan dan kegunaan *software*, *XP* menekankan adanya kolaborasi yang baik, komunikasi yang efektif mengenai konsep *software* serta adanya umpan balik terus menerus.

1. Kesederhanaan (*simplicity*)

Dalam mencapai kesederhanaan, *XP* membatasi *developer* untuk merancang kebutuhan yang mendesak dibanding kebutuhan yang akan datang. Hal ini bertujuan agar mudah diimplementasikan pada pemrograman. Jika perancangan ingin diubah atau ditingkatkan, dapat dilakukan *refactoring*. *Refactoring* yang dimaksud adalah mengubah struktur internal desain atau source code tetapi tidak mengubah fungsi eksternal serta perilakunya.

1. Umpan balik (*feedback*)

Umpan balik berasal dari 3 (tiga) sumber yaitu implementasi *software* itu sendiri, *user*, dan anggota *development* team yang lain. Umpan balik diberikan setelah *software* dilakukan pengujian (*testing*) apakah setiap operasi berjalan sesuai fungsi yang ditetapkan. Umpan balik dilihat dari implementasi *output*, fungsi, dan karakteristik *usecase*.

1. Keberanian (*courage*)

Setiap tindakan yang dilakukan dalam praktek *XP* menuntut keberanian atau kata yang tepat adalah disiplin. *Developer* team dalam *XP* harus mempunyai kedisiplinan (keberanian) dalam merancang kebutuhan sekarang, mengenali kebutuhan di masa yang akan datang yang mungkin akan berubah secara drastis, sehingga menuntut untuk merubah perancangan bahkan *source* *code*.

1. Menghargai (*respect*)

Setiap anggota *developer* team memiliki nilai yang harus dihargai oleh sesama anggota dan juga user dalam pengembangan *software* tersebut. Ketika telah mencapai keberhasilan dalam mengembangkan *software*, maka team akan menghargai pula proses *XP* itu sendiri.

Terdapat 4 (empat) tahapan utama dalam perancangan menggunakan *XP* (Pressman, 2010), yaitu:

1. *Planning*

Pada tahapan ini, *developer* team dimulai dengan mengumpulkan persyaratan (*requirement*) yang memungkinkan team mengerti tentang teknikal yang diperlukan dalam pengembangan *software*. Kemudian team mulai melakukan penyusunan *user* *stories* untuk menggambarkan *output* apa yang diperlukan, fitur, dan kegunaan *software* yang akan dibuat. Setiap *story* diberikan nilai dan diurutkan oleh *user*. *Developer* kemudian menentukan waktu yang dikerjakan pada setiap *story* ketika terdapat *story* yang membutuhkan waktu lebih dari tiga minggu, *user* diminta untuk membagi *story* ke beberapa bagian yang lebih sederhana. Saat *developer* mengerjakan, *user* dapat menambahkan *story* baru, mengubah nilai *story* bahkan menghapusnya. *XP* team kemudian mempertimbangkan kembali dan melakukan modifikasi sesuai rencananya.

1. *Design*

*XP* menerapkan prinsip *KIS* (*Keep* *It* *Simple*) dalam desain atau perancangan. Desain yang digunakan menerapkan implementasi dari *story* yang telah dibuat tanpa ditambahkan atau dikurangi.

1. *Coding*

Pada tahapan ini, *XP* mendukung adanya proses *refactoring*. *Refactoring* merupakan proses untuk mengubah *software* system dimana struktur *code* berubah dan menjadi sederhana namun hasil akhir yang ditampilkan tetap sama. Setelah *story* selesai disusun dan tahap awal perancangan telah selesai, *developer* team sebaiknya tidak langsung *coding* melainkan melakukan *testing* dan evaluasi pada setiap *story*. Setelah melewati *testing*, *developer* mulai melakukan *coding*.

1. *Testing*

Pada tahap ini, program yang telah selesai akan dilakukan pengujian *code* dengan menggunakan unit test yang telah tersedia dan dilakukan pengujian dengan menggunakan *acceptance* test yang berasal dari user

* 1. **Perancangan Basis Data**
     1. **Definisi Basis Data**

Basis data (*database*) adalah suatu pengorganisasian sekumpulan data yang sehingga memudahkan aktifitas untuk memperoleh informasi. Basis data dimaksudkan untuk mengatasi problem pada sistem yang memakai pendekatan barbasis berkas. Basis data dapat diartikan sebagai kumpulan data tentang suatu benda atau kejadian yang saling berhubungan satu sama lainnya (Kadir, 2003). Sedangkan data adalah merupakan suatu fakta yang mewakili suatu objek seperti manusia, hewan, peristiwa, konsep, keadaan dan sebainya. Data dicatat atau direkan dalam bentuk angka, huruf, simbol, gambar, bunyi, atau kombinasinya. Kumpulan data tersebut dinamakan dengan sebutan basis data. Perancangan basis data dapat dilakukan melalui beberapa tahap, yaitu:

1. Normalisasi
2. *ERD* (*Entity* *Relationship* *Diagram*)
3. *LRS* (*Logical* *Record* *Structured*)
   * 1. **Normalisasi**

Normalisasi merupakan teknik dalam *logical* desain sebuah basis data atay *database* (Irmansyah, 2003). Dalam mendesain basis data tersebut harus melewati proses pengelompokan data elemen menjadi tabel-tabel yang menunjukkan entitas dan relasinya. Pada proses normalisasi selalu diuji pada beberapa kondisi. Apakah ada kesulitan pada saat menambah (*insert*), menghapus (*delete*), mengubah (*update*), membaca *retrieve* pada satu *database*. Bila ada kesulitan pada pengujian tersebut maka relasi tersebut dipecahkan pada beberapa tabel lagi atau dengan kata lain perancangan belumlah mendapat *database* optimal. Tahap-tahap dalam normalisasi data adalah sebagai berikut:

1. Bentuk tidak normal (*Unnormalized* *Form*)

Bentuk ini adalah merupakan kumpulan data yang akan direkam, tidak ada keharusan mengikuti suatu format tertentu dapat saja data tidak lengkap atau terduplikasi.

1. Bentuk normal kesatu

Merupakan proses yang dilakukan dengan memisahkan data pada *field*. *Field* yang tepat dan berinisial otomatis, seluruh *record* harus lengkap. Bentuk ini berupa flat atau tabel.

1. Bentuk normal kedua

Merupakan proses yang dilakukan dengan jalan membagi data yang ada kedalam beberapa kelompok yang dibedakan menurut *field* kunci masing-masing dan harus mempunyai hubungan satu dengan lainnya.

1. Bentuk normal ketiga

Dalam proses ini masih membagi tabel dalam beberapa bagian supaya tiap tabel tidak mempunyai *field* yang bergantung penuh pada kunci utama.

* + 1. ***Entity Relationship Diagram (ERD)***

*Entity* *Relationship* *Diagram* (*ERD*) merupakan suatu alat bantu untuk melakukan pemodelan basis data (Jogiyanto, 1990). *ERD* juga merupakan suatu model jaringan yang menggunakan susunan data yang tersimpan secara sistem atau tehnik menggambar suatu skema *database* dimana setiap komponen yang terlibat dalam *ERD* memiliki atribut masing-masing yang mempresentasikan fakta dari dunia nyata yang sedang ditinjau. Definisi Kardinalitas relasi menunjukan jumlah maksimum tupel yang dapat berelasi dengan entitas pada entitas lain. Yang merupakan informasi yang dibuat, disimpan dan digunakan dalam sistem bisnis. Terdapat 3 macam kardinalitas relasi yaitu sebagai berikut:

1. Relasi satu ke satu (*One* *to* *One*).
2. Relasi Satu ke Banyak atau Banyak ke Satu (*One* *to* *Many* atau *Many* *to* *One*).
3. Relasi Banyak-ke-Banyak (*Many* *to* *Many*). Tingkat hubungan banyak ke banyak terjadi jika tiap kejadian pada sebuah entitas akan mempunyai banyak hubungan dengan kejadian pada entitas lainnya, dilihat dari sisi entitas yang pertama maupun dilihat dari sisi yang kedua.

**Tabel 2.1. Tabel keterangan *class diagram***

|  |  |  |  |
| --- | --- | --- | --- |
| **NO** | **GAMBAR** | **NAMA** | **KETERANGAN** |
| 1 |  | Entitas | Persegi panjang menyatakan himpunan entitas adalah orang, kajadian, atau berada di mana data akan dikumpulkan. |
| 2 |  | Relationship | Belah ketupat menyatakan himpunan relasi merupakan hubungan antar entitas. |
| 3 |  | Connector  Link | Garis sebagai penghubung antara himpunan, relasi, dan himpunan entitas dengan atributnya. |
| 4 |  | Weak Entity | Merupakan entitas yang tidak memiliki primary key dan tergantung pada primary key entitas lain. |

* + 1. **Transformasi *ERD* ke bentuk *LRS***

Transformasi diagram *ERD* ke *LRS* merupakan suatu kegiatan untuk membentuk data-data dari diagram hubungan entitas ke suatu *LRS* (Tabrani, 2014). Diagram *ERD* di atas akan ditransformasikan ke bentuk *LRS*. Berikut adalah perubahan yang terjadi mengikuti aturan-aturan sebagai berikut :

1. Jika hubungan yang terjadi antara set entitas yang satu dengan yang lainnya adalah satu ke satu (*one to one*) maka atribut dari relationship set diambil dan dimasukan ke set entitas yang tidak memiliki atribut tersebut.
2. Jika hubungan yang terjadi antara set entitas yang satu dengan set yang entitas yang lainnya adalah satu ke banyak (*one to many*) maka atribut dari *relationship* set digambungkan dengan set entitas mempunyai kardinalitas yang banyak.
3. Jika hubungan yang terjadi antara set entitas yang satu dengan set yang lainnya adalah banyak ke banyak (*many* *to* *many*) maka atribut dari *relationship* set dibentuk menjadi satu set entitas.
   * 1. ***Logical Record Structured (LRS)***

*LRS* (*Logical* *Record* *Structure*) dibentuk dengan nomor dari tipe *record* (Tabrani, 2014). LRS juga merpakan representasi dari struktur *record*-*record* pada tabel-tabel yang terbentuk dari hasil antar himpunan entitas. Setelah *ERD* ditransformasikan ke bentuk *LRS*, maka hasil akhir dari proses transformasi tersebut adalah sebuah diagram yang sudah dapat menggambarkan basis data yang akan digunakan. *LRS* terdiri dari tipe *record*, yang berupa sebuah persegi dengan *field* yang dibutuhkan di dalamnya. *LRS* terdiri juga dari hubungan antara tipe *record* tersebut.

* 1. **Perancangan Sistem (*Unified Modeling Language*)**

Perancangan sistem adalah suatu pemodelan secara visual yang penggambaran dan perencanaan yang terdiri dari obyek, unsur-unsur atau suatu komponen yang berkaitan serta saling berhubungan antara satu sama lainnya, sehingga hasil dari unsur itu merupakan satu kesatuan (Yuniarthe, 2013).

* + 1. ***Usecase Diagram***

*Usecase* adalah salah satu diagram yang ada dalam UML (*Unified Modeling Language*) yang merupakan sebuah konstruksi untuk mendeskripsikan bagaimana sistem akan terlihat di mata pengguna potensial. *Usecase* terdiri dari sekumpulan skenario yang dilakukan oleh seorang aktor (orang, perangkat keras, urutan waktu atau sistem yang lain). Sedangkan *usecase* diagram memfasilitasi komunikasi di antara analis dan mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan aplikasi yang akan dibuat. Secara kasar, use case diagram digunakan untuk mengetahui fungsi atau proses apa saja yang ada di dalam sebuah aplikasi dan siapa saja yang berhak menggunakan fungsi-fungsi / proses-proses itu (Imbar & Kurniawan, 2012).

**Tabel 2.2. Tabel keterangan *usecase diagram***

|  |  |  |  |
| --- | --- | --- | --- |
| **NO** | **GAMBAR** | **NAMA** | **KETERANGAN** |
| 1 |  | *Actor* | Menspesifikasikan himpunan peran yang pengguna mainkan ketika berinteraksi dengan *use case*. |
| 2 |  | *Dependency* | Hubungan dimana perubahan yang terjadi pada suatu elemen mandiri *(independent)* akan mempengaruhi elemen yang bergantung padanya elemen yang tidak mandiri |
| 3 |  | *Generalization* | Hubungan dimana objek anak (*descendent*) berbagi perilaku dan struktur data dari objek yang ada di atasnya objek induk (*ancestor*). |
| 4 |  | *Include* | Menspesifikasikan bahwa *use case* sumber secara *eksplisit*. |
| 5 |  | *Extend* | Menspesifikasikan bahwa *use case* target memperluas perilaku dari *use case* sumber pada suatu titik yang diberikan. |
| 6 |  | *Association* | Apa yang menghubungkan antara objek satu dengan objek lainnya. |
| 7 |  | *System* | Menspesifikasikan paket yang menampilkan sistem secara terbatas. |
| 8 |  | *Case* | Deskripsi dari urutan aksi-aksi yang ditampilkan sistem yang menghasilkan suatu hasil yang terukur bagi suatu actor |
| 9 |  | *Collaboration* | Interaksi aturan-aturan dan elemen lain yang bekerja sama untuk menyediakan prilaku yang lebih besar dari jumlah danelemen-elemennya (sinergi). |
| 10 |  | *Note* | Elemen fisik yang eksis saat aplikasi dijalankan dan mencerminkan suatu sumber daya komputasi |

* + 1. ***Activity Diagram***

*Activity* *diagram* merupakan alur kegiatan dalam sistem yang sedang dirancang, bagaimana masing-masing alur berawal sampai berakhir (Wahono, 2003). *Activity* *diagram* tidak menggambarkan kegiatan internal sebuah sistem (dan interaksi antar sub sistem) secara ekstrak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari *level* atas secara umum. Menggambarkan proses bisnis dan urutan aktivitas dalam sebuah proses. Dipakai pada business modeling untuk memperlihatkan urutan aktifitas proses bisnis. Struktur diagram ini mirip *flowchart* atau data *flow* diagram pada perancangan terstruktur.

**Tabel 2.3. Tabel keterangan *activity diagram***

|  |  |  |  |
| --- | --- | --- | --- |
| **NO** | **GAMBAR** | **NAMA** | **KETERANGAN** |
| 1 |  | *Actifity* | Activity menggambarkan sebuah pekerjaan/tugas dalam workflow. |
| 2 |  | *Action* | State dari sistem yang mencerminkan eksekusi dari suatu aksi |
| 3 |  | *Initial Node* | *Initial* *node* atau *start* *state* dengan tegas menunjukkan dimulainya suatu *workflow* pada sebuah *activity* *diagram*. Start state digambarkan dengan simbol lingkaran yang solid. |
| 4 |  | *Final Node* | *Final* *node* atau *end* *state* menggambarkan akhir atau terminal dari pada sebuah activity diagram. Bisa terdapat lebih dari satu end state pada sebuah activity diagram. |
| 5 |  | State Transition | State transition menunjukkan kegiatan apa berikutnya setelah suatu kegiatan sebelumnya. |
| 6 |  | Decision | Decision adalah suatu titik/point pada activity diagram yang mengindikasikan suatu kondisi dimana ada kemungkinan perbedaan transisi. |
| 7 |  | Swimlane | Swimlane adalah digunakan untuk membagi partisi diagram aktivitas untuk membantu user |

* + 1. ***Squence Diagram***

*Sequence* *Diagram* menggambarkan skenario atau langkah-langkah yang dilakukan sebagai respon dari sebuah *event* untuk menghasilkan *output* tertentu (Dharwiyanti, 2003). Skenario objek pada use case dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambarkan diagram sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah *usecase* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu. Biasanya *sequence* diagram dibuat untuk setiap *usecase* pada *usecase* *diagram*. Banyaknya *sequence* *diagram* yang harus digambarkan adalah sebanyak pendefinisian *usecase* yang memiliki proses sendiri atau yang penting semua *usecase* yang telah mendefinisikan interaksi jalannya pesan sudah dicakup pada *sequence* *diagram* sehingga semakin banyak *usecase* yang didefinisikan maka *sequence* *diagram* yang harus dibuat juga semakin banyak. Berikut simbol-simbol yang ada pada *sequence* *diagram*.

**Tabel 2.4. Tabel keterangan *squence diagram***

|  |  |  |  |
| --- | --- | --- | --- |
| **NO** | **GAMBAR** | **NAMA** | **KETERANGAN** |
| 1 |  | *Actor* | Aktor adalah seseorang atau sesuatu diluar sistem yang harus berinteraksi dengan sistem. |
| 2 |  | *Entity* | Digambarkan dengan bentuk persegi panjang. Entity adalah sesuatu apa saja yang ada dalam system, nyata maupun abstrak dimana data disimpan atau dimana terdapat data. |
| 3 |  | *Boundary* | Adalah yang memodelkan interaksi antara satu atau lebih actor dengan sistem. |
| 4 |  | *Control* | Digunakan untuk memodelkan “Perilaku mengatur”, khusus untuk 1 atau beberapa use case saja. |
| 5 |  | Entitas | Memodelkan informasi yang harus disampaikan oleh sistem. |
| 6 |  | *call* | arah panah mengarah pada objek yang memiliki operasi atau metode, karena arah ini memanggil operasi atau metode maka yang di-panggil harus ada pada diagram kelas sesuai dengan kelas objek yang berinteraksi. |
| 7 | *1: masukan* | *send* | Menyatakan bahwa suatu objek mengirimkan data atay masukan informasi ke objek lainnya, arah panah mengarah pada objek yang dikirimi. |
| 8 | *1: keluaran* | *return* | Menyatakan bahwa suatu objek yang telah menjalankan suatu operasi atau metode, menghasilkan suatu kembalian ke objek tertentu, arah panah mengarah pada objek ang menerima kembalian |
| 9 |  | *destroy* | Menyatakan suatu objek mengakhiri hidup objek yang lain, arah panah mengarah pada objek yang diakhiri, sebaiknya jika ada *create* maka akan ada *destroy* |

* + 1. ***Class Diagram***

*Class* *Diagram* menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain (Dharwiyanti, 2003). *Class* *diagram* juga merupakan salah satu diagram yang ada pada *UML*. Diagram kelas atau class diagram menggambarkan struktur aplikasi berorientasi objek dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun aplikasi. Kelas memiliki apa yang disebut atribut dan metode atau operasi.

1. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas.
2. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

**Tabel 2.5. Tabel keterangan *class diagram***

|  |  |  |  |
| --- | --- | --- | --- |
| **NO** | **GAMBAR** | **NAMA** | **KETERANGAN** |
| 1 |  | *Package* | *Package* merupakan sebuah bungkusan dari satu atau lebih kelas |
| 2 | |  | | --- | | Nama Kelas | | + atribut | | + operasi | | *Class* | Kelas pada struktur sistem |
| 3 | INTERFACE\_NAME | *interface* | Sama dengan konsep *interface* dalam pemrograman berorientasi objek |
| 4 |  | Asosiasi (*association*) | Relasi antarkelas dengan makna umum, asosiasi biasanya juga disertai dengan *multiplicity* |
| 4 |  | Ketergantungan (*dependency*) | Relasi antar kelas dengan makna keterkaitan antar kelas |
| 5 |  | Agregasi (*aggregation*) | Relasi antarkelas dengan makna semua-bagian (*whole-part*) |

* 1. **Jaringan Komputer**

Jaringan komputer adalah sekumpulan peralatan atau komputer yang saling dihubungkan untuk berbagi sumber daya. (Andi Micro, 2011 : 6) Agar terjadi jaringan antar komputer maka setiap bagian dari jaringan komputer meminta dan memberikan layanan (*servis*). Pihak yang meminta layanan disebut client dan yang memberi layanan disebut *server*.

Jaringan komputer dengan jangkauan lokal area (*LAN*) adalah jaringan yang dibatasi oleh area yang relatif kecil, umumnya dibatasi oleh area lingkungan seperti kantor pada sebuah gedung atau tiap-tiap ruangan pada sebuah sekolah. Keuntungan dari jenis jaringan *LAN* seperti lebih irit dalam pengeluaran biaya operasional, lebih irit dalam penggunaan kabel, transfer data antar node dan komputer labih cepat karena mencakup wilayah yang sempit atau lokal, dan tidak memerlukan operator telekomunikasi untuk membuat sebuah jaringan *LAN*.

* 1. ***Website***

*Website* merupakan salah satu sumber daya *internet* yang berkembang pesat. Pendistribusian informasi *website* dilakukan melalui pendekatan *hyperlink*, yang memungkinkan suatu teks, gambar, ataupun objek yang lain menjadi acuan untuk membuka halaman-halaman yang lain. Melalui pendekatan ini, seseorang dapat memperoleh informasi dengan beranjak dari satu halaman ke halaman lain. (Abdul Kadir, 2005).

* 1. **Aplikasi *Website***

Pada awalnya aplikasi web dibangun hanya dengan menggunakan bahasa yang disebut *HTML* (*HyperText* *Markup* *Languange*) dan protokol yang digunakan dinamakan *HTTP* (*HyperText* *Transfer* *Protocol*). Pada peerkembangan berikutnya, sejumlah *script* dan objek yang dikembangkan untuk memperluas kemampuan *HTML*. Pada saat ini, banyak *script* seperti itu; antara lain *PHP* dan *ASP*, sedangkan contoh yang berupa objek antara lain adalah *applet* (*Java*). (Abdul Kadir, 2005).

* 1. **Antar Muka (*User* *Interface*)**

Pengguna sering menilai sistem dari *interface*, bukan dari fungsinya. Jika desain user *interface*nya yang buruk, maka itu sering jadi alasan untuk tidak menggunakan sebuah aplikasi. Selain itu *interface* yang buruk dapat menyebabkan pengguna membuat kesalahan yang fatal. Pada subbab berikutnya akan dijelaskan mengenai definisi, macam-macam *UI*, komponen antarmuka dan prinsip-prinsip umum *user* *interface.*

* + 1. **Definisi *User* *Interface* (*UI*)**

*User* *interface* (*UI*) atau antarmuka pemakai merupakan bagian dari komputer dan perangkat lunaknya yang dapat dilihat, didengar, disentuh, dan

diajak bicara, baik secara langsung maupun dengan proses pemahaman tertentu. *User* *interface* yang baik adalah *user* *interface* yang tidak disadari, dan *user* *interface* yang memungkinkan pengguna fokus pada informasi dan task tanpa perlu mengetahui mekanisme untuk menampilkan informasi dan melakukan task tersebut.

* + 1. **Macam-macam *User* *Interface* (*UI*)**

*User* *interface* (*UI*) terdiri dari dua jenis, diantaranya adalah sebagai berikut:

1. *Graphical*

*User* *Interface* atau (*GUI*) *Graphical* *user* *interface* biasanya menggunakan unsur-unsur multimedia seperti gambar, suara dan video agar dapat berkomunikasi dengan pengguna.

1. *Text* *Based*

*Text* *Based* biasanya menggunakan sintaks atau rumus yang sudah ditentukan untuk memberikan perintah.

* 1. **Pengujian**

Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain, dan pengkodean. Pentingnya pengujian perangkat lunak dan implikasinya yang mengacu pada kualitas perangkat lunak tidak dapat terlalu ditekan karena melibatkan sederetan aktivitas produksi di mana peluang terjadinya kesalahan manusia sangat besar dan arena ketidakmampuan manusia untuk melakukan dan berkomunikasi dengan sempurna maka pengembangan perangkat lunak diiringi dengan aktivitas jaminan kualitas. Meningkatnya visibilitas (kemampuan) perangkat lunak sebagai suatu elemen sistem dan biaya yang muncul akibat kegagalan perangkat lunak.

* + 1. **Pengujian *Black* *Box***

Pengujian *black* *box* adalah metode pengujian yang berfokus pada domain informasi dari perangkat lunak, dengan melakukan *test* *case* dengan mempartisi domain input dari suatu program dengan cara yang memberikan cakupan pengujian yang mendalam (Hendrawansyah & Catur, 2009). Metode pengujian *graph*-*based* mengeksplorasi hubungan antara dan tingkah laku objek-objek program. Partisi ekivalensi membagi domain input ke dalam kelas data yang mungkin untuk melakukan fungsi perangkat lunak tertentu. Analisis nilai batas memeriksaa kemampuan program untuk menangani data pada batas yang dapat diterima.

Metode pengujian yang terspesialisasi meliputi sejumlah luas kemampuan perangkat lunak dan area aplikasi. *GUI* (*Graphical* *User* *Interface*), arsitektur *client*/ *server*, dokumentasi dan fasilitas *help* dan sistem *real* *time* masing-masing membutuhkan pedoman dan teknik khusus untuk pengujian perangkat lunak. Pengujian *black*-*box* didesain untuk mengungkap kesalahan pada persyaratan fungsional tanpa mengabaikan kerja internal dari suatu program.

* + 1. **Pengujian *White Box***

Pengujian *white* *box* merupakan pengujian *basis* *path*. Dengan menggunakan basis path ini memungkinkan desainer *test* *case* mengukur kompleksitas logis dari desain *procedural* dan menggunakannya sebagai pedoman untuk menetapkan basis set dari setiap jalur eksekusi (Hendrawansyah & Catur, 2009). Cara pengujian *white* *box* juga dapat dengan melihat ke dalam modul untuk meneliti kode-kode program yang ada, dan menganalisis apakah ada kesalahan atau tidak. Jika ada modul yang menghasilkan *output* yang tidak sesuai dengan proses bisnis yang dilakukan, maka baris-baris program, variabel, dan parameter yang terlibat pada unit tersebut akan dicek satu persatu dan diperbaiki, kemudian di-*compile* ulang.

Pengujian white box berfokus pada struktur kontrol program. *Test* *case* dilakukan untuk memastikan bahwa semua pernyataan pada program telah dieksekusi paling tidak satu kali selama pengujian dan bahwa semua kondisi logis telah diuji. Pengujian *basic* *path*, teknik pengujian white box, menggunakan grafik (*matriks* *grafiks*) untuk melakukan serangkaian pengujian yang *independent* secara linier yang akan memastikan cakupan. Pengujian aliran data dan kondisi lebih lanjut menggunakan logika program dan pengujian *loop* menyempurnakan teknik *white* *box* yang lain dengan memberikan sebuah prosedur untuk menguji *loop* dari tingkat kompleksitas yang bervariasi. memotivasi dilakukannya perencanaan yang baik melalui pengujian yang teliti. Pada dasarnya, pengujian merupakan satu langkah dalam proses rekayasa perangkat lunak yang dapat dianggap sebagai hal yang merusak dari pada membangun.

* 1. **Aplikasi Pendukung**
     1. ***HTML (Hyper Text Markup Language)***

*HTML* merupakan suatu halaman yang berada pada suatu situs internet atau *web* (Kuswayatno, 2011). *HTML* salah satu varian dari *SGML* (*Standard* *Generalized* *Markup* *Language*), yaitu sebuah standar dari *ISO* (*International* *Organization* *for* *Standarization*) untuk pertukaran dokumen secara elektronik melalui protocol *HTTP*. *HTML* sendiri merupakan tata penulisan yang digunakan dalam dokumen *web*. Dokumen ini, akan menghasilkan suatu dokumen sesuai dengan keinginan yang mendesain *page* Dokumen ini mempunyai kemampuan menampilkan gambar, suara, *text*, maupun penyediaan lirik terhadap halaman *web* lainnya, baik dengan alamat yang sama serta alamat yang berbeda. *HTML* sendiri secara formal diumumkan sebagai RFC 1866.

* + 1. ***PHP (Hypertext Preprocessor)***

*PHP* merupakan bahasa *script* yang menyatu dengan *HTML* dan dijalankan pada *server* *side*. *PHP* dibuat sebagai perantara yang memungkinkan aplikasi bisa menghasilkan sesuatu yang bersifat dinamis dan dapat berinteraksi dengan *database*. Cara kerja *PHP* yaitu *PHP* akan melakukan permintaan pada *database* *server* dan hasil dari *database* *server* akan diproses lebih lanjut (Kadir, 2009). *PHP* awalnya merupakan program *CGI (Common Getway Interface)* yang dikhususkan untuk menerima input melalui form yang ditampilkan dalam *web* *browser* kemudian *software* ini disebarkan dan dilisensikan sebagai perangkat lunak *open* *source*. *PHP* secara resmi merupakan kependekan dari *PHP*: *Hypertext* *Processor*, merupakan bahasa *script* *server* *side* yang disisipkan dalam dokumen *HTML*.

* + 1. ***CSS (Cascading Style Sheets)***

*Css* merupakan sebuah dokumen yang dapat berdiri sendiri yang dapat dimasukan kedalam kode *HTML* (Fasuri, 2011). *Css* biasanya digunakan dalam dokumen *HTML* dalam menciptakan suatu *style* (penyajian) yang dapat membuat kemampuan *HTML* menjadi lebih luas. Yang menarik, *style* dapat didefinisikan pada *file* terpisah sehingga tidak menambah keruwetan dalam dokumen. Selain itu tentu saja *style* dapat digunakan pada sejumlah dokumen. Salah satu kegunaan *CSS* yang dibahas di dalam dokumen ini adalah untuk mengimplementasikan *drag* *and* *drop*.

* + 1. ***JavaScript***

*Javascript* adalah bahasa skrip yang ditempelkan pada kode *HTML* dan diproses di sisi klien (Ramadhan, 2007). Dengan adanya bahasa ini, kemampuan dokumen *HTML* menjadi semakin luas. Sebagai contoh, dengan menggunakan *Javascript* dimungkinkan untuk memvalidasi masukan masukan pada formulir sebelum formulir dikirimkan ke *server*. *Javascript* bukanlah bahasa *Java* dan merupakan dua bahasa yang berbeda. *Javascript* diinterpretasikan oleh klien (kodenya bisa dilihat pada sisi klien), sedangkan kode *Java* dikompilasi oleh pemrogram dan hasil kompilasinyalah yang dijalankan oleh klien.

* + 1. ***MySQL***

*MySQL* (*My* *Structured* *Query* *Language*) adalah suatu perangkat lunak *database* relasi (*Relational* *Database* *Management* *System*) atau *RDBMS*. *MySQL* itu bekerja menggunakan bahasa basis data atau sering disebut dengar dengan sebutan *DBMS* (*Database* *Management* *System*). *Data* *Language* ini terbagi dua macam, yaitu:

* + - 1. *DDL* (*Data* *Definition* *Language*)

*DDL* yaitu perintah yang digunakan untuk pendefinisian suatu struktur data. Misalnya menciptakan *database*, *field*, dan sebagainya.

1. *CREATE*, untuk membuat objek
2. *ALTER*, untuk mengubah struktur objek
3. *DROP*, untuk menghapus objek
4. *RENAME*, untuk memberi nama baru objek
5. *TRUNCATE*, untuk mengkosongkan objek, dll
   * + 1. *DML* (*Database* *Manipulation* *Language*)

*DML* yaitu perintah untuk proses manipulasi data, misalnya *create*, *read*, *update*, *delete* (*CRUD*) :

1. *SELECT*, untuk memilih data
2. *INSERT* , untuk menambahkan data
3. *UPDATE*, untuk mengubah data
4. *DELETE*, untuk menghapus data

*MySQL* menggunakan bahasa *SQL* (*Structure* *Query* *Language*) adalah bahasa *query* basis data yang baku bagi seluruh dunia dan bahasa yang digunakan untuk berkomunikasi antara pemakai dan komputer. *MySQL* kembali dipublikasikan sejak tahun 1996, tetapi sejarah pengembangannya telah dilakukan dari tahun 1979. *MySQL* tersedia dengan lisensi *open* *source* (Welling, 2001), tetapi lisensi komersial pun tersedia apabila diperlukan.