BAB 2

TINJAUAN PUSTAKA

2.1 Profil Tempat Kerja Praktek

Pt. *Technomed Asia* beralamat di Jalan Tembokan RT 01/01 Padalarang Bandung 40553 .Telp. 022 – 6868424, 6868423, 2. Fax 022 – 6868420.

2.1.1 Sejarah Instansi

Pt. *Technomed Asia* diresmikan pada Bulan Maret 2005 tepatnya pada tanggal 27 maret 2005. Diresmikan langsung oleh Menkes pada saat itu, yaitu Siti Fadilah Supari selaku menteri kesehatan.

Pt. *Technomed Asia* merupakan sebuah perusahaan Joint Venture Indonesia-Belanda. Kegiatan utamanya adalah bergerak di bidang produksi alat kesehatan yang mencakup beberapa divisi.

Produk yang dihasilkan oleh perusahaan ini terdiri dari beberapa jenis antara lain:

- *Medical soft bag*

  Soft bag adalah kantung infus yang digunakan oleh perusahaan farmasi untuk mengemas cairan infus. Target pasaranya adalah pasar domestik
- *Neuro Diagnostic Accessoris*

  a. Disposable Subdermal Needle Electrode

  Electrode untuk perekam dan stimulasi EMG (*electromyography*)

  b. Disposable Concentric Needle Elcetrode

  Electrode untuk perekam motorik otot.

2.1.2 Logo Instansi

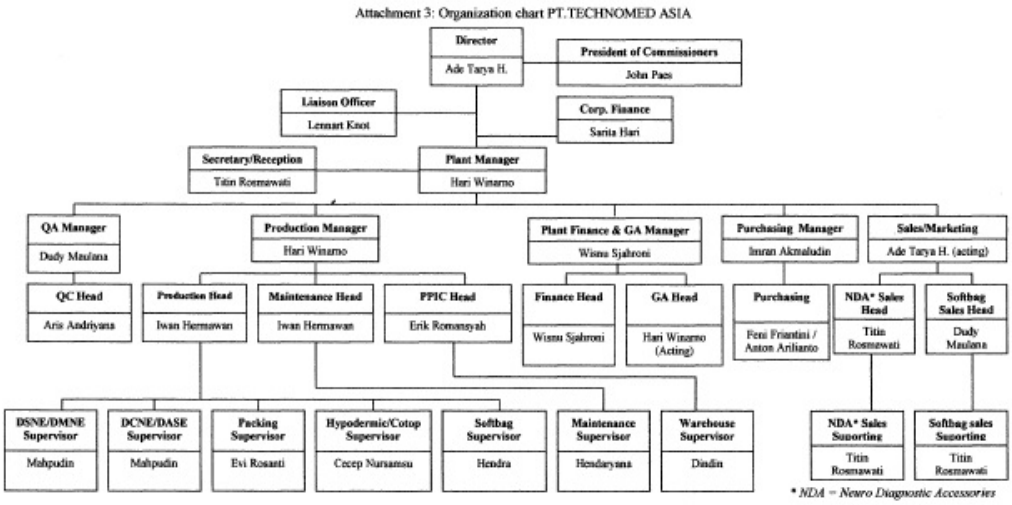
Berikut gambar logo Pt. *Technomed Asia* :

Technomed Asia
Medical Accessories

Gambar 2.1.1 Logo Pt. *Technomed Asia*

2.1.3 Struktur Organisasi dan Job Description

Berikut gambar Struktur Organisasi Pt. *Technomed Asia*:



Gambar 2.1.2 Struktur Organisasi Pt. *Technomed Asia*

2.2 Landasan Teori

Selama pelaksanaan kerja praktek di Pt. *Technomed Asia*, saya menggunakan pengetahuan yang diperoleh selama masa perkuliahan sebagai landasan teori dalam pengembangan aplikasi dan beberapa pengetahuan baru yang didapat pada saat pelaksanaan kerja praktek berlangsung.

2.2.1 Pemrograman Berorientasi Objek

Adalah suatu paradigma yang menggunakan objek dengan identitas yang membungkus propertis dan operasi, melewatkan pesan, dan inheritance atau pewarisan untuk menyelesaikan domain permasalahan [2].

Object Oriented System adalah Sebuah sistem yang dibangun berdasarkan metode berorientasi objek. Adapun konsep dasar PBO adalah sebagai berikut :

2.2.1.1 Class

Class mengizinkan kita dalam mendeklarasikan tipe *data* baru. Ia dijalankan sebagai *blueprint*, dimana model dari *object* yang Anda buat berdasarkan pada tipe *data* baru ini [2].

2.2.1.2 Object

Sebuah *object* adalah sebuah entiti yang memiliki keadaan, *behaviour* dan identitas yang yang tugasnya dirumuskan dalam suatu lingkup masalah dengan baik. Inilah instance sebenarnya dari sebuah *class* [2]. Ini juga dikenal sebagai *instance*. *Instance* dibuat sewaktu Anda meng-*instantiate class* menggunakan kata kunci *new*. Dalam sistem registrasi siswa, contoh dari sebuah *object* yaitu entiti Student.

2.2.1.3 Atribut

Atribut menunjuk pada elemen *data* dari sebuah object. Atribut menyimpan informasi tentang object. Dikenal juga sebagai member *data*, variabel instance, properti atau sebuah field *data[2]*.

2.2.1.4 Method

Sebuah method menjelaskan behaviour dari sebuah object. Method juga dikenal sebagai fungsi atau prosedur. Sebagai contoh, method yang mungkin tersedia untuk entiti siswa adalah method register[2].

2.2.1.5 Konstruktor

Konstruktor adalah sebuah tipe khusus dari method yang digunakan untuk membuat dan menginisialisasi sebuah *object* baru. Ingat bahwa konstruktor bukan member (yaitu atribut, method atau inner *class* dari sebuah object) [2].

2.2.1.6 Package

Package menunjuk pada pengelompokkan *class* dan/atau subpackages. Strukturnya dapat disamakan dengan direktorinya[2].

2.2.1.7 Enkapsulasi

Enkapsulasi menunjuk pada prinsip dari menyembunyikan desain atau mengimplementasikan informasi yang tidak sesuai pada *object* yang ada[2].

2.2.1.8 Abstaraksi

enkapsulasi menyembunyikan detail, abstraksi mengabaikan aspek dari subyek yang tidak sesuai dengan tujuan yang ada supaya lebih banyak mengkonsentrasikan yang ada[2].

2.2.1.9 Pewarisan atau Inheritance

Pewarisan adalah hubungan antara *class* dimana dalam satu *class* ada super*class* atau *class* induk dari *class* yang lain. Pewarisan menunjuk pada properti dan behaviour yang diterima dari nenek moyang dari *class*. Ini dikenal juga sebagai hubungan "is-a" [2].

2.2.1.10 Polimorphisme

Polimorfisme adalah kemampuan dari sebuah *object* untuk membolehkan mengambil beberapa bentuk yang berbeda. Secara harfiah, "poli" berarti banyak sementara "morph" berarti bentuk. Menunjuk pada contoh sebelumnya pada pewarisan, kita lihat bahwa *object SuperHero* dapat juga menjadi *object FlyingSuperHero* atau *object UnderwaterSuperHero* [2].

2.2.1.11 Interface

Sebuah interface adalah sebuah *contract* dalam bentuk kumpulan method dan deklarasi konstanta. Ketika sebuah *class implements* sebuah interface, ini mengimplementasikan semua method yang dideklarasikan dalam interface [2].

2.2.2 Struktur Program *Java*

Pada bagian ini meringkaskan *syntax* dasar yang digunakan dalam pembuatan aplikasi *Java*.

2.2.2.1 Mendeklarasikan *Class Java*

<*class*Declaration> ::=
<*modifier*> *class* <name> {
<attributeDeclaration>*
<constructorDeclaration>*

<methodDeclaration>*
}

dimana <*modifier*> adalah sebuah *access modifier,* yang mana boleh dikombinasikan dengan tipe yang lain dari *modifier*.

Contoh berikut ini mendeklarasikan blueprint *SuperHero.*

```
Class SuperHero {
String superPowers[];
void setSuperPowers(String superPowers[]) {
this.superPowers = superPowers;
}
void printSuperPowers() {
for (int i = 0; i < superPowers.length; i++) {
System.out.println(superPowers[i]);
}
}
```

}

2.2.2.2 Mendeklarasikan Atribut

<attributeDeclaration> ::=
<*modifier*> <type> <name> [= <default_value>];
<type> ::=
byte | short | int | long | char | float | double | boolean
| <*class*>

Petunjuk Penulisan Program:

[] = Menunjukkan bahwa bagian ini hanya pilihan.

Inilah contohnya.

```
public class AttributeDemo {
private String studNum;
public boolean graduating = false;
protected float unitsTaken = 0.0f;
```

String college;
}

2.2.2.3 Mendeklarasikan Method

<methodDeclaration> ::=
<*modifier*> <returnType> <name>(<parameter>*) {
<statement>*
}
<parameter> ::=
<parameter_type> <parameter_name>[,]

Sebagai contoh:
class MethodDemo {
int *data*;
int get*Data*() {
return *data*;
}
void set*Data*(int *data*) {
this.*data* = *data*;
}
void setMax*Data*(int *data*1, int *data*2) {
data = (*data*1>*data*2)? *data*1 : *data*2;
}
}

2.2.2.4 Mendeklarasikan sebuah Konstruktor

<constructorDeclaration> ::=
<*modifier*> <*class*Name> (<parameter>*) {
<statement>*
}

Jika tidak ada konstruktor yang disediakan secara jelas, konstruktor default secara otomatis
membuatkannya untuk Anda. Konstruktor default tidak membawa argumen dan tidak berisi pernyataan pada tubuh *class*.

Petunjuk Penulisan Program:
Nama konstruktor harus sama dengan nama class.
<modifier> yang valid untuk konstruktor adalah public, protected, dan private.
Konstruktor tidak memiliki nilai return.
Perhatikan contoh berikut.

```
class ConstructorDemo {
private int data;
public ConstructorDemo() {
data = 100;
}
ConstructorDemo(int data) {
this.data = data;
}
}
```

2.2.2.5 Menginstansiasi sebuah *Class*

Untuk meng-*instantiate* sebuah *class*, dengan sederhana kita gunakan kata kunci *new* diikuti dengan pemanggilan sebuah konstruktor. Mari lihat langsung ke contohnya.

```
class ConstructObj {
int data;
ConstructObj() {
/* menginisialisasi data */
}
public static void main(String args[]) {
```

ConstructObj obj = new ConstructObj(); //di-*instantiate*
}
}

2.2.2.6 Mengakses anggota object

Untuk mengakses anggota dari sebuah object, kita gunakan notasi "dot". Penggunaanya seperti
berikut:
<object>.<member>
Contoh selanjutnya berdasar pada sebelumnya dengan pernyataan tambahan untuk mengakses
anggota dan method tambahan.

```
class ConstructObj {
int data;
ConstructObj() {
/* inisialisasi data */
}
void setData(int data) {
this.data = data;
}
public static void main(String args[]) {
ConstructObj obj = new ConstructObj(); //instantiation
obj.setData = 10; //access setData()
System.out.println(obj.data); //access data
}
}
```

2.2.2.7 Package

Untuk menunjukkan bahwa *file* asal termasuk package khusus, kita gunakan *syntax* berikut:
<packageDeclaration> ::=

package <packageName>;

Untuk mengimpor package lain, kita gunakan *syntax* berikut:

<importDeclaration> ::=

import <packageName.elementAccessed>;

Dengan ini, *source code* Anda harus memiliki format berikut:

[<packageDeclaration>]

<importDeclaration>*

<*class*Declaration>+

Pengenalan Pemrograman 2 6

J.E.N.I.

Petunjuk Penulisan Program:

+ menunjukkan bahwa boleh ada 1 atau lebih kejadian pada baris ini dalam pengaplikasiannya.

Sebagai contoh.

package registration.*report*s;

import registration.processing.*;

import *java*.util.List;

import *java*.lang.*; //imported by default

class My*Class* {

/* rincian dari My*Class* */

2.2.2.8 Access *Modifier*

Tabel berikut meringkas access modoifier dalam *java*

	Private	Default/package	Protected	Public
Class yang sama	Yes	Yes	Yes	Yes
Package yang sama		Yes	Yes	Yes
Package yang berbeda(sub*class*)			Yes	Yes
Package yang berbeda(non				Yes

sub*class*)				

Tabel 2.2.2.1 Access *Modifier*

2.2.2.9 Enkapsulasi

Menyembunyikan elemen dari penggunaan sebuah *class* dapat dilakukan dengan pembuatan anggota yang ingin Anda sembunyikan secara private. Contoh berikut menyembunyikan field *secret*. Catatan bahwa field ini tidak langsung diakses oleh program lain menggunakan method getter dan setter[3].

```
class Encapsulation {
private int secret; //field tersembunyi
public boolean setSecret(int secret) {
if (secret < 1 || secret > 100) {
return false;
}
this.secret = secret;
return true;
}
public getSecret() {
return secret;
}
}
```

2.2.2.10 Pewarisan

Untuk membuat *class* anak atau sub*class* berdasarkan *class* yang telah ada, kita gunakan kata kunci *extend* dalam mendeklarasikan *class[3]*. Sebuah *class* hanya dapat meng-*extend* satu *class* induk.

Sebagai contoh, *class Point* di bawah ini adalah super*class* dari *class ColoredPoint.*

```
import java.awt.*;
class Point {
int x;
int y;
}
class ColoredPoint extends Point {
Color color;
}
```

2.2.2.11 Overriding Method

Overriding adalah penggunaan method dari suatu parent/super *class* (Inheretance) namun memiliki isi method yang berbeda dari method aslinya.

Contoh overriding method

```
public class OverRide extends Nama{
   @Override
   void setnama(String namadepan, String namabelakang){
      nama = namabelakang+" "+namadepan;
   }
}

public static void main(String[] args) {Nama nama1=new Nama();
      OverRide nama3=new OverRide();
      nama3.setnama("Christ");
      System.out.println("nama kecil = "+nama3.getnama());
```

```
        OverRide nama4=new OverRide();
        nama4.setnama("Christian", "Ronaldo");
        System.out.println("nama lengkap = "+nama4.getnama());

}
```

2.2.2.12 Abstract *Class* dam Method

Bentuk umum dari sebuah method *abstract* adalah sebagai berikut:

abstract <*modifier*> <returnType> <name>(<parameter>*);
Sebuah *class* yang berisi method *abstract* harus dideklarasikan sebagai sebuah *class abstract.*
abstract <*modifier*> <returnType> <name>(<parameter>*);abstract *class*
<name> {
/* constructors, fields and methods */
}

Kata kunci tidak dapat digunakan pada konstruktor atau method *static*. Ini juga penting untuk diingat bahwa *class abstract* tidak dapat di-*instantiate*. *Class* yang meng-*extends* sebuah *class abstract* harus mengimplementasikan semua method *abstract.* Jika tidak sub*class* sendiri dapat dideklarasikan sebagai *abstract.*

Petunjuk Penulisan Program:
catatan bahwa mendeklarasikan sebuah method abstract hampir mirip dalam mendeklarasikan class normal kecuali itu suatu method abstract yang tidak memiliki tubuh dan kepala sehingga dengan segera diakhiri dengan semicolon(;).

Sebagai contoh:

```
abstract class SuperHero {
String superPowers[];
void setSuperPowers(String superPowers[]) {
this.superPowers = superPowers;
}
void printSuperPowers() {
for (int i = 0; i < superPowers.length; i++) {
System.out.println(superPowers[i]);
}
}
abstract void displayPower();
}
class UnderwaterSuperHero extends SuperHero {
void displayPower() {
System.out.println("Communicate with sea creatures...");
System.out.println("Fast swimming ability...");
}
}
class FlyingSuperHero extends SuperHero {
void displayPower() {
System.out.println("Fly...");
}
}
```

2.2.2.13 Interface

Mendeklarasikan sebuah interface pada dasarnya mendeklarasikan sebuah *class* tetapi sebagai

penggantinya menggunakan kata kunci *class*, kata kunci *interface* digunakan. Berikut *syntax*nya.

<interfaceDeclaration> ::=
<*modifier*> interface <name> {
<attributeDeclaration>*
[<*modifier*> <returnType> <name>(<parameter>*);]*
}
Anggotanya adalah *public* ketika interface dideklarasikan *public.*

Class mengimplementasikan sebuah interface yang telah ada dengan menggunakan kata kunci *implements* [3]. *Class* ini dibuat untuk mengimplementasikan semua method interface. Sebuah *class* boleh mengimplementasikan lebih dari satu interface.
Contoh berikut menunjukkan bagaimana mendeklarasikan dan menggunakan sebuah interface.

```
interface MyInterface {
void iMethod();
}
class MyClass1 implements MyInterface {
public void iMethod() {
System.out.println("Interface method.");
}
void myMethod() {
System.out.println("Another method.");
}
}
class MyClass2 implements MyInterface {
public void iMethod() {
System.out.println("Another implementation.");
```

```
}
}
class InterfaceDemo {
public static void main(String args[]) {
MyClass1 mc1 = new MyClass1();
MyClass2 mc2 = new MyClass2();
mc1.iMethod();
mc1.myMethod();
mc2.iMethod();
}
}
```

2.2.2.14 Key*word* This

Kata kunci *this* dapat digunakan untuk beberapa alasan berikut:

1. Adanya ambigu pada atribut lokal dari variabel lokal
2. Menunjuk pada *object* yang meminta method non-static
3. Menunjuk pada konstruktor lain.

Sebagai contoh pada maksud pertama, perhatikan kode berikut dimana variabel *data* disediakan sebagai sebuah atribut dan parameter lokal pada saat yang sama.

```
class ThisDemo1 {
int data;
void method(int data) {
this.data = data;
/* this.data menunjuk ke atribut
sementara data menunjuk ke variabel lokal */
}
}
```

Contoh berikut menunjukkan bagaimana *object this* secara mutlak menunjuk ketika anggota nonstatic
dipanggil.

```
class ThisDemo2 {
int data;
void method() {
System.out.println(data); //this.data
}
void method2() {
method(); //this.method();
}
```

Sebelum melihat ke contoh yang lain, mari pertama meninjau pengertian method overloading. Konstruktor seperti juga method dapat juga menjadi overload. Method yang berbeda dalam *class* dapat memberi nama yang sama asalkan list parameter juga berbeda. Method overloaded harus berbeda dalam nomor dan/atau tipe dari parameternya. Contoh selanjutnya memiliki konstruktor overloaded dan referensi *this* yang dapat digunakan untuk menunjuk versi lain dari konstruktor.

```
class ThisDemo3 {
int data;
ThisDemo3() {
this(100);
}
ThisDemo3(int data) {
this.data = data;
}
}
```

2.2.2.15 Key*word* Super

Penggunaan kata kunci *super* berhubungan dengan pewarisan. *Super* digunakan untuk meminta konstruktor super*class*. *Super* juga dapat digunakan seperti kata kunci *this* [3]. untuk menunjuk pada anggota dari super*class*. Program berikut mendemonstrasikan bagaimana referensi *super* digunakan untuk memanggil konstruktor super*class*.

```
class Person {
String firstName;
String lastName;
Person(String fname, String lname) {
firstName = fname;
lastName = lname;
}
}
class Student extends Person {
String studNum;
Student(String fname, String lname, String sNum) {
Pengenalan Pemrograman 2 12
J.E.N.I.
super(fname, lname);
studNum = sNum;
}
}
```

Petunjuk Penulisan Program:

super() menunjuk pada superclass dengan segera. Ini harus berada pada pernyataan pertama
dalam konstruktor superclass.

Kata kunci dapat juga digunakan untuk menunjuk anggota
super*class* seperti yang ditunjukkan
pada contoh berikut.

```
class Superclass{
int a;
void display_a(){
System.out.println("a = " + a);
}
}
class Subclass extends Superclass {
int a;
void display_a(){
System.out.println("a = " + a);
}
void set_super_a(int n){
super.a = n;
}
void display_super_a(){
super.display_a();
}
}
class SuperDemo {
public static void main(String args[]){
Superclass SuperObj = new Superclass();
Subclass SubObj = new Subclass();
SuperObj.a = 1;
SubObj.a = 2;
```

```
SubObj.set_super_a(3);
SuperObj.display_a();
SubObj.display_a();
SubObj.display_super_a();
System.out.println(SubObj.a);
}
}
```

Program tersebut akan menampilkan hasil berikut.

```
a = 1
a = 2
a = 3
2
```

2.2.2.16 Key*word* Static

Kata kunci *static* dapat digunakan untuk anggota dari sebuah *class*. Kata kunci ini menyediakan *static* atau anggota *class* untuk diakses sama sebelum beberapa instance dari *class* dibuat. Variabel *class* bersifat seperti variabel umum. Ini artinya bahwa variabel dapat diakses oleh semua instance dari *class* [3].

Method *class* mungkin dapat diambil tanpa membuat sebuah *object* dari *class* tersebut. Bagaimanapun, mereka hanya dapat mengakses anggota static dari *class*. Ditambahkan juga, mereka tidak dapat menunjuk *this* dan *super*.

Kata kunci *static* dapat juga diaplikasikan pada blok. Ini dinamakan dengan blok static. Blok ini dieksekusi hanya sekali, ketika *class* diisi. Hal ini biasanya digunakan untuk menginisialisasi variabel *class*.

```
class Demo {
static int a = 0;
```

```
static void staticMethod(int i) {
System.out.println(i);
}
static { //blok static
System.out.println("This is a static block.");
a += 1;
}
}
class StaticDemo {
public static void main(String args[]) {
System.out.println(Demo.a);
Demo.staticMethod(5);
Demo d = new Demo();
System.out.println(d.a);
d.staticMethod(0);
Demo e = new Demo();
System.out.println(e.a);
d.a += 3;
System.out.println(Demo.a+", " +d.a +", " +e.a);
}
}
```

Keluaran dari source kode ditunjukkan di bawah ini.

```
This is a static block.
15
10
1
4, 4, 4
```

2.2.2.17 Key*word* Final

Kata kunci *final* dapat diaplikasikan pada variabel, method dan *class*. Untuk mengingat fungsi dari kata kunci, ingat bahwa itu secara mudah dibatasi apa yang kita dapat lakukan dengan variabel, method dan *class*. Nilai dari variabel final dapat tidak lama diubah sesudah nilainya telah diatur. Sebagai contoh,

final int *data* = 10;

Pernyataan berikut akan menyebabkan terjadi *compilation* error:

data++;

Method final tidak dapat di-override dalam *class* anak.

final void myMethod() { //in a parent *class*

}

myMethod tidak dapat lama di-override dalam *class* anak.

class final tidak dapat diwariskan tidak seperti *class* yang biasanya.

final public *class* My*Class* {

}

Petunjuk Penulisan Program:

Perintah penulisan kata kunci final dan public memungkinkan bertukar tempat.

Pernyataan ini akan menyebabkan kesalahan *compilation* terjadi karena *MyClass* dapat tidak lama

di-*extended*.

public Wrong*Class* extends My*Class* {

}

2.2.2.18 Inner *Class*es

Sebuah inner *class* secara mudah dideklarasikan dalam *class* lain.

```
class OuterClass {
int data = 5;
class InnerClass {
int data2 = 10;
void method() {
System.out.println(data);
System.out.println(data2);
}
}
public static void main(String args[]) {
OuterClass oc = new OuterClass();
InnerClass ic = oc.new InnerClass();
System.out.println(oc.data);
System.out.println(ic.data2);
ic.method();
}
}
```

Untuk mampu mengakses anggota dari inner *class*, kita butuh sebuah instance dari inner *class*. Method-method dari inner *class* dapat secara langsung mengakses anggota dari outer *class*.

2.2.3 *Data Base*

Database adalah sebuah struktur yang umumnya terbagi kedalam 2 hal, yaitu *database* flat dan sebuah *database* relasional [4]. *Database* relasional lebih mudah difahami daripada *database* flat karena *database*

relasional mempunyai bentuk yang sederhana serta mudah dilakukan operasi *data*. *MYSQL* sendiri adalah sebuah *database* relasional. *Database* yang memiliki struktur relasional terdapat tabel tabel untuk menyimpan *data*. Pada setiap tabel terdiri dari serta kolom untuk mendefinisikan jenis informasi apa yang harus disimpan.

2.2.3.1 *Database* Relasional

Ketika menggunakan software sistem manajemen *database* terkomputerisasi, umumnya menggunakan *database* relasional. Prinsip *database* relasional adalah informasi dibagi bagi menjadi beberapa *data* yang terpisah secara logis. *Data data* yang terpisah tersebut diletakkan dalam bentuk tabel. Tabel adalah objek dasar yang merupakan jantung dari *database* relasional. Tabel adalah dasar penyimpanan informasi dan pengambilannya. Ketika informasi sudah di tabel tabel yang terpisah, kita nantinya dapat melihat (*view*), meng*edit* (*edit*), menambah (add) dan menghapus (delete) informasi dengan berbagai metode. Selain itu, kita juga dapat mengambil informasi menggunakan query dan menampilkan informasi menggunakan *report*.

Keunggulan penyimpanan *data* menggunakan *database* relasional sangat banyak dibandingkan dengan penyimpanan ke satu tabel ukuran besar dua dimensi (yang disebut flat) seperti dokumen *word* atau spreadsheet exccel.

Kelebihan *data base* antara lain:

i. Flexibilitas

   Jika *data* berubah, kita dapat mengupdatenya di satu tempat. Dengan demikian, semua query, form dan lainnya akan berubah secara konsekuen dengan nilai yang baru tersebut.

ii. Simple

Model penyimpanan tabel merupakan dasar dari sistem relasional memang simple, dan merupakan metode penyimpanan *data* yang tak redundan.

iii. Power

Penyimpanan *data* dalam tabel tabel yang terpisah lebih memudahkan adanya pengelompokan, searching, dan pengambilan *data* menggunakan banyak cara yang tak terbatas.

iv. Kemudahan manajemen

Dengan tabel yang kecil dan tidak kompleks, informasi jadi mudah dilacak dan diatur.

2.2.3.2 Query

Kita dapat mengambil informasi tertentu yang disimpan di tabel atau multi tabel menggunakan query. Untuk membuat query, kita harus menentukan parameter – parameter dari informasi yang diinginkan. Misalnya kita ingin mengambil *data* pelanggan yang telah membeli barang tipe x selama 3 bulan lalu. Dengan demikian kita harus mengisikan variabel nama barang dan periode. Informasi yang ditampilkan juga nantinya bisa diurutkan.

Query *MYSQL* secara garis besar mengenal beberapa statement – statement sebagai berikut:

a. Statement SELECT

Umumnya, sebuah *data* disimpan kedalam *database*, karena nantinya *data* tersebut ingin ditampilkan kembali. Cara menampilkannya menggunakan perintah select. Syntax dasar untuk perintah select di *MYSQL* adalah sebagai berikut

SELECT daftar_pilihan_kolom FROM daftar_tabel WHERE batasan_batasan_utama GROUP BY grup_kolom ORDER BY

urutan_kolom HAVING batasan_batasan_tambahan LIMIT jumlah.

Agar dapat memahami sintaxnya, berikut akan diuraikan satu persatu :

- Daftar_pilihan_kolom, menunjukkan kolom-kolom apa saja yang akan dipilih atau ditampilkan.
- Daftar_tabel, menunjukkan dari tabel apa saja baris baris *data* diambil atau dipilih.
- Batasan_batasan_utama, menunjukkan apa saja kondisi – kondisi baris *data* yang harus dipenuhi
- Grup_kolom, menunjukkan bagaimana mengelompokkan hasil –hasil query.
- Urutan_kolom, menunjukkan bagaimana mengurutkan hasil query.
- Batasan_batasan_tambahan, menunjukkan apa saja kondisi – kondisi tambahan dari baris *data* yang harus terpenuhi.
- Jumlah, menunjukkan batasan jumlah baris dari query.

b. Statement REPLACE

Statement replace mengembalikan sebuah informasi yang menunjukkan jumlah baris yang dipengaruhi. Jika jumlah baris yang dipengaruhi oleh replace adalah satu, baris hanya ditambahkan kedalam tabel tanpa ada proses mengganti *data* yang ada. Jika jumlah baris *data* yang dipengaruhi oleh perintah replace adalah dua baris, ada sebuah baris yang dihapus sebelum proses penambahan baris *data* dilakukan. Perintah replace memiliki format penulisan perintah yang sama dengan insert, yaitu sebagai berikut:

```
REPLACE [ LOW_PRIORITY | DELAYED]
[INTO] table_name [{col_name,...}]
VALUES ({expr |DEFAULT},...),(...),...
```

Atau

```
REPLACE INTO nama_tabel (daftar_kolom)
VALUES (daftar_nilai);
```

c. Memperbarui *Data* (UPDATE)

Statement update digunakan untuk mengubah isi *data* di baris-baris yang ada. Cara menggunakannya adalah dengan menentukan nama tabel yang akan diubah isi *data*nya. Bisa juga ditampilkan fungsi SET yang diikuti dengan daftar pasangan nama kolom dan nilainya, dan biasanya perintah update diikuti dengan fungsi where yang menetukan baris baris mana sajakah yang akan diubah. Berikut format penulisannya:

```
UPDATE nama_tabel SET
kolom1='data_baru',
kolom2='data_baru',
....
Kolomn='data_baru'
WHERE [kondisi];
```

d. JOIN TABLE

Kemampuan untuk mengakomodir join bisa dibilang merupakan fitur utama *MySQL*. Karena itu, sangat penting bagi anda untuk membuat sebuah perintah select yag mengandung join. *MySQL* adalah *database* relasional, artinya satu kelompok/record dan dapat dibagi kedalam beberapa tabel. Namun, pemisahan tersebut nantinya harus bisa digabungkan kembali sehingga pemisahannya haruslah secara logis.

Karena itulah ada “hubungan khusus” antara tabel-tabel agar nantinya bisa digabungkan lagi. Misalnya, tabel mahasiswa akan berkaitan erat dengan dengan tabel mata_kuliah dan tabel KRS.

Dengan menyimpan informasi kedalam tempat yang berbeda, ada keuntungan yang bisa diperoleh, yaitu minimalisasi *data* yang ganda/duplikat.

Untuk membuat itu semua, digunakanlah join. Join adalah kegiatan penggabungan dua buah tabel atau lebih dimana penggabungannya relevan, artinya ada hubungan antara nilai dikolom yang dipilih. Join tabel menghasilkan *data*, dimana *data* tersebut hanya ada secara temporer, selama query dijalankan.

e. CROSS-JOIN

Cross-join antara dua buah tabel adalah mengambil *data* dari baris *data* dalam tabel 1 dan menggabungkannya dengan setiap baris-baris *data* dari tabel 2. Untuk melakukannya, gunakan format perintah sebagai berikut:

```
SELECT * FROM table_name1
CROSS JOIN table_name2;
```

Jika query diatas dijalankan maka akan menghasilkan *data* sebanyak hasil perkalian antara jumlah baris di table_name1 dengan jumlah baris di table_name2. Untuk mendapatkan hasil yang diinginkan dapat digunakan Klausa WHERE untuk membatasi hasil dari query tersebut

2.2.4 Mengenal *MySQL*

MySQL pertama kali dirintis oleh seorang programmer *database* bernama Michael Widenius.

MySQL database server adalah RDBMS (Relasional *Database Manage*ment *System*) yang dapat menangani *data* yang bervolume besar [4]. Meskipun begitu, tidak menuntut resource yang besar. *MySQL* adalah *database* yang paling populer diantara *database* engine yang lainnya.

MySQL adalah program *database* yang mampu mengirim dan menerima *data* dengan sangat cepat dan multi *user*. *MySQL* memiliki dua bentuk lisensi, yaitu freeware dan shareware. Beberapa fase penting dalam pengembangan *MySQL* adalah sebagai berikut:

- *MySQL* dirilis pertama kali secara internal pada 23 Mei 1995.
- Versi Windows pertama kali dirilis pada 8 Januari 1998 untuk windows 95 dan windows NT.
- Versi 3.23:beta dari juni 2000, dan dirilis pada Januari 2001.
- Versi 4.0:beta dari Agustus 2002, dan dirilis pada Maret 2003 (Unions).
- Versi 4.1:beta dari bulan Juni 2004, dirilis pada Oktober 2004.
- Versi 5.0:beta dari bulan Maret 2005, dirilis pada oktober 2005. (cursor,stored procedure,trigger,*view*s XA transaction).
- Sun Micro*System* membeli *MySQL* AB pada tanggal 26 Februari 2008.
- Versi 5.1: dirilis 27 Nopember 2008 (event scheduler, artitioning,plug in API, row-*base*d replication, server log table).

Load Balancing adalah sebuah konsep yang gunanya untuk menyeimbangkan beban atau muatan pada infrastruktur Teknologi Informasi sebuah perusahaan/ intansi.Agar seluruh departemen/ bagian dapat memanfaatkan secara maksimal dan optimal. Berfungsi menggabungkan beberapa *line Internet Service Provider.*

2.2.5 Netbeans

NetBeans adalah *Integrated Development Environment* (IDE) berbasiskan Java dari Sun Microsystems yang berjalan di atas Swing.

Swing sebuah teknologi Java untuk pengembangan aplikasi Desktop yang dapat berjalan di berbagai macam platforms seperti Windows, Linux, Mac OS X and Solaris. NetBeans mengacu pada kedua platform kerangka untuk aplikasi desktop Java, dan sebuah lingkungan pengembangan terpadu (IDE) untuk pengembangan dengan Java , JavaScript , PHP , Python , Ruby , Groovy , C , C + + , Scala , Clojure,dan lain-lain [5].