

BAB II

TINJAUAN PUSTAKA

2.1 Penelitian Terkait

Pada bagian ini mengemukakan tentang referensi dari beberapa penelitian terkait dengan topik bahasan yang penulis angkat dalam penelitian ini. Penelitian yang digunakan sebagai bahan studi literatur dalam penelitian ini, antara lain :

1. Penelitian yang dilakukan oleh Muhammad Insan Kamil (2015) dengan judul “Rancang Bangun Aplikasi Pencarian Rute Terpendek Lokasi Wisata Kuliner Kota Pontianak Berbasis Mobile”. Penelitian tersebut bertujuan membangun aplikasi yang dapat membantu penikmat kuliner melakukan perjalanan ke tempat kuliner dengan jarak yang lebih optimal. Aplikasi yang dibangun berbasis *web* dengan menggunakan algoritma dijkstra dan memanfaatkan *location based service*.
2. Penelitian yang dilakukan oleh Putri M. Hutabarat (2017) dengan judul “Penyelesaian *Travelling Salesman Problem* (TSP) Dengan Menggunakan Algoritma *Simulated Annealing* (Studi Kasus : *Mail Processing Center* (MPC) Medan PT. Pos Indonesia)”. Pada penelitian tersebut algoritma *simulated annealing* digunakan menyelesaikan masalah *traveling salesman problem* pada MPC Medan PT. POS Indonesia. Pengujian dilakukan pada kasus pengambilan paket pengiriman yang terdapat pada 8 tempat di kota Medan, proses perhitungan pada penelitian ini dilakukan dengan bantuan aplikasi MATLAB.
3. Penelitian yang dilakukan oleh Edi Samana, Bayu Prihandono, dan Evi Noviani (2015) dengan judul “Aplikasi *Simulated Annealing* untuk Menyelesaikan *Traveling Salesman Problem*”. Pada penelitian tersebut mereka berfokus melakukan pengujian dalam menerapkan metode *simulated annealing* untuk menyelesaikan *traveling salesman problem* pada PT. XX di kota Pontianak. Pada penelitian tersebut aplikasi untuk melakukan perhitungan belum dibuat dan pengujian masih menggunakan perhitungan manual.

2.2 *Traveling Salesman Problem*

Traveling salesman problem memiliki sejarah panjang. Dicituskan pertama kali oleh Euler pada awal tahun 1759, walaupun dengan nama yang

berbeda, yang tertarik untuk menyelesaikan permasalahan “*knight’s tour*”. Solusi yang tepat akhirnya diperoleh ketika pion kuda melewati tiap-tiap rute yang memungkinkan dari 64 kotak pada papan catur tepat satu kali dalam perjalanannya. Yang artinya setelah pion kuda tersebut melewati seluruh kemungkinan rute akhirnya dia menemukan solusi yang tepat.

Kemudian sekitar tahun 1800 diperkenalkan kembali oleh matematikawan Irlandia bernama William Rowan Hamilton dan matematikawan Inggris bernama Thomas Penyngton yang berupa suatu permainan bernama Icosian Hamilton yang mengharuskan pemain untuk menyelesaikan perjalanan dari 20 titik dengan menggunakan hanya jalur-jalur tertentu. Oleh karena itu *travelling salesman problem* sangat erat hubungannya dengan *cycle hamilton* di mana dideskripsikan sebagai lintasan seorang *salesman* yang harus mengunjungi sebanyak n kota.

Misalkan adalah jarak perjalanan dari kota i ke kota j dan *salesman* ingin melakukan perjalanan dengan biaya total yang minimum, yang mana biaya total adalah jumlah masing-masing biaya tiap *edge* atau jalur perjalanannya (Vasudev, 2006). Sehingga *travelling salesman problem* didefinisikan sebagai suatu permasalahan optimasi yang bertujuan untuk mendapatkan rute terpendek (minimum) dari beberapa tempat atau kota yang harus dilalui seorang *salesman* tepat satu kali ia hingga kembali ke tempat awal keberangkatannya. Jadi, secara sederhana *travelling salesman problem* merupakan permasalahan seorang *salesman* yang harus melakukan kunjungan tepat satu kali pada semua kota dalam sebuah lintasan sebelum dia kembali ke titik awal keberangkatannya.

2.3 *Simulated Annealing*

Menurut Kirkpatrick et al (1983) dan Cerny (1985) penyelesaian dengan *simulated annealing* dinilai ampuh dalam mencari solusi optimum yang bersifat numerik karena mampu menghindari kondisi jebakan optimum lokal. Optimum lokal adalah suatu keadaan dimana semua tetangga yang berdekatan dengannya lebih buruk atau sama dengan keadaan dirinya yang mengakibatkan solusi yang dihasilkan tidak diterima.

Secara garis besar, dasar algoritma *simulated annealing* (Kirkpatrick et al, 1983) adalah sebagai berikut:

1. Inisialisasi solusi *trial* awal yaitu membangkitkan solusi awal secara acak.
2. Iterasi, dengan cara menukarkan dan mengubah solusi *trial* terdekat yang digunakan. Jika solusi yang terbaru adalah yang terbaik, maka solusi tersebut akan menggantikan solusi yang digunakan sebelumnya.
3. Memeriksa temperature *schedule*.
4. Menghentikan aturan perpindahan iterasi.

Sedangkan Johnson dan McGeoch (1995) memperkenalkan bentuk umum dari algoritma *simulated annealing* adalah sebagai berikut:

1. Membangkitkan kondisi awal simulasi S dan menginisialisasi solusi akhir $S^* = S$
2. Menentukan temperatur T awal
3. Jika “pendinginan” belum terpenuhi, maka :
 - 3.1 jika temperatur yang “seimbang” belum terpenuhi, maka:
 - a) pilih *neighbor* acak S' sebagai solusi terbaru
 - b) tentukan $\Delta = \text{panjang}(S') - \text{panjang}(S)$
 - c) jika $\Delta \leq 0$ (menurun) maka $S = S'$,
 - d) jika $\text{panjang}(S) < \text{panjang}(S^*)$; $S^* = S$
 - e) selain itu (meningkat):
 - f) pilih bilangan r yang berdistribusi seragam secara acak dari $[0,1]$,
 - g) jika $r < e^{-\Delta/T}$; $S = S'$.
 - h) Akhiri simpul “keseimbangan belum terpenuhi.”
 - 3.2 turunkan temperatur T
 - 3.3 akhiri simpul “pendinginan belum terpenuhi”
4. Kembali ke S^* .

Menurut Rizal (2007) ada lima komponen utama yang sangat penting dalam menyusun algoritma *simulated annealing* pada *traveling salesman problem*, yaitu :

1. Konfigurasi sistem atau *image* awal proses.

2. Fungsi objektif di mana fungsi ini didefinisikan sebagai fungsi sasaran yang diminimumkan yang analogi dengan energi (total jarak yang ditempuh *salesman*).
3. Parameter kontrol yang analogi dengan temperatur sistem dan merupakan parameter bebas.
4. Mekanisme untuk mengubah konfigurasi interaksi antar titik pengamatan yang analogi dengan pertukaran rute-rute perjalanan antar titik.
5. *Annealing* schedule yang dinotasikan dengan fungsi $T_n = f(T_0, i, N)$, analogi dengan menurunkan temperatur untuk setiap iterasinya.

Menurut Suyanto (2010) Struktur algoritma SA untuk TSP adalah sebagai berikut :

1. Evaluasi rute awal. Jika rute awal merupakan tujuan, maka pencarian selesai dan keluar. Jika bukan, lanjutkan dengan menetapkan rute awal sebagai rute sekarang.
2. Inisialisasi BEST-SO-FAR dengan rute sekarang.
3. Inisialisasi sesuai dengan *annealing schedule*.
4. Ulangi hingga solusi ditemukan atau sudah tidak ada lagi aturan yang bisa diaplikasikan ke rute sekarang.
 - a. Pilih sebuah aturan yang belum pernah digunakan tersebut untuk menghasilkan rute baru.
 - b. Evaluasi rute yang baru dengan menghitung total energi:

$$\Delta E = \text{nilai rute sekarang } (Z_c) - \text{nilai rute baru } (Z_n)$$

- i. Jika rute baru merupakan tujuan, maka pencarian berhasil dan keluar.
- ii. Jika rute baru bukan merupakan tujuan tetapi memiliki nilai yang lebih baik daripada rute sekarang ($Z_n \leq Z_c$), maka tetapkan rute baru sebagai rute sekarang. Demikian juga tetapkan BEST-SO-FAR ke rute yang baru.

- iii. Jika nilai rute baru tidak lebih baik daripada nilai rute sekarang ($Z_n \geq Z_c$), maka tetapkan rute baru sebagai rute sekarang dengan probabilitas:

$$p = e^{-\frac{Z_n - Z_c}{T_i}} \quad (2.1)$$

dengan :

Z_n = Nilai rute sekarang

Z_c = Nilai rute baru

T_i = Sebuah parameter yang ukurannya cenderung untuk menerima rute baru.

Langkah ini biasanya dikerjakan dengan membangkitkan suatu bilangan random (r) pada interval $[0,1]$. Jika $r < p$, maka rute baru menjadi rute sekarang. Jika tidak, maka tidak akan dikerjakan apapun.

- c. Perbaiki T berdasarkan *annealing schedule*.

5. BEST-SO-FAR adalah solusi minimum global yang diharapkan.

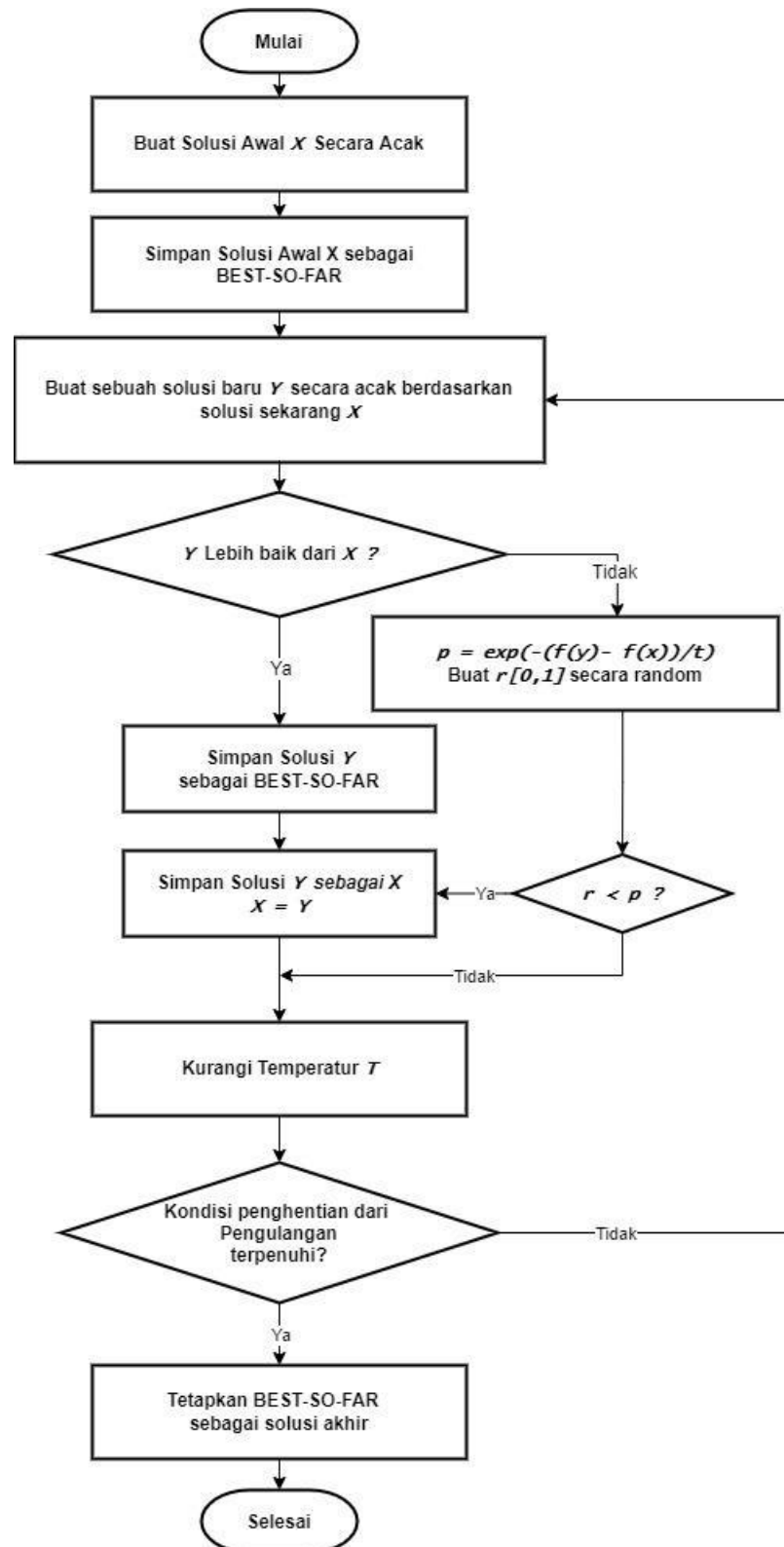
Adapun Tabel 2.1 yang menunjukkan pemetaan dari *physical annealing* kedalam SA adalah sebagai berikut:

Tabel 2.1 Pemetaan *Physical Annealing* Kedalam SA

Fisika (Termodinamika)	<i>Simulated Annealing</i> (Optimasi)
Keadaan sistem	Solusi yang mungkin
Energi	Biaya
Perubahan keadaan	Solusi tetangga
Temperatur	Parameter kontrol
Keadaan beku	Solusi heuristik

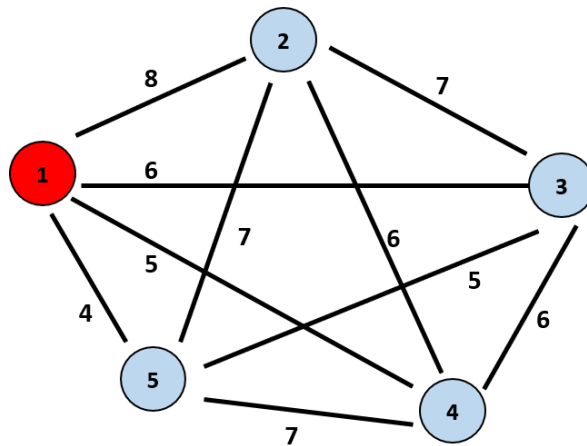
Sumber : Suyanto (2010)

Gambaran cara kerja *Simulated Annealing* dalam bentuk *flowchart* dapat dilihat pada Gambar 2.1 seperti sebagai berikut:



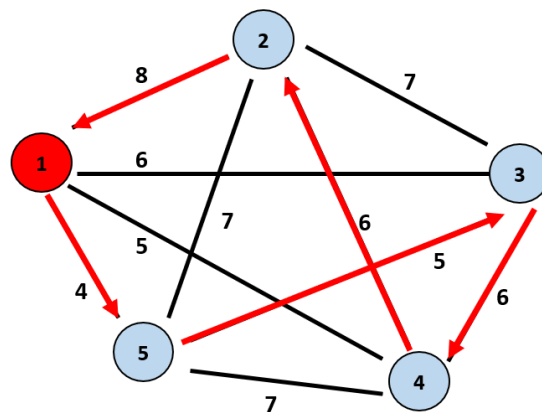
Gambar 2.1 *Flowchart Simulated Annealing*

Berikut contoh penerapan metode *simulated annealing* menggunakan graf yang terlihat pada Gambar 2.2:



Gambar 2.2 Graf Simulasi Perjalanan *Salesman*

Pada Gambar 2.2 terdapat graf perjalanan *salesman* dimana posisi *salesman* berada pada titik nomor 1, kondisi yang harus dipenuhi adalah *salesman* harus mengunjungi seluruh titik kemudian kembali ketitik awal. Jika tanpa menggunakan aplikasi umumnya *salesman* akan mengunjungi tempat terdekat lebih dahulu, seperti gambar sebagai berikut :

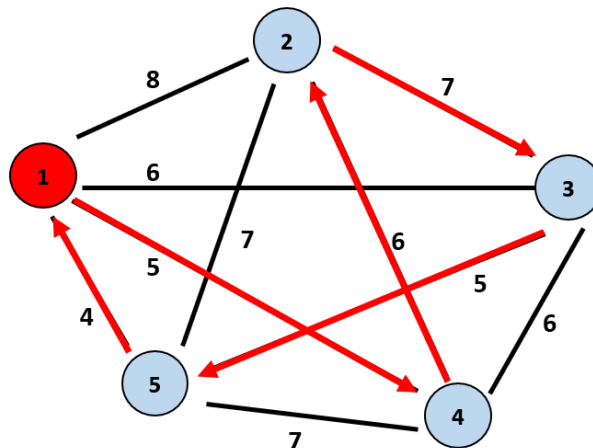


Gambar 2.3 Rute Manual dengan Memilih Tempat Terdekat

Berdasarkan gambar 2.3 total rute perjalanan yang dilalui *salesman* jika memilih tempat terdekat lebih dahulu adalah :

$$4 + 5 + 6 + 6 + 8 = 29$$

Sedangkan jika kita menggunakan metode *simulated annealing* jalur yang akan dilalui *salesman* adalah sebagai berikut:



Gambar 2.4 Rute menggunakan *Simulated Annealing*

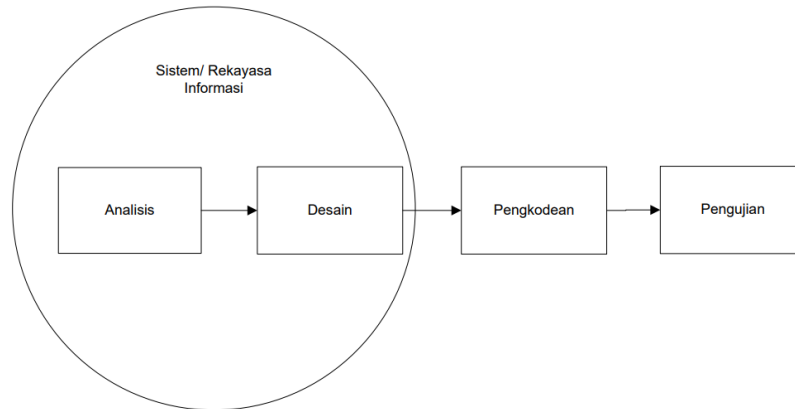
Berdasarkan Gambar 2.4 total rute yang akan dilalui *salesman* jika menggunakan metode *simulated annealing* adalah :

$$5 + 6 + 7 + 5 + 4 = 27$$

Total jarak yang diperoleh rute manual dengan cara mengunjungi tempat terdekat lebih dahulu adalah 29 Sedangkan total jarak yang diperoleh dengan menggunakan metode *simulated annealing* adalah 27. Pada kasus ini dapat disimpulkan metode *simulated annealing* memperoleh rute dengan total jarak yang lebih optimal dibandingkan rute dengan pemilihan tempat secara manual.

2.4 Model Pengembangan Perangkat Lunak

Model pengembangan perangkat lunak atau biasa dikenal dengan *software development life cycle* (SDLC) yang digunakan untuk pembangunan aplikasi pada penelitian ini adalah model SDLC *waterfall*. Model ini menggunakan pendekatan secara sistematis dan terurut karena setiap tahap yang akan dilalui harus menunggu tahap sebelumnya selesai dan berjalan berurutan. Gambaran model *waterfall* dapat dilihat pada Gambar 2.5.



Gambar 2.5 Model *Waterfall*

Sumber : A.S & Shalahuddin (2015)

Adapun penjelasan tahapan model *waterfall* menurut A.S & Shalahuddin (2015) seperti sebagai berikut :

1. Analisis kebutuhan perangkat lunak

Sebelum sistem buat, diperlukan suatu analisis sebagai dasar untuk mengetahui kebutuhan sistem kedepannya. Analisis kebutuhan sistem terdiri dari analisis kebutuhan fungsional yang bertujuan untuk mengetahui kebutuhan fungsi sistem dan analisis kebutuhan non fungsional untuk mengetahui perangkat keras dan perangkat lunak yang dibutuhkan serta kriteria pengguna sistem.

2. Desain

Desain berfungsi sebagai dasar perancangan yang mengubah data-data yang didapat dari analisis menjadi sebuah rancangan yang terdiri dari desain struktur data, struktur navigasi, dan rancangan antarmuka.

3. Pembuatan kode program

Tahapan ini merupakan lanjutan dari tahapan desain, yaitu mentranslasi desain menjadi sebuah program. Tahap ini menghasilkan suatu program yang sesuai dengan desain.

4. Pengujian

Program yang telah dibuat wajib diuji terlebih dahulu untuk memastikan bahwa program layak digunakan dari segi *logic* maupun fungsional. Pengujian ini dilakukan untuk meminimalisir kesalahan (*error*) dan memastikan keluaran yang dihasilkan sesuai dengan yang diinginkan.

5. Pendukung (*support*) atau pemeliharaan (*maintenance*)

Program yang telah diuji dapat mengalami perubahan ketika sudah dikirimkan ke pengguna. Perubahan dapat terjadi karena terjadi kesalahan yang tidak terdeteksi saat pengujian program harus beradaptasi dengan lingkungan baru (*hardware* baru). Tahap pendukung atau pemeliharaan bertujuan untuk menjaga stabilitas program yang telah dibuat tanpa harus membuat program yang baru

2.5 Aplikasi

Menurut Yuhefizar (2012) aplikasi merupakan program yang dikembangkan untuk memenuhi kebutuhan pengguna dalam menjalankan pekerjaan tertentu. Sedangkan, Pramana (2020) aplikasi merupakan suatu unit perangkat lunak yang dibuat untuk melayani kebutuhan akan beberapa aktivitas seperti perniagaan, *game*, pelayanan masyarakat, periklanan, atau semua proses yang hampir dilakukan oleh manusia.

Berdasarkan kedua definisi diatas, dapat diambil kesimpulan bahwa aplikasi merupakan program perangkat lunak yang dikembangkan sesuai dengan kebutuhan pengguna untuk melayani aktivitas yang dilakukan oleh pengguna.

2.6 Google Maps API

Google Maps API adalah sebuah layanan (*service*) yang diberikan oleh *Google* kepada para pengguna untuk memanfaatkan *Google Map* dalam mengembangkan aplikasi. *Google Maps API* menyediakan beberapa fitur untuk memanipulasi peta, dan menambah konten melalui berbagai jenis *services* yang dimiliki, serta mengizinkan kepada pengguna untuk membangun aplikasi *enterprise* di dalam *website*-nya, *Google Maps API* adalah suatu *library* yang berbentuk *JavaScript* (Kindarto, 2008).

Google Maps API akan penulis gunakan untuk menampilkan peta pada aplikasi yang penulis buat. Adapun *Direction API* salah satu *service* dari *Google* yang akan penulis gunakan untuk memperoleh data jarak secara dinamis yang terdapat pada *Google Maps*.

Directions API adalah salah satu layanan dari *google* yang memudahkan kita (*developer*) untuk mencari rute dan navigasi dari satu tempat ke tempat tertentu.

Kita tinggal memasukkan *latitude* dan *longitude* posisi berangkat dan juga *latitude* *longitude* posisi tujuan. Keunggulan dari API ini adalah dia mudah digunakan, kita hanya tinggal melakukan *HTTP Request* untuk memanggil *Google Directions API*.

Kode Program 2.1 *Google Maps API Key*

```

1 <script src="https://maps.googleapis.com/maps/api/js?key=AIzaSy...
2 ..&language=id&region=ID&v=3&libraries=geometry,places,drawing">
3 </script>

```

2.7 Alat Bantu Perancangan Sistem

Dalam penelitian ini, proses perancangan dilakukan dengan menggunakan beberapa alat bantu yang dapat diuraikan sebagai berikut :

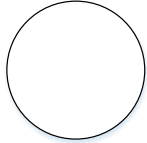
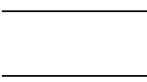

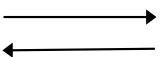
1. *Data Flow Diagram* (DFD)
2. *Entity Relationship Diagram* (ERD)

2.7.1 Data Flow Diagram (DFD)

Menurut Kristanto (2008), *Data Flow Diagram (DFD)* merupakan suatu model logika data atau proses yang dibuat untuk menggambarkan darimana asal data dan kemana tujuan data yang keluar dari sistem, dimana data disimpan, proses apa yang menghasilkan data tersebut dan interaksi antara data yang tersimpan dan proses yang dikenakan pada data tersebut. Selanjutnya, Sukamto dan Salahudin (2014) mendefinisikan bahwa *Data Flow Diagram (DFD)* atau dalam bahasa Indonesia menjadi Diagram Alir Data (DAD) adalah representasi grafik yang menggambarkan aliran informasi dan transformasi informasi yang diaplikasikan sebagai data yang mengatur dari masukan dan keluaran. *DFD* tidak sesuai untuk memodelkan sistem yang menggunakan pemrograman berorientasi objek.

Berdasarkan kedua pendapat di atas, dapat disimpulkan bahwa *Data Flow Diagram (DFD)* merupakan gambaran dalam bentuk diagram grafik yang menjelaskan tentang aliran data atau informasi yang berjalan pada suatu sistem. Selanjutnya terdapat bentuk notasi untuk penggambaran *Data Flow Diagram* yang diuraikan pada tabel 2.2.

Tabel 2.2 Simbol *Data Flow Diagram*

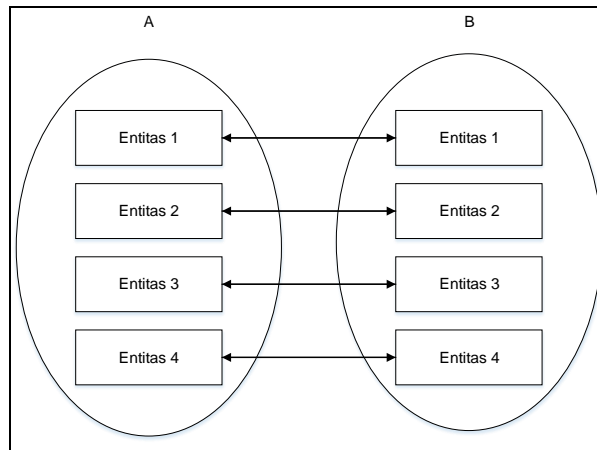
No.	Notasi	Keterangan
1		Merupakan notasi proses atau fungsi atau prosedur pada pemodelan perangkat lunak yang akan diimplementasikan dengan pemrograman terstruktur, maka pemodelan notasi ini yang kemudian menjadi fungsi atau prosedur di dalam kode program
2		Merupakan notasi untuk basis data atau penyimpanan pada pemodelan perangkat lunak yang akan diimplementasikan dengan pemrograman terstruktur.
3		Merupakan notasi entitas luar atau masukan atau <i>input</i> atau orang yang berinteraksi dengan perangkat lunak yang dimodelkan.
4		Merupakan notasi aliran data berupa data yang dikirim antar proses, dari penyimpanan ke proses, atau dari proses masukan ke proses keluaran.

Sumber : Sukamto & Salahudin (2014)

2.7.2 *Entity Relationship Diagram (ERD)*

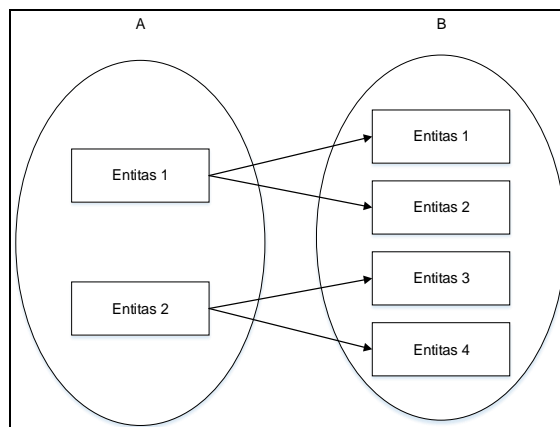
Ladjamudin (2005) menyatakan bahwa *Entity Relationship Diagram (ERD)* adalah suatu model jaringan yang menggunakan susunan data yang disimpan dalam sistem secara abstrak. *Entity Relationship Diagram* menggunakan sejumlah notasi dan simbol untuk menggambarkan struktur dan hubungan antar data dengan menggunakan kardinalitas relasi. Selanjutnya (Ladjamudin, 2005), menyatakan bahwa terdapat 3 macam kardinalitas relasi yang digunakan dalam penggambaran *Entity Relationship Diagram*, yaitu :

1. Relasi Satu ke Satu (*One to One*), merupakan relasi yang dinyatakan dengan satu kejadian pada entitas pertama dan kedua. Entitas pertama hanya memiliki satu hubungan dengan entitas kedua dan sebaliknya. Contoh gambaran relasi satu ke satu dapat dilihat pada gambar 2.6.

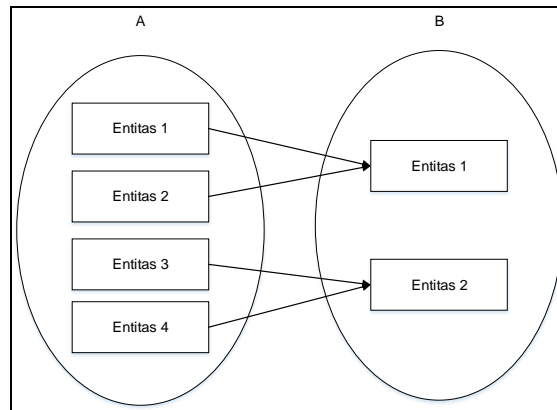


Gambar 2.6 ERD relasi satu ke satu

2. Relasi Satu ke Banyak atau Banyak ke Satu (*One to Many* atau *Many to One*), merupakan relasi yang dinyatakan dengan satu kejadian pada entitas pertama yang memiliki banyak hubungan dengan kejadian pada entitas kedua. Sebaliknya entitas kedua hanya memiliki satu kejadian dengan entitas pertama. Berikut contoh gambaran relasi satu ke banyak atau banyak ke satu :

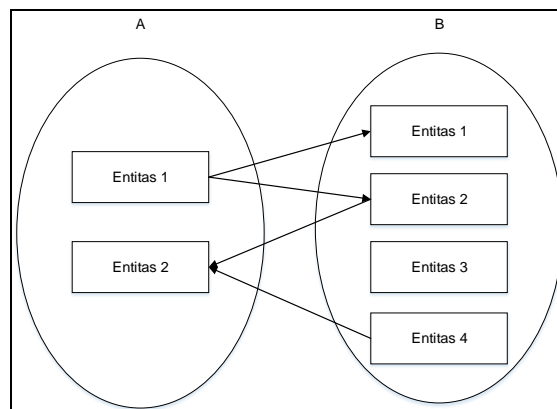


Gambar 2.7 ERD relasi satu ke banyak



Gambar 2.8 ERD relasi banyak ke satu

3. Relasi Banyak ke Banyak (*Many to Many*), merupakan relasi yang dinyatakan dengan satu entitas yang memiliki banyak hubungan dengan entitas lainnya dan sebaliknya. Gambaran mengenai relasi banyak ke banyak dapat dilihat pada gambar 2.9.



Gambar 2.9 ERD relasi banyak ke banyak

2.8 Teknologi Pendukung

2.8.1 *HyperText Markup Language (HTML)*

Menurut Kuswayatno (2006), *HTML* merupakan halaman yang berada pada suatu situs internet atau *web*. *HTML* merupakan metode yang menautkan (*link*) satu dokumen ke dokumen lain melalui teks. Menurut Deris Setiawan, *HTML* merupakan *framework* internet, hampir semua situs *web* yang ada menggunakan *HTML* untuk menampilkan teks, grafik, suara, dan animasinya. Sedangkan menurut Oktavian (2010), *HTML* adalah suatu bahasa yang dikenali oleh *web browser* untuk

menampilkan informasi dengan lebih menarik dibandingkan dengan tulisan teks biasa (*plain text*).

HTML adalah sebuah bahasa markah yang digunakan untuk membuat sebuah halaman *web*, menampilkan berbagai informasi di dalam sebuah penjelajah *web* Internet dan pemformatan hiperteks sederhana yang ditulis dalam berkas format *ASCII* agar dapat menghasilkan tampilan wujud yang terintegrasi. Dengan kata lain, berkas yang dibuat dalam perangkat lunak pengolah kata dan disimpan dalam format *ASCII* normal sehingga menjadi halaman *web* dengan perintah-perintah *HTML*.

Bermula dari sebuah bahasa yang sebelumnya banyak digunakan di dunia penerbitan dan percetakan yang disebut dengan *SGML (Standard Generalized Markup Language)*, *HTML* adalah sebuah standar yang digunakan secara luas untuk menampilkan halaman *web*. *HTML* saat ini merupakan standar Internet yang didefinisikan dan dikendalikan penggunaannya oleh *World Wide Web Consortium (W3C)*. *HTML* dibuat oleh kolaborasi Caillau TIM dengan Berners-lee Robert ketika mereka bekerja di CERN pada tahun 1989 (CERN adalah lembaga penelitian fisika energi tinggi di Jenewa).

Kode Program 2.2 Struktur Dokumen *HTML* Sederhana

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title> HTML SEDERHANA </title>
5  </head>
6  <body>
7    <h1> Skripsi Qaedi </h1>
8  </body>
9  </html>

```

2.8.2 Hypertext Preprocessing (PHP)

Menurut Anhar (2010) *PHP* adalah kependekan dari *PHP : Hypertext Preprocessor*, bahasa interpreter yang mempunyai kemiripan dengan bahasa C dan Perl yang mempunyai kesederhanaan dalam perintah. *PHP* merupakan bahasa *scripting* yang menyatu dengan *HTML* dan berada di server (*server-side HTML-embedded scripting*), artinya sintaks dan perintah-perintah yang diberikan akan sepenuhnya dijalankan di server tetapi disertakan pada halaman *HTML* biasa. Tujuan dari bahasa *scripting* ini adalah untuk membuat aplikasi-aplikasi yang dijalankan di atas teknologi *web*.

Kode Program 2.3 Koneksi Basis Data Pada Dokumen *PHP*

```

1  <?php
2  // Membuat variable dengan nama host dan database pada hosting
3  $host = "localhost";
4  $user = "root";
5  $pass = "";
6  $db   = "skripsi_qaedi";
7
8  //contoh menggunakan objek mysqli untuk membuat koneksi
9  $mysqli = new mysqli($host, $user, $pass, $db);
10
11 //contoh menggunakan native
12 $conn = mysqli_connect($host, $user, $pass, $db);

```

2.8.3 *MySQL*

Menurut Anhar (2010) “*MySQL* merupakan *software* yang tergolong *database server* yang bersifat *Open Source*. *Open Source* menyatakan bahwa *software* ini dilengkapi dengan *source code* (kode yang dipakai membuat *MySQL*). Selain tentu saja untuk *executable*-nya atau kode yang dapat dijalankan secara langsung dalam sistem operasi dan bisa diperoleh dengan cara mengunduh di internet secara gratis“.

MySQL menggunakan *SQL* sebagai bahasa dasar untuk mengakses *database*-nya. Sistem *database MySQL* memiliki sistem keamanan dengan tiga verifikasi yaitu *username*, *password*, dan *host*. Verifikasi *host* memungkinkan untuk membuka keamanan di ‘*localhost*’, tetapi tertutup bagi *host* lain (bekerja di lokal komputer). Sistem keamanan ini ada di dalam *database MySQL* dan pada tabel *user*. Proteksi juga dapat dilakukan terhadap *database*, tabel hingga kolom secara terpisah. Sebenarnya, *MySQL* merupakan turunan salah satu konsep utama dalam *database* sejak lama yaitu *SQL (Structured Query Language)*. *SQL* adalah sebuah konsep pengoperasian *database*, terutama untuk pemilihan atau seleksi dan pemasukan data, yang memungkinkan pengoperasian data dikerjakan dengan mudah secara otomatis.

Sebagai *database server*, *MySQL* dapat dikatakan lebih unggul dibandingkan *database server* lainnya dalam *query* data. Hal ini terbukti untuk *query* yang dilakukan oleh *single user*, kecepatan *query MySQL* bisa sepuluh kali lebih cepat dari *PostgreSQL* dan lima kali lebih cepat dibandingkan *Interbase*. Saat ini *MySQL* tersedia dibawah izin *open source*, tetapi juga ada izin untuk penggunaan secara komersial.

Keunggulan dari *MySQL* adalah :

1. Bersifat *open source*.
2. Sistem yang digunakan oleh perangkat lunak ini tidak memberatkan kerja dari server, karena dapat bekerja di *background*.
3. Mempunyai koneksi yang stabil dan kecepatan yang tinggi.

Kode Program 2.4 *Query MySQL* Dalam Dokumen *PHP*

```

1  <?php
2
3  $query = $mysqli->query("SELECT * FROM kategori WHERE id_kategori='1'");
4  $data = $query->fetch_array();
5  echo json_encode($data);

```

2.8.4 *Javascript*

Javascript menurut Sunyoto (2007) adalah bahasa *scripting* yang populer di internet dan dapat bekerja di sebagian besar *browser* populer seperti *Internet Explorer (IE)*, *Mozilla Firefox*, *Netscape* dan *Opera*. Kode *Javascript* dapat disisipkan dalam halaman *web* menggunakan tag *script*. Beberapa hal tentang *Javascript*:

1. *Javascript* didesain untuk menambah interaktif suatu *web*.
2. *Javascript* merupakan sebuah bahasa *scripting*.
3. Bahasa *scripting* merupakan bahasa pemrograman yang ringan.
4. *Javascript* berisi baris kode yang dijalankan di komputer (*web browser*).
5. *Javascript* biasanya disisipkan (*embedded*) dalam halaman *HTML*.
6. *Javascript* adalah bahasa *interpreter* (yang berarti skrip dieksekusi tanpa proses kompilasi).

Kode Program 2.5 *Javascript* Dalam Dokumen *HTML*

```

1  <!DOCTYPE html>
2  <html>
3  <head>
4    <title>Hello World Javascript</title>
5  </head>
6  <body>
7    <script>
8      console.log("Saya belajar Javascript di Dicoding");
9      document.write("Hallo Dicoding");
10   </script>
11 </body>
12 </html>

```

2.8.5 Bootstrap

Bootstrap merupakan sebuah *framework CSS* yang memudahkan pengembang untuk membangun *website* yang menarik dan responsif. *Bootstrap* adalah *CSS* tetapi dibentuk dengan *LESS*, sebuah *pre-processor* yang memberi fleksibilitas dari penggunaan *CSS* biasa. *Bootstrap* dapat dikembangkan dengan tambahan lainnya karena ini cukup fleksibel terhadap pekerjaan *web* yang mengutamakan desain (Otto, 2020). Selanjutnya Fauzi (2008), mengungkapkan bahwa “*Bootstrap* merupakan suatu metode berbasis komputer yang sangat potensial untuk dipergunakan pada masalah ketidakstabilan dan keakuratan, khususnya dalam menentukan interval konfidensi”.

Berdasarkan kedua pendapat ahli tersebut, dapat disimpulkan bahwa *bootstrap* merupakan sebuah *framework CSS* yang dirancang untuk membantu *developer* dalam melakukan desain terhadap rancangan *website* yang akan dibangun agar dapat lebih menarik, interaktif dan responsif.

```

1  <!doctype html>
2  <html lang="en">
3  <head>
4  <!-- Required meta tags -->
5  <meta charset="utf-8">
6  <meta name="viewport" content="width=device-width, initial-scale=1">
7
8  <!-- Bootstrap CSS -->
9  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
10 rel="stylesheet" crossorigin="anonymous">
11
12 <title>Hello, world!</title>
13 </head>
14 <body>
15 <h1>Hello, world!</h1>
16
17 <!-- Optional JavaScript; choose one of the two! -->
18
19 <!-- Option 1: Bootstrap Bundle with Popper -->
20 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
21 integrity="sha384-
22 MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
23 crossorigin="anonymous"></script>
24
25 <!-- Option 2: Separate Popper and Bootstrap JS -->
26 <!--
27 <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.9.2/dist/umd/popper.min.js"
28 crossorigin="anonymous"></script>
29 <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.min.js"
30 crossorigin="anonymous"></script>
31 -->
32 </body>
33 </html>

```

2.9 Pengujian Perangkat Lunak

Pengujian merupakan elemen kritis dari jaminan terhadap kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain, dan pengkodean. Karakteristik umum dari pengujian perangkat lunak adalah sebagai berikut (Sukamto R. A., 2009) :

1. Pengujian dimulai pada level modul dan bekerja keluar ke arah integrasi pada sistem berbasis komputer.
2. Teknik pengujian yang berbeda sesuai dengan poin-poin yang berbeda pada waktunya.
3. Pengujian diadakan oleh *software developer* dan untuk proyek yang besar oleh *group testing* yang independen.
4. *Testing* dan *debugging* adalah aktivitas yang berbeda tetapi *debugging* harus diakomodasikan pada setiap strategi *testing*.

Metode pengujian perangkat lunak ada 3 jenis, yaitu (Sukamto R. A., 2009):

1. *White Box / Glass Box* - pengujian operasi.
2. *Black Box* - untuk menguji sistem.
3. *Use Case* - untuk membuat *input* dalam perancangan *black box* dan pengujian *state based*.

Pengujian dalam penelitian ini dilakukan dengan menggunakan metode *Black Box*.

2.9.1 Pengujian *White Box*

Pengujian *White Box* adalah pendekatan pengujian yang berasal dari pengetahuan tentang struktur dan implementasi perangkat lunak. Pengujian *White Box* biasanya diterapkan pada unit program yang relatif kecil seperti subrutin atau operasi yang terkait dengan suatu objek (Sommerville, 2003).

Dengan menggunakan metode pengujian *White Box*, perekayasa sistem dapat melakukan *test case* yang (Pressman R. S., 2002):

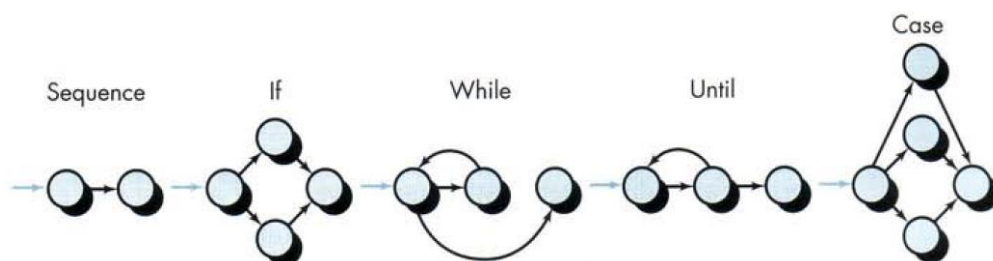
1. Menjamin bahwa seluruh independent *paths* dalam modul telah dilakukan sedikitnya satu kali,
2. melakukan seluruh keputusan logikal baik pada sisi *true* maupun *false*,
3. melakukan seluruh perulangan sesuai batasannya dan batasan operasionalnya,

4. menguji struktur data internal untuk menjamin validitasnya.

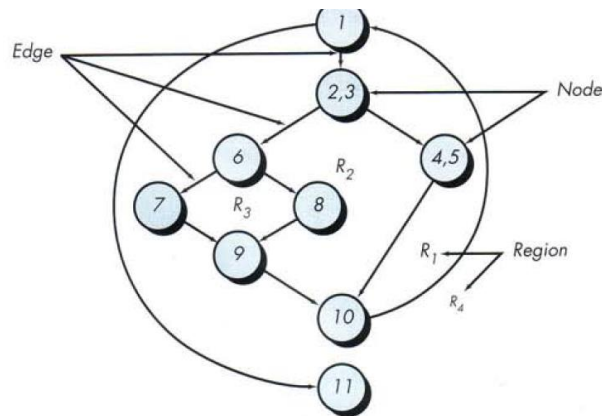
Pengujian *basic path* adalah pengujian *white box* yang diusulkan pertama kali oleh Tom McCabe. Metode ini memungkinkan pengukuran kompleksitas logis dari desain prosedural dan menggunakannya sebagai pedoman untuk menetapkan *basic set* dari semua jalur eksekusi. Konsep utama *basic path* yaitu tiap *basic path* harus diidentifikasi, tidak boleh ada yang terabaikan (setidaknya dites 1 kali) dan kombinasi dan permutasi dari suatu *basic path* tidak perlu dites.

1. Notasi Diagram Alir

Dokumentasi yang digunakan untuk menggambarkan cara pelaksanaannya adalah dokumentasi *flowchart* (atau ilustrasi program), yang menggunakan dokumentasi lingkaran (simpul atau *node*) dan anak panah (*link* atau *edge*). Dokumentasi ini menggambarkan aliran rasional kontrol yang digunakan dalam dialek pemrograman. Berdasarkan Gambar 2.7 dan 2.8, setiap lingkaran, yang disebut *flow graph node*, yang mewakili kepada satu atau lebih artikulasi prosedural. Proses berurutan yang digambarkan dalam bingkai kotak pada *flow graph* atau pilihan yang digambarkan dalam belah ketupat pada *flow graph* dapat diwakili oleh satu *node*. *flow graph*, yang disebut *edge* atau *links*, mewakili dengan aliran alat angkut kontrol dan secara praktis setara dengan baut pada diagram arus. *Edge* harus diakhiri dengan *links*. Area yang dibatasi oleh *edges* dan *nodes* disebut *regions*. Bila menghitung *regions*, harus juga mengikutkan area di luar dari grafik sebagai bagian dari *regions*.



Gambar 2.10 Notasi Diagram Alir



Gambar 2.11 Diagram Alir

2. Kompleksitas Siklomatis (*Cyclomatic Complexity*)

Kompleksitas siklomatis dapat berupa kisi program yang memberikan estimasi kuantitatif dari kompleksitas yang konsisten dari suatu program, nilai yang diperoleh akan menentukan jumlah cara bebas dalam cara yang ditetapkan, dan akan memberikan nilai batas atas untuk jumlah pengujian yang harus dilakukan. dilakukan, untuk menjamin bahwa semua pernyataan telah dieksekusi sedikitnya satu kali. Cara bebas dapat berupa cara dalam program yang menampilkan setidaknya satu set pegangan atau artikulasi kondisi yang tidak digunakan.

Adapun penghitungan siklomatis dapat diperoleh dari:

- *Region* (daerah muka) grafik alir

- $CC = E - N + 2$

E adalah jumlah edge, dan N adalah jumlah *node*

- $CC = P + 1$

P adalah simpul predikat

Simpul predikat adalah penggambaran suatu node yang memiliki satu atau lebih masukan (*input*), dan lebih dari satu keluaran (*output*).

2.9.2 Pengujian *Black Box*

Menurut Pressman (2010), *black box testing* juga disebut pengujian tingkah laku, memusat pada kebutuhan fungsional perangkat lunak. Teknik pengujian *black box* memungkinkan memperoleh serangkaian kondisi masukan yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu

program. Selanjutnya Shalahuddin dan Rosa (2010) mengungkapkan bahwa *black box testing* merupakan pengujian perangkat lunak dari segi spesifikasi fungsional tanpa menguji desain dan kode program. Pengujian dimaksudkan untuk mengetahui apakah fungsi-fungsi, masukan, dan keluaran dari perangkat lunak sesuai dengan spesifikasi yang dibutuhkan.

Berdasarkan kedua pendapat ahli di atas, dapat disimpulkan bahwa pengujian *black box* merupakan pengujian perangkat lunak dari segi fungsional suatu program atau aplikasi yang bertujuan untuk mengetahui apakah fungsi-fungsi, masukan, dan keluaran dari program atau aplikasi tersebut telah sesuai dengan spesifikasi yang dibutuhkan.

2.9.3 Pengujian Skala Likert

Pengujian *skala likert* merupakan pengujian yang dilakukan dengan metode kuesioner, kuesioner merupakan jenis pengujian dengan memberikan lembaran pertanyaan atau angket kepada orang lain mengenai penilaian terhadap sistem dimana nantinya hasil jawaban dari pengguna dapat dijadikan acuan dalam pengembangan sistem (Yusuf, 2005).

Perhitungan hasil dari kuesioner dilakukan dengan menggunakan cara *skala likert*. *Skala likert* digunakan untuk mengukur persepsi, sikap, dan pendapat seseorang atau sekelompok orang tentang fenomena sosial (Sugiyono, 2013). Untuk setiap pilihan jawaban akan diberi skor, maka responden harus menggambarkan, mendukung pertanyaan dengan jawaban yang dipilih. Dengan *skala likert*, variabel yang akan diukur dijabarkan menjadi indikator variabel. Kemudian indikator tersebut akan dijadikan sebagai tolak ukur menyusun item-item instrumen yang dapat berupa pertanyaan atau pernyataan.

Tabel 2.3 Skala Penilaian Untuk Pertanyaan Positif dan Negatif

Nilai	Kriteria
1	Sangat Buruk
2	Buruk
3	Cukup
4	Baik
5	Sangat Baik