BAB II

TINJAUAN PUSTAKA

II.1. Sistem

II.1.1. Konsep Dasar Sistem

Sistem merupakan kumpulan dari unsur atau elemen-elemen yang saling berkaitan / berinteraksi dan saling mempengaruhi dalam melakukan kegiatan bersama untuk mencapai suatu tujuan tertentu. Menurut Jerry FithGerald, sistem adalah suatu jaringan kerja dari prosedur-prosedur yang saling berhubungan dan berkumpul bersama-sama untuk melakukan suatu kegiatan atau menyelesaikan suatu sasaran tertentu. Contoh : Sistem komputer, terdiri dari software, hardware, dan brainware (Hendra ; 2012 : 157).

Menurut Hendra (2012 : 163) berdasarkan prinsip dasar secara umum, sistem terbagi dalam :

1. *Sistem Terspesialisasi* adalah sistem yang sulit diterapkan pada lingkungan yang berbeda, misalnya sistem biologi: ikan yang dipindahkan ke darat.
2. *Sistem Besar* adalah sistem yang sebagian besar sumber dayanya berfungsi melakukan perawatan harian. Misalnya dinosaurus sebagai sistem biologi menghabiskan sebagian besar masa hidupnya dengan makan.

3. *Sistem Sebagai Bagian dari Sistem Lain*. Sistem selalu merupakan bagian dari sistem yang lebih besar, dan dapat terbagi menjadi sistem yang lebih kecil.
4. *Sistem Berkembang*. Walaupun tidak berlaku bagi semua sistem, hamper semua sistem selalu berkembang.

II.1.2. Karakteristik Sistem

Model umum sebuah sistem terdiri dari *input*, proses, dan *output*. Hal ini merupakan konsep sebuah sistem yang sangat sederhana mengingat sebuah sistem dapat mempunyai beberapa masukan dan keluaran sekaligus. Selain itu sebuah sistem juga memiliki karakteristik atau sifat-sifat tertentu, yang mencirikan bahwa hal tersebut bisa dikatakan sebagai suatu sistem (Sutabri ; 2012 : 13-14).

Adapun karakteristik yang dimaksud adalah sebagai berikut (Sutabri ; 2012 : 13-14):

1. Komponen Sistem (*Components*)

   Suatu sistem terdiri dari sejumlah komponen yang saling berinteraksi, yang bekerja sama membentuk satu kesatuan. Komponen-komponen sistem tersebut dapat berupa suatu bentuk subsistem.

2. Batasan Sistem (*Boundary*)

   Ruang lingkup sistem merupakan daerah yang membatasi antara sistem dengan sistem lainnya atau sistem dengan lingkungan luarnya. Batasan sistem ini memungkinkan suatu sistem dipandang sebagai satu kesatuan yang tidak dapat dipisah-pisahkan.

3. Lingkungan Luar Sistem (*Environtment*)

   Bentuk apapun yang ada diluar ruang lingkup atau batasan sistem yang mempengaruhi operasi sistem tersebut disebut dengan lingkungan luar sistem. Lingkungan luar sistem ini dapat menguntungkan dan dapat juga merugikan sistem tersebut.

4. Penghubung Sistem (*Interface*)

   Media yang menghubungkan sistem dengan subsistem yang lain disebut dangan penghubung sistem atau *interface*. Penghubung ini memungkinkan sumber-sumber daya mengalir dari satu subsistem ke subsistem lain. Keluaran suatu subsistem akan menjadi masukan untuk subsistem yang lain dengan melewati penghubung.

5. Masukan Sistem (*Input*)

   Energi yang dimasukkan ke dalam sistem disebut masukan sistem, yang dapat berupa pemeliharaan (*maintenance input*) dan sinyal *(signal input).*

6. Keluaran Sistem (*Output*)

   Hasil dari energi yang diolah dan diklasifikasikan menjadi keluaran yang berguna. Keluaran ini merupakan masukan bagi subsistem yang lain.

7. Pengolah Sistem (*Procces*)

   Suatu sistem dapat mempunyai suatu proses yang akan mengubah masukan menjadi keluaran.

8. Sasaran Sistem (*Objective*)

   Suatu sistem memiliki tujuan dan sasaran yang pasti dan bersifat deterministik. Suatu sistem dikatakan berhasil bila mengenai sasaran atau tujuan yang telah dirancanakan.

II.2. Sistem Pendukung Keputusan

Decision Support System atau sistem pendukung keputusan didefenisikan juga sebagai sebuah sistem yang mampu memberikan kemampuan, baik kemampuan pemecahan masalah maupun kemampuan pengkomunikasian untuk masalah semi tersetruktur. Secara khusus, SPK didefenisikan sebagai sebuah sistem yang mendukung kerja seseorang manager maupun sekelompok manager dalam memecahkan masalah semi tersetruktur dengan cara memberikan informasi ataupun usulan menuju pada keputusan tertentu.

Konsep DSS (sistem pendukung keputusan) pertama kali diperkenalkan pada awal tahun 1970-an oleh Michael Scott Morton, yang selanjutnya dengan istilah *"Management Decision System"*. Konsep DSS merupakan sebuah sistem interaktif berbasis komputer yang membantu pembuatan keputusan memanfaatkan data untuk menyelesaikan masalah-masalah yang bersifat tidak tersetruktur dan terstruktur. DSS dirancang untuk menunjang seluruh tahapan pembuatan keputusan, yang dimulai dari tahapan mengidentifikasi masalah, memilih data yang relevan, menentukan pendekatan yang digunakan dalam proses pembuatan keputusan sampai pada kegiatan mengevaluasi pemilihan alternative (Sherly ; 2013 : 43).

II.2.1. Komponen-Komponen Sistem Pendukung Keputusan

Menurut Hidayat (2015 : 44) komponen-komponen sistem pendukung keputusan adalah sebagai berikut:

a. Subsistem manajemen data (Hidayat ; 2015 : 44).

   Subsistem ini menyediakan data bagi sistem, termasuk didalamnya basis data. Berisi data yang relevan untuk situasi dan diatur oleh perangkat lunak yang disebut database management system (DBMS).

b. Subsistem manajemen model

   Subsistem ini berfungsi sebagai pengelola berbagai model, mulai dari model keuangan, statistik, matematik, atau model kuantitatif lainnya yang memiliki kemampuan analisis dan manajemen perangkat lunak yang sesuai. Perangkat lunak ini sering disebut Model Base Management System (MBMS).

c. Subsistem manajemen pengetahuan

   Subsistem ini mendukung berbagai subsistem lainnya, atau dapat dikatakan berperan sebagai komponen yang independen. subsistem ini menyediakan intelegensi untuk menambah pertimbangan pengambil keputusan.

d. Subsistem manajemen antar muka pengguna

   Subsistem ini berupa tampilan yang disediakan yang mampu mengintegrasikan sistem terpasang dengan pengguna secara interaktif. melalui subsistem ini pengguna dapat berkomunikasi dengan sistem pendukung keputusan serta memerintah sistem pendukung keputusan.

II.2.2. Jenis-Jenis Sistem Pendukung Keputusan

Keputusan-keputusan yang dibuat pada dasarnya dikelompokkan dalam 2 jenis, antara lain (Yunitarini ; 2013 : 45):

1. Keputusan Terperogram

   Keputusan ini bersifat berulang dan rutin, sedemikian hingga suatu prosedur pasti telah dibuat menanganinya sehingga keputusan tersebut tidak perlu diperlakukan *de bovo* (sebagai sesuatu yang baru) tiap kali terjadi.

2. Keputusan Tak Terprogram

   Keputusan ini bersifat baru, tidak tersetruktur jarang konsekuen. Tidak ada metode yang pasti untuk menangani masalah ini karena belum ada sebelumnya atau karena sifat dan struktur persisnya tak terlihat atau rumit atau karena begitu pentingnya sehingga memerlukan perlakuan yang sangat khusus.

II.3. Logika *Fuzzy*

Logika adalah ilmu yang mempelajari secara sistematis kaidah-kaidah penalaran yang absah (valid). Ada 2 konsep logika, yaitu logika tegas dan logika fuzzy. Logika tegas hanya mengenal dua keadaan yaitu: ya atau tidak, on atau off, high atau low, 1 atau 0.Logika semacam ini disebut dengan logika himpunan tegas. Sedangkan logika fuzzy adalah logika yang menggunakan konsep sifat kesamaran. Sehingga logika fuzzy adalah logika dengan tak hingga banyak nilai kebenaran yang dinyatakan dalam bilangan real dalam selang (0,1).

Logika fuzzy adalah suatu cara yang tepat untuk memetakan suatu ruang input kedalam suatu ruang output, mempunyai nilai kontinyu. Fuzzy dinyatakan dalam derajat dari suatu keanggotaan dan derajat dari kebenaran. Oleh sebab itu sesuatu dapat dikatakan sebagian benar dan sebagian salah pada waktu yang sama (Ula ; 2014 : 38-39).

II.3.1. Metode *Fuzzy Tsukamoto*

Sistem Inferensi Fuzzy merupakan suatu kerangka komputasi yang didasarkan pada teori himpunan fuzzy, aturan fuzzy berbentuk IF-THEN, dan penalaran fuzzy. Selama ini telah dikenal beberapa metode dalam FIS, seperti metode Tsukamoto, metode Mamdani dan metode Sugeno. Pada metode Tsukamoto, setiap konsekuen pada aturan yang berbentuk IFTHEN harus direpresentasikan dengan suatu himpunan fuzzy dengan fungsi keanggotaan monoton. Sebagai hasilnya, keluaran hasil inferensi dari tiap-tiap aturan diberikan secara tegas (*crisp*) berdasarkan α-predikat (*firestrength*). Hasil akhir menggunakan rata-rata terbobot (Ula ; 2014 : 41).

Secara umum bentuk model *fuzzy Tsukamoto* adalah (Ula ; 2014 ; 41) :

If (X IS A) and (Y IS B) Then (Z IS C)

Di mana A, B, dan C adalah himpunan *fuzzy*.

Misalkan diketahui 2 rule berikut.

IF (x is A_1) AND (y is B_1) THEN (z is C_1)

IF (x is A_2) AND (y is B_2) THEN (z is C_2)

Dalam inferensinya, metode *Tsukamoto* menggunakan tahapan berikut :

1) *Fuzzyfikasi*
2) Pembentukan basis pengetahuan *Fuzzy* (Rule dalam bentuk IF....THEN)
3) Mesin Inferensi

   Menggunakan fungsi implikasi MIN untuk mendapatkan nilai α-predikat tiap-tiap rule (α_1, α_2 , α_3,... α_n).

   Kemudian masing-masing nilai α-predikat ini digunakan untuk menghitung keluaran hasil inferensi secara tegas (*crisp*) masing-masing rule (z_1, z_2, z_3,.... z_n).
4) *Defuzzyfikasi*

   Menggunakan metode Rata-rata (*Average)*

$$z^* = \frac{\sum \alpha_i z_1}{\sum \alpha_i} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (1)$$

   Proses *DeFuzzyfikasi*

   Hasil akhir output (z) diperoleh dengan menggunakan rata-rata pembobotan :

$$z = \frac{\alpha_1 z_1 + \alpha_2 z_2}{\alpha_1 + \alpha_2} \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots (2)$$

II.4. Pengertian Basis Data (*Database*)

Basis data dapat dipahami sebagai suatu kumpulan data terhubung (*interrelated data*) yang disimpan secara bersama-sama pada suatu media, tanpa *mengatap* satu sama lain atau tidak perlu suatu kerangkapan data (kalaupun ada

maka kerangkapan data tersebut harus seminimal mungkin dan terkontrol [*controlled redudancy*]), data disimpan dengan cara-cara tertentu sehingga mudah digunakan/atau ditampilkan kembali; data dapat digunakan oleh satu atau lebih program-program aplikasi secara optimal; data disimpan tanpa mengalami ketergantungan dengan program yang akan menggunakannya; data disimpan sedemikian rupa sehingga proses penambahan, pengambilan, dan modifikasi data dapat dilakukan dengan mudah dan terkontrol (Sutanta ; 2011 : 29-30).

Berdasarkan tingkat kompleksitas nilai data, tingkatan data dapat disusun dalam sebuah hierarki, mulai dari yang paling sederhana hingga paling sederhana hingga paling kompleks (Sutanta ; 2011 : 35-36).

1. Sisem basis data, merupakan sekumpulan subsistem yang terdiri atas basis data dengan para pemakai yang menggunakan basis data secara bersama-sama, personal-personal yang merancang dan mengelola basis data, teknik-teknik untuk merancang dan mengelola basis data, serta sistem komputer untuk mendukungnya.
2. Basis data, merupakan sekumpulan dari bermacam-macam tipe *record* yang memiliki hubungan antar-*record* dan rincian data terhadap obyek tertentu.
3. File, merupakan sekumpulan *record* sejenis secara relasi yang tersimpan dalam media penyimpanan sekunder.
4. *Record,* merupakan *field/*atribut/data item yang saling berhubungan terhadap obyek tertentu.

5. *Data item/field/*atribut, merupakan unit terkecil yang disebut data, sekumpulan *byte* yang mempunyai makna.
6. *Data agregate*, merupakan sekumpulan data item/*field*/atribut dengan ciri tertentu dan diberi nama.
7. *Byte*, adalah bagian terkecil yang dialamatkan dalam memori. *Byte* merupakan sekumpulan *bit* yang secara konvensional terdiri atas kombinasi 8 *bit* biner yang menyatakan sebuah karakter dalam memori (1 *byte* = 1 karakter).
8. *Bit*, adalah sistem biner yang terdiri atas dua macam nilai, yaitu 0 dan 1. Sistem biner merupakan dasar yang dapat digunakan untuk komunikasi antara manusia dan mesin (komputer).

II.5. Normalisasi

Normalisasi diartikan sebagai suatu teknik yang menstrukurkan/ mendekomposisi data dalam cara-cara tertentu untuk mencegah timbulnya permasalahan pengolahan data dalam basis data. Permasalahan yang dimaksud adalah berkaitan dengan penyimpangan-penyimpangan *(anomalies)* yang terjadi akibat adanya kerangkapan data dalam relasi dan in-efisiensi pengolahan (Sutanta ; 2011 : 174).

Proses normalisasi menghasilkan relasi yang optimal, yaitu (Martin, 1975) : (Sutanta ; 2011 : 175)

1. Memiliki struktur *record* yang konsisten secara logik;
2. Memiliki struktur *record* yang mudah untuk dimengerti;

3. Memiliki struktur *record* yang sederhana dalam pemeliharaan;
4. Memiliki struktur *record* yang mudah ditampilkan kembali untuk memenuhi kebutuhan pengguna;
5. Minimalisasi kerangkapan data guna meningkatkan kinerja sistem.

Secara berturut-turut masing-masing level normal tersebut dibahas berikut ini, dimulai dari bentuk tidak normal. (Sutanta ; 2011 : 176-179)

1. Relasi bentuk tidak normal (*Un Normalized Form* / UNF)

   Relasi-relasi yang dirancang tanpa mengindahkan batasan dalam defisi basis data dan karakteristik *Relational Database Management System* (RDBM) menghasilkan relasi *Un Normalized Form* (UNF). Bentuk ini harus di hindari dalam perancangan relasi dalam basis data. Relasi *Un Normalized Form* (UNF) mempunyai kriteria sebagai berikut.

   a. Jika relasi mempunyai bentuk *non flat file* (dapat terjadi akibat data disimpan sesuai dengan kedatangannya, tidak memiliki struktur tertentu, terjadi duplikasi atau tidak lengkap)
   b. Jika relasi membuat *set atribut* berulang *(non single values)*
   c. Jika relasi membuat *atribut non atomic value*

2. Relasi bentuk normal pertama (*First Norm Form* / 1NF)

   Relasi disebut juga *First Norm Form* (1NF) jika memenuhi kriteria sebagai berikut.

   a. Jika seluruh atribut dalam relasi bernilai *atomic (atomic value)*
   b. Jika seluruh atribut dalam relasi bernilai tunggal *(single value)*

c. Jika relasi tidak memuat set atribut berulang

d. Jika semua record mempunyai sejumlah atribut yang sama.

Permasalahan dalam *First Norm Form* (1NF) adalah sebagai berikut.

a. Tidak dapat menyisipkan informasi parsial

b. Terhapusnya informasi ketika menghapus sebuah *record*

c. Pembaruan atribut nonkunci mengakibatkan sejumlah *record* harus diperbaharui

3. Bentuk normal kedua *(Second Normal Form /* 2NF)

Relasi disebut sebagai *Second Normal Form (*2NF) jika memenuhi kriteria sebagai berikut

a. Jika memenuhi kriteria *First Norm Form (*1NF)

b. Jika semua atribut nonkunci *Functional Dependence* (FD) pada *Primary Key* (PK)

Permasalahan dalam *Second Normal Form /* 2NF adalah sebagai berikut:

a. Kerangkapan data *(data redundancy)*

b. Pembaharuan yang tidak benar dapat menimbulkan inkonsistensi data *(data inconsistency)*

c. Proses pembaharuan data tidak efisien

d. Penyimpangan pada saat penyisipan, penghapusan, dan pembaruan.

Kriteria tersebut mengidentifikasikan bahwa antara atribut dalam *Second Normal Form* masih mungkin mengalami *Third Norm Form*. Selain itu, relasi *Second Normal Form* (2NF) menuntut telah

didefinisikan atribut *Primary Key* (PK) dalam relasi. Mengubah relasi *First Norm Form* (1NF) menjadi bentuk *Second Normal Form* (2NF) dapat dilakukan dengan mengubah struktur relasi dengan cara :

a. Identifikasikan *Functional Dependence* (FD) relasi *First Norm Form* (1NF)
b. Berdasarkan informasi tersebut, dekomposisi relasi *First Norm Form* (1NF) menjadi relasi-relasi baru sesuai *Functional Dependence* nya. Jika menggunakan diagram maka simpul-simpul yang berada pada puncak diagram ketergantungan data bertindak *Primary Key* (PK) pada relasi baru

4. Bentuk normal ketiga (*Third Norm Form* / 3NF)

   Suatu relasi disebut sebagai *Third Norm Form* jika memenuhi kriteria sebagai berikut.

   a. Jika memenuhi kriteria *Second Normal Form* (2NF)
   b. Jika setiap atribut nonkunci tidak (TDF) *(Non Transitive Dependeny)* terhadap *Primary Key* (PK)

   Permasalahan dalam *Third Norm Form* (3NF) adalah keberadaan penentu yang tidak merupakan bagian dari *Primary Key* (PK) menghasilkan duplikasi rinci data pada atribut yang berfungi sebagai *Foreign Key* (FK) (duplikasi berbeda dengan keterangan data).

   Mengubah relasi *Second Normal Form* (2NF) menjadi bentuk *Third Norm Form* (3NF) dapat dilakukan dengan mengubah struktur relasi dengan cara :

a. Identifikasi TDF relasi *Second Normal Form* (2NF)

b. Berdasarkan informasi tersebut, dekomposisi relasi *Second Normal Form* (2NF) menjadi relasi-relasi baru sesuai TDF-nya.

5. Bentuk normal *Boyce-Cood (Boyce-Codd Norm Form* / BCNF)

   Bentuk normal *Boyce-Codd Norm Form* (BCNF) dikemukakan oleh R.F. Boyce dan E.F. Codd. Suatu relasi disebut sebagai *Boyce-Codd Norm Form* (BCNF) jika memenuhi kriteria sebagai berikut.

   a. Jika memenuhi kriteria *Third Norm Form* (3NF)

   b. Jika semua atribut penentu (determinan) merupakan CK

6. Bentuk normal keempat (*Forth Norm Form* / 4NF)

   Relasi disebut sebagi *Forth Norm Form* (4NF) jika memenuhi kriteria sebagai berikut.

   a. Jika memenuhi kriteria *Boyce-Codd Norm Form*.

   b. Jika setiap atribut didalamnya tidak mengalami ketergantungan pada banyak nilai.

7. Bentuk normal kelima (*Fifth Norm Form* / 5NF)

   Suatu relasi memenuhi kriteria *Fifth Norm Form* (5NF) jika kerelasian antar data dalam relasi tersebut tidak dapat direkonstruksi dari struktur relasi yang sederhana.

8. Bentuk normal kunci domain (*Domain Key Norm Form* / DKNF)

a. Relasi disebut sebagai *Domain Key Norm Form* (DKNF) jika setiap batasan dapat disimpulkan secara sederhana dengan mengetahui

sekumpulan nama atribut dan domainnya selama menggunkan sekumpulan atribut pada kuncinya.

II.6. *Entity Relationship* Diagram (ERD)

Entity-Relationship Diagram (ERD) adalah suatu pemodelan konseptual yang didesain secara khusus untuk mengidentifikasikan entitas yang menjelaskan data dan hubungan antardata, yaitu dengan menuliskan dalam *cardinality*. Sementara seolah-olah teknik diagram atau alat peraga memberikan dasar untuk desain *database* relasional yang mendasari sistem informasi yang dikembangkan. ERD bersama-sama dengan detail pendukung merupakan model data yang pada gilirannya digunakan sebagai spesifikasi untuk *database*. Dalam sistem *Entity Relationship Diagram* (ERD), terdapat beberapa istilah penting diantaranya (Pratama dan Junianto ; 2015 : 215-216)

1. Entitas (*Entity*)

   Suatu entitas yang dapat berupa orang, tempat, obyek, atau kejadian yang dianggap penting bagi perusahaan, sehingga segala atributnya harus dicatat dan disimpan dalam basis data.

2. Atribut (*Atribute*)

   Setiap entitas mempunyai karakterisik tertentu yang dinamakan dengan atribut.

3. Relasi (*relationship*)

   Hubungan antara dua atau lebih entitas yang saling berkaitan. Menurut Romney (2009:596) ada tiga tipe relasi (*relationship*), yaitu:

a. *One-to-one relationship* (1:1)

Dimana maximum *cardinality* setiap *entity* adalah 1.

Contoh : Satu nasabah bank hanya memiliki satu *account*.

b. *One-to-many relationship* (1:N).

Dimana maximum *cardinality* dari suatu *entity* adalah 1 dan maximum *cardinality* dari *entity* lain adalah N.

Contoh : Satu nasabah bank dapat memiliki lebih dari satu *account.*

c. *Many-to-many relationship* (M:N).

Dimana maximum *cardinality* kedua *entity* yang berhubungan adalah N.

Contoh : Satu nasabah dapat memilliki beberapa *account* dan satu *account* dapat dimiliki oleh beberapa nasabah (rekening bersama).

4. *Identifier*

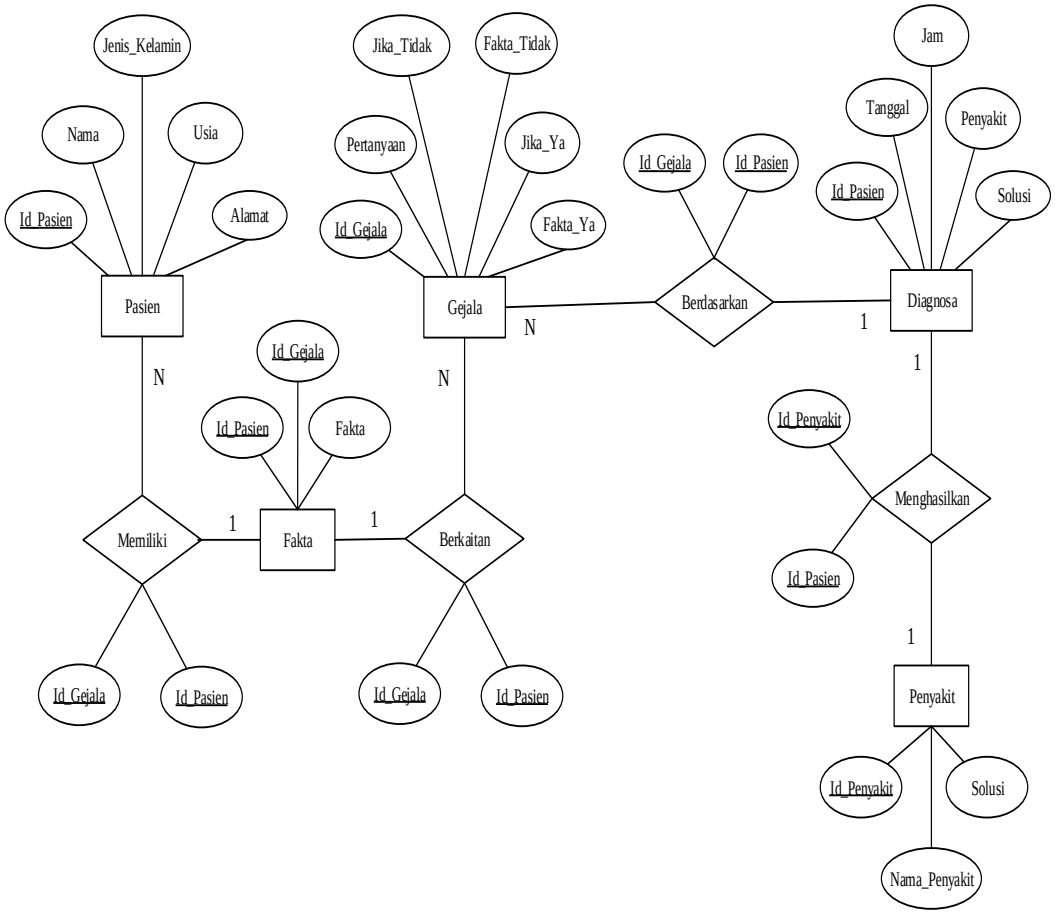
Merupakan nama *atribute* yang digunakan untuk mengidentifikasi suatu entitas, Ada tiga jenis *identifier* diantaranya *Primary Key*, *Secondary Key*, dan *Foreign Key*. Berikut ini penjelasan dari *Primary Key* dan *Foreign Key*:

a. *Primary Key* merupakan suatu kode identifikasi yang bersifat unik yang ditunjukkan oleh masing-masing *record* dalam sistem. Tujuan dari *Primary key* adalah untuk menunjukkan lokasi tiap catatan di dalam suatu *file* mengenai catatan-catatan serupa.

b. *Foreign Key* merupakan *attribute* yang merupakan *Primary key* dari relasi lain yang ditarik/dihubungkan ke suatu relasi.

5. Kardinalitas (*Cardinality*)

   Merupakan kendala-kendala yang timbul dalam hubungan antar entitas.



Gambar II.1. *Entity Relationship Diagram* (ERD)

(Sumber : (Pratama dan Junianto ; 2015 : 215-216)

II.7. Unified Modelling Language (UML)

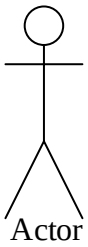
Unifed Modeling Language adalah sebuah bahasa yang diterima dan digunakan oleh developer dan *software analyst* sebagai suatu bahasa yang cocok untuk merepresentasikan grafi darisuatu relasi antar entitas-entitas *software*. Dengan menggunakan UML, tim pengembang *software* akan mempunyai banyak keuntungan, seperti memudahkan komunikasi dengan sesama anggota tim tentang *software* apa yang akan dibuat, memudahkan integrasi ke dalam area pengerjaan *software* karena bahasa ini berbasiskan meta-models dimana meta-models bisa mendefinisikan proses-proses untuk mengkonstruksikan konsep-konsep yang ada. UML juga menggunakan formatur *input* dan *output* yang sudah mempunyai bentuk standar yaitu XML Metadata *Interchange* (XMI), menggunakan aplikasi dan pemodelan data yang universal, merepresentasikan dari tahap analisis ke implementasi lalu ke *deployment* yang terpadu, dan mendeskripsikan keutuhan tentang spesifiasi *software.*

UML menyediakan kumpulan alat yang sudah terstandarisasi, yang digunakan untuk mendokumentasikan analisis dan perancangan sebuah sistem perangkat lunak. (Kendall & Kendall, 2005, p663) Peralatan utama UML adalah diagram-diagram yang digunakan untuk membantu manusia dalam memvisualisasikan proses pengembangan sebuah sistem perangkat lunak, sama seperti penggunaan denah (blueprint) dalam pembuatan bangunan (Winata dan Setiawan ; 2013 : 37).

II.7.1. *Use Case Model*

Use case model adalah teknik pemodelan untuk mendapatkan *functional requirement* dari sebuah sistem, menggambarkan interaksi antara pengguna dan sistem, menjelaskan secara naratif bagaimana sistem akan digunakan, menggunakan skenario untuk menjelaskan setiap aktivitas yang mungkin terjadi. Ada beberapa bagian didalam *use case model* (Winata dan Setiawan ; 2013 : 38).

Gambar II.2 *Use Case* Pada *Use Case Diagram*
(Sumber : Winata dan Setiawan ; 2013 : 38)



Gambar II.3. *Actor* Pada *Use Case Diagram*
(Sumber : Winata dan Setiawan; 2013 : 38)

a. Use Case, untuk mengetahui action atau prosedur apa yang ada didalam sistem.
b. Actor, siapa saja yang terlibat dalam action tersebut.
c. Relationship, bagaimana actions saling berelasi satu sama lain didalam sistem.

II.7.2. *Class Diagram*

Class diagram merupakan diagram paling umum yang dijumpai dalam pemodelan berbasis UML. Didalam *Class diagram* terdapat class dan interface

beserta atribut-atribut dan operasinya, relasi yang terjadi antar objek, *constraint* terhadap objek-objek yang saling berhubungan dan *inheritance* untuk organisasi class yang lebih baik. *Class diagram* juga terdapat *static view* dari elemen pembangun sistem. Pada intinya *Class diagram* mampu membantu proses pembuatan sistem dengan memanfaatkan konsep *forward* ataupun *reverse engineering* (*Rational Software Corporation*, 1997).

Class diagram mempunyai 2 komponen penting, yaitu:

a. *Structural*, yaitu ciri pembeda objek.
b. *Behavioral*, yaitu tingkah laku atau kegiatan yang mampu dilakukan oleh objek.

Berbagai simbol yang hadir didalam *class diagram*

a. *Class*, yang berfungsi untuk merepresentasikan tipe dari data yang dimilikinya. Class diagram dapat ditampilkan dengan menunjukkan atribut dan operasi yang dimilikinya atau hanya menunjukkan nama class-nya saja. Dapat juga kita tuliskan nama class dengan atributnya saja atau nama class dengan operasinya.
b. *Attribute*, merupakan data yang terdapat didalam class dan instance-nya dengan operator.
c. *Operation*, berfungsi untuk merepresentasikan fungsi-fungsi yang ditampilkan oleh class dan instance-nya dengan operator.
d. *Association*, digunakan untuk menunjukkan bagaimana dua class berhubungan satu sama lainnya. Association ditunjukkan dengan sebuah garis yang terletak diantara dua class. Didalam setiap

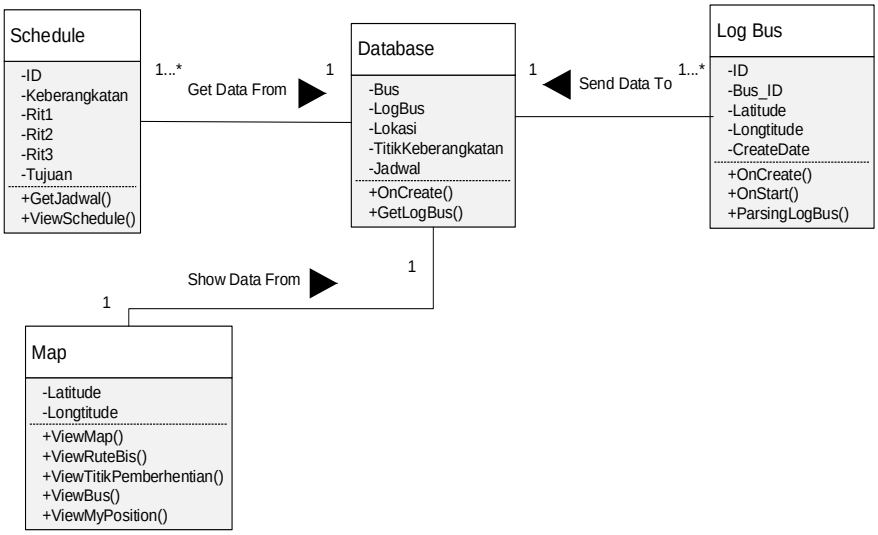
association terdapat multiplicity, yaitu simbol yang mengindikasikan berapa banyak instance dari class pada ujung association yang satu dengan instance class di ujung association lainnya.

e. *Generalizations*, berfungsi untuk mengelompokkan class ke dalam hirarki inheritance.

f. *Aggregation*, merupakan bentuk khusus dari association yang merepresentasikan hubungan "part-whole". Bagian "whole" dari hubungan ini sering disebut dengan assembly atau aggregate. Class yang satu dapat dikatakan merupakan bagian dari class yang lain yang ikut membentuk class tersebut.

g. *Composition*, merupakan jenis aggregation yang lebih kuat diantara dua class yang memiliki association dimana jika *whole* ditiadakan, maka *part*-nya juga ikut ditiadakan. Berbeda dengan aggregation, *part* akan tetap bisa berdiri sendiri meskipun bagian *whole*-nya ditiadakan.

h. Penggunaan operator (+) dalam class diagram diartikan dengan public, operator (-) diartikan private, dan operator (#) diartikan protected.



Gambar II.4. Contoh *Class Diagram*

(Sumber : Winata dan Setiawan ; 2013 : 43)

Class diagram menggambarkan karakteristik statis sistem tanpa menjelaskan proses secara mendetil. Sebuah class diagram juga memperlihatkan hubungan antar class (Winata dan Setiawan; 2013 : 38-39).

II.7.3. *Sequence Diagram*

Menjelaskan interaksi obyek-obyek yang saling berkolaborasi (berhubungan), mirip dengan *activity diagram* yaitu menggambarkan alur kejadian sebuah aktivitas tetapi lebih detil dalam menggambarkan aliran data termasuk data atau behaviour yang dikirimkan atau diterima namun kurang mampu menjelaskan detil dari sebuah algoritma.

Dalam *sequence diagram* terdapat beberapa bagian.

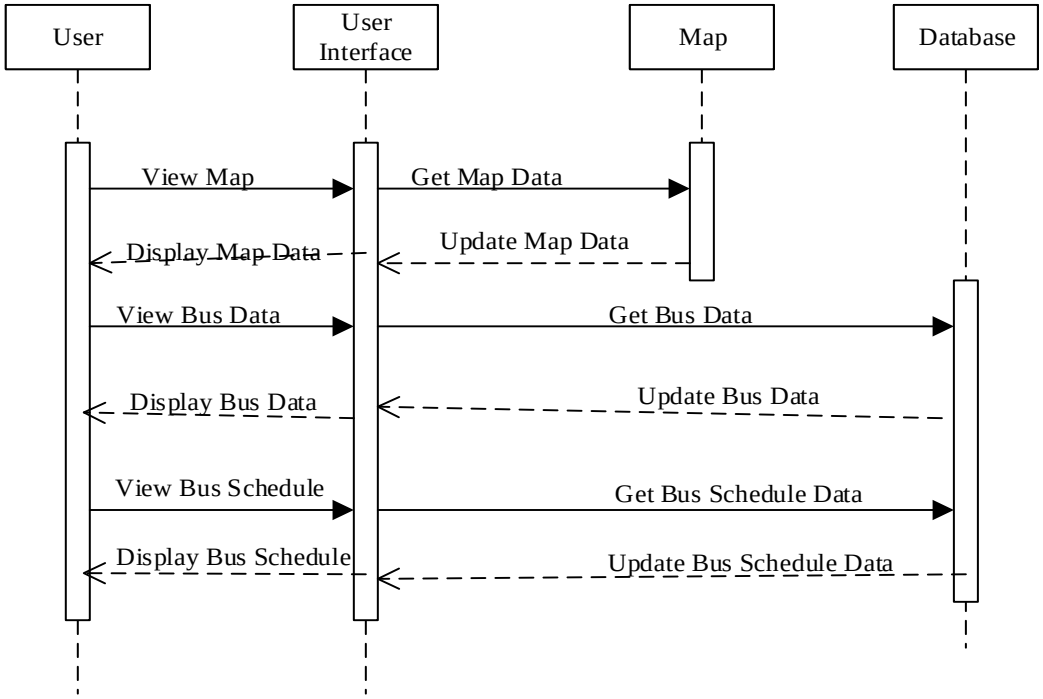
a. *Participant*, yaitu objek yang terkait dengan sebuah urutan proses.

b. *Lifeline*, menggambarkan daur hidup sebuah objek.

c. *Activation*, suatu titik waktu dimana sebuah objek mulai berpartisipasi dalam sebuah sequence.

d. *Time*, elemen paling penting dalam sequence diagram yang konteksnya adalah urutan, bukan durasi.

e. *Return*, suatu hasil kembalian sebuah operasi. Operasi mengembalikan hasil tetapi boleh tidak ditulis jika tidak ada perbedaan dengan Getternya.



Gambar II.5. Contoh *Squence Diagram*

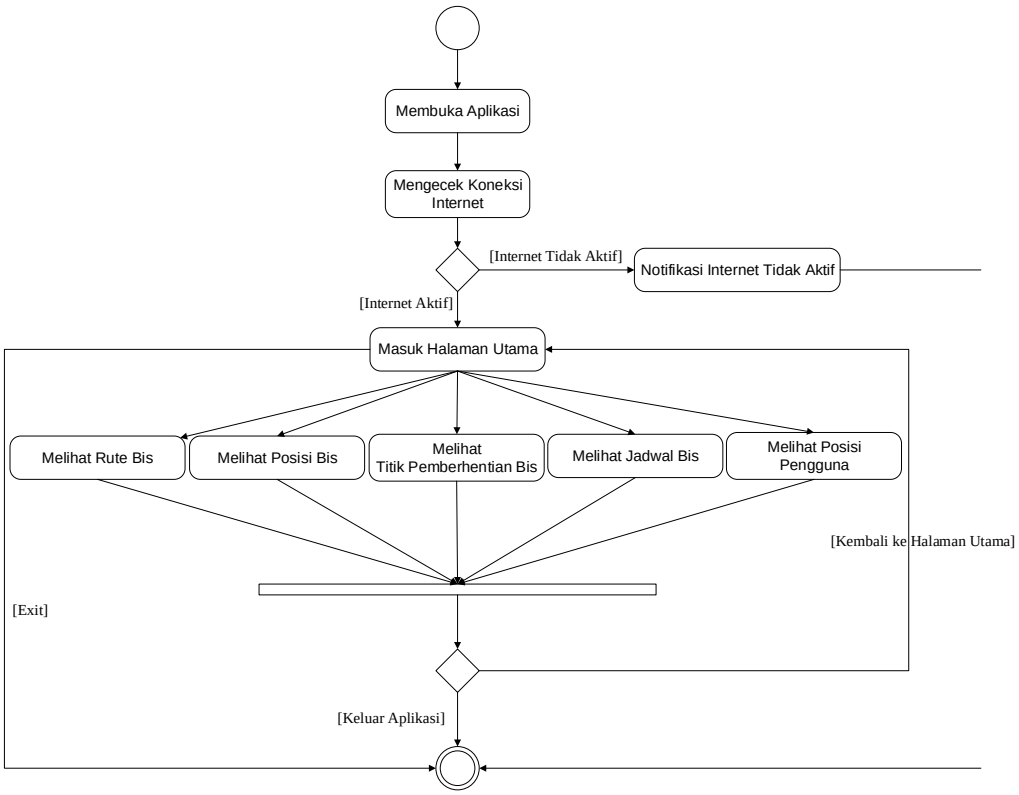
(Sumber : Winata dan Setiawan ; 2013 : 38)

Dalam penggunaannya, *sequence diagram* tepat untuk memperlihatkan dengan jelas bagaimana urutan kejadian suatu proses karena didalamnya terlihat interaksi beberapa objek (Winata dan Setiawan; 2013 : 39).

II.7.4. Activity Diagram

Teknik untuk menjelaskan *business process*, *procedural logic*, dan *work flw*. Bisa dipakai untuk menjelaskan teks use case dalam notasi grafik dengan menggunakan notasi yang mirip *flw chart*, meskipun terdapat sedikit perbedaan notasi (Winata dan Setiawan ; 2013 : 39).

a. *Nodes*, menandakan initial dan fial node, fial node boleh lebih dari 1.
b. *Activity*, aktivitas sistem dapat berupa aktivitas fiik juga bagi user.
c. *Flow/edge*, arah sebuah proses.
d. *Fork*, awal sebuah proses paralel.
e. *Join* akhir proses paralel.
f. *Condition*, kondisi yang dituliskan dalam bentuk teks
g. *Decision*, implementasi if dan then.
h. *Merge*, penyatuan beberapa flw.
i. *Partition*, siapa atau apa yang menjalankan aktivitas.



Gambar II.6. Contoh *Activity Diagram*
(Sumber : Winata dan Setiawan ; 2013 : 39)

II.8. PHP *(Hypertext Preprocessor)*

PHP adalah bahasa server-side scripting yang menyatu dengan HTML untuk membuat halaman web yang dinamis. Karena PHP merupakan server-side scripting maka sintaks dan perintah-perintah PHP akan dieksekusi di server kemudian hasilnya dikirimkan ke web browser dalam format HTML. Dengan demikian kode program yang ditulis dalam PHP tidak akan terlihat oleh user sehingga keamanan halaman web lebih terjamin. PHP dirancang untuk membentuk halaman web dinamis, yaitu halaman web yang dapat membentuk

suatu tampilan berdasarkan permintaan terkini, seperti menampilkan isi basis data ke halaman web.

PHP termasuk dalam Open Source Product, sehingga source code PHP dapat diubah dan didistribusikan secara bebas. PHP juga dapat berjalan pada berbagai web server seperti ISS (Internet Information Server), PWS (Personal Web Server), Apache, Xitami. PHP juga mampu lintas platform. Artinya PHP dapat berjalan di banyak sistem operasi yang beredar saat ini, di antaranya: Sistem Operasi Microsoft Windows (semua versi), Linux, Mac OS, Solaris. PHP dapat dibangun sebagai modul pada web server Apache dan sebagai binary yang dapat berjalan sebagai CGI (Common Gateway Interface). PHP dapat mengirim HTTP header, dapat mengatur cookies, mengatur outhentication dan redirect users.

Salah satu keunggulan yang dimiliki oleh PHP adalah kemampuannya untuk melakukan koneksi ke berbagai macam software sistem manajemen basis data/Database Management System (DBMS), sehingga dapat menciptakan suatu halaman web yang dinamis. PHP mempunyai koneksitas yang baik dengan beberapa DBMS antara lain Oracle, Sybase, mSql, MySQL, Microsoft SQL Server, Solid, PostgreSQL, Adabas, FilePro, Velocis, dBase, Unix dbm, dan tak terkecuali semua database ber-interface ODBC (Arief, 2011: 43).

II.9. MySQL

MySQL adalah salah satu jenis *database server* yang sangat terkenal dan banyak digunakan untuk membangun aplikasi web yang menggunakan *database* sebagai sumber dan pengolahan datanya. Kepopuleran MySQL antara lain karena

MySQL menggunakan SQL sebagai bahasa dasar untuk mengakses *database*-nya sehingga mudah untuk digunakan, kinerja *query* cepat, dan mencukupi untuk kebutuhan *database* perusahaan-perusahaan skala menengah-kecil. MySQL juga bersifat *open source* dan *free* pada berbagai *platform* (kecuali pada windows yang bersifat *shareware*). MySQL didistribusikan dengan lisensi *open source* GPL (*General Public License*) mulai versi 3.23, pada bulan Juni 2000.

MySQL merupakan *database* yang pertama kali didukung oleh bahasa pemrograman *script* untuk internet (PHP dan Perl). MySQL dan PHP dianggap sebagai pasangan *software* pengembangan aplikasi web yang ideal. MySQL lebih sering digunakan untuk membangun aplikasi berbasis web, umumnya pengembangan aplikasinya menggunakan bahasa pemrograman *script* PHP (Arief, 2011: 151).