BAB 2
LANDASAN TEORI

2.1 Peringkas Teks Otomatis

Dalam Kamus Besar Bahasa Indonesia Departemen Pendidikan Nasional Balai Pustaka (2008) mendefinisikan ringkasan, yaitu hasil meringkaskan, ikhtisar, singkatan cerita, dan kependekan. Sedangkan definisi otomatis dalam Kamus Besar Bahasa Indonesia Departemen Pendidikan Nasional Balai Pustaka (2008) yaitu dengan sendirinya, secara otomat. S*ummary* atau ringkasan didefinisikan sebagai penyajian bacaan dalam bentuk singkat dengan mempertahankan urutan isi dan sudut pandang / memendekkan bacaan dengan mengambil inti sari bacaan itu tanpa mengubah struktur wacana [6].

Peringkas teks otomatis adalah teknik pembuatan ringkasan dari sebuah teks secara otomatis dengan memanfaatkan aplikasi yang dijalankan pada komputer untuk menghasilkan informasi yang paling penting dari dokumen aslinya [7].

2.1.1 Tipe Ringkasan

Berdasar teknik pembuatan, suatu ringkasan diambil dari bagian terpenting dari teks aslinya [8], terdapat 2 tipe yaitu :

1. Abstraktif

   Tipe peringkasan abstraktif menghasilkan sebuah interpretasi terhadap teks aslinya. Dimana sebuah kalimat akan ditransformasikan menjadi kalimat yang lebih singkat dan kalimat baru yang tidak terdapat dalam dokumen yang asli atau dengan kalimat yang berbeda.

2. Ekstraktif

   Tipe peringkasan ekstraktif menghasilkan suatu ringkasan dengan memilih sebagian dari kalimat yang ada dalam dokumen asli. Metode ini menggunakan metode statistical, linguistical dan heuristic atau kombinasi dari semuanya dalam menetapkan ringkasan suatu teks.

Pada penelitian ini metode pendekatan yang digunakan untuk peringkasan teks otomatis yaitu teknik ekstraktif dengan memanfaatkan aplikasi yang dijalankan pada komputer untuk menghasilkan sebuah ringkasan dari dokumen aslinya.

Tujuan dari peringkasan teks (*text summarization*) dapat dikategorikan berdasarkan maksud, fokus dan cakupannya [9], sebagai berikut :

1. Informatif

   Informatif, ringkasan ini menyatakan informasi - informasi penting yang terdapat pada dokumen asal.

2. Indikatif

   Indikatif, tujuan dari ringkasan ini adalah untuk dijadikan sebuah referensi, yang membantu pembaca untuk mengetahui isi dari teks daripada membaca keseluruhan teks yang ada. Ringkasan ini meliputi topik kunci dari teks asal.

3. Evaluatif

   Evaluatif, atau ringkasan yang melibatkan pembuatan sebuah pertimbangan pada teks asal, seperti suatu tinjauan ulang atau opini.

4. *User-focused (query-relevant)*

   User-focused, ringkasan yang dibuat berdasarkan topik yang dipilih oleh user, sering merupakan jawaban dari query yang dimiliki oleh *user*.

5. *Generic*

   Generic, disebut juga *author-focused*, sifatnya lebih umum dan berdasarkan pada teks aslinya.

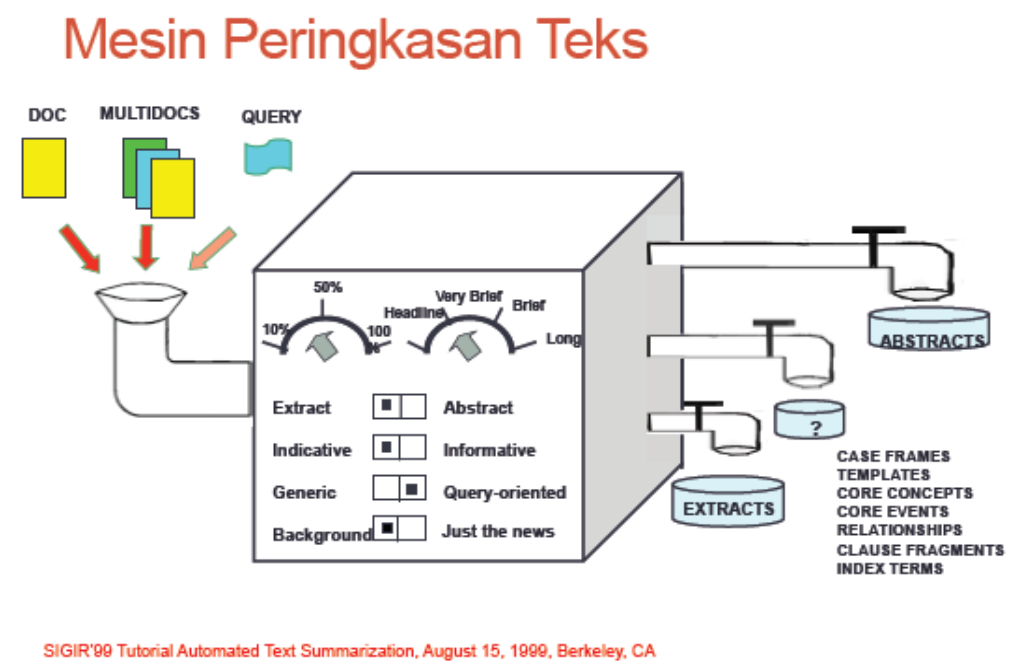
6. Dokumen tunggal (*single document*)

   Dokumen tunggal, ringkasan merupakan ringkasan dari satu dokumen.

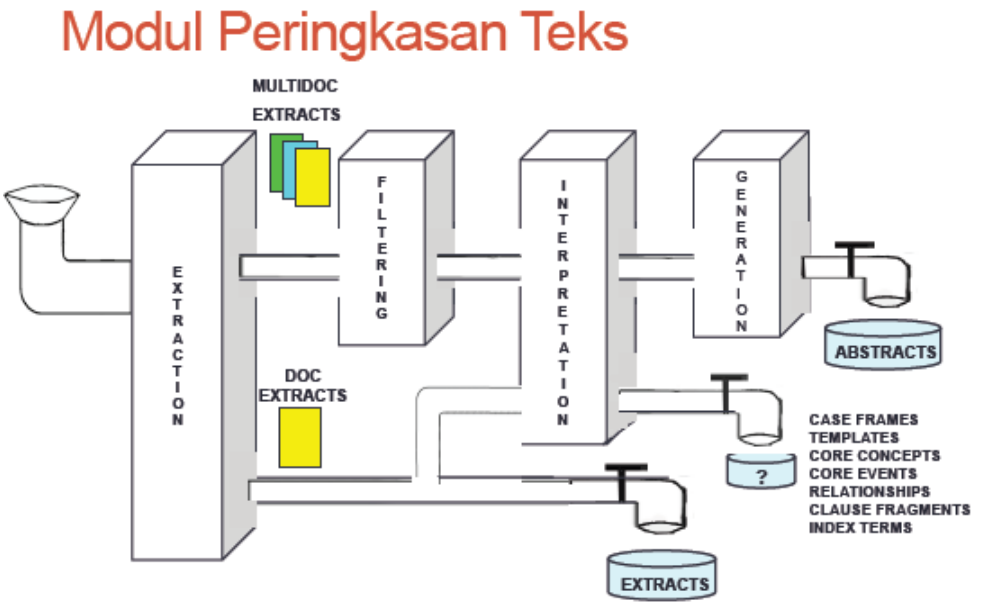
7. Banyak dokumen (*multi document*)

   Banyak dokumen, ringkasan merupakan hasil ringkasan dari banyak dokumen.

Berikut gambar mesin dan modul peringkas teks menurut [10]:



Gambar 2.1 Mesin Peringkas Teks



Gambar 2.2 Modul Peringkas Teks

2.2 Algoritma Text Summarization

Pada tahap ini akan menjelaskan algoritma yang digunakan dalam membangun aplikasi peringkas teks otomatis berbahasa Indonesia.

2.2.1 Pra proses (*Preprocessing*)

Pada tahap pra proses (*preprocessing*) dilakukan penyiapan dokumen mentah menjadi dokumen atau representatif dokumen yang siap diproses untuk langkah selanjutnya. *Preprocessing* pada penelitian ini terdiri dari beberapa tahapan, yaitu: proses pemisahan kalimat, proses *case folding*, proses *filtering*, proses *tokenizing*, proses *stopword removal* dan proses *stemming.*

2.2.1.1 Pemisahan Kalimat

Langkah awal tahapan *preprocessing* yang dilakukan yaitu pemisahan kalimat. Pemisahan kalimat ini merupakan proses pemecahan teks pada dokumen menjadi kumpulan kalimat. Teknik yang digunakan dalam pemisahan kalimat adalah memisahkan kalimat dengan tanda titik (.), tanda tanya (?), dan tanda seru (!) sebagai delimiter.

2.2.1.2 Case folding

Case folding adalah proses pemrosesan teks dimana semua teks diubah ke dalam case yang sama dan pada kasus ini teks diubah representasinya ke dalam huruf kecil semua [11].

2.2.1.3 *Filtering*

Data teks dalam dokumen yang sebelumnya sudah diubah ke dalam huruf kecil semua. Selanjutnya dilakukan proses *filtering* teks. *Filtering* adalah tahapan pemrosesan teks dimana semua teks selain karakter “a” sampai “z” akan dihilangkan dan hanya menerima spasi [11].

2.2.1.4 *Tokenizing*

Proses *Tokenizing* adalah proses pemotongan *string* text berdasarkan tiap kata yang menyusunnya. Pemecahan kalimat menjadi kata-kata tunggal dilakukan dengan men-*scan* kalimat dan setiap kata terindentifikasi atau terpisahkan dengan kata yang lain oleh *delimiter* spasi [12].

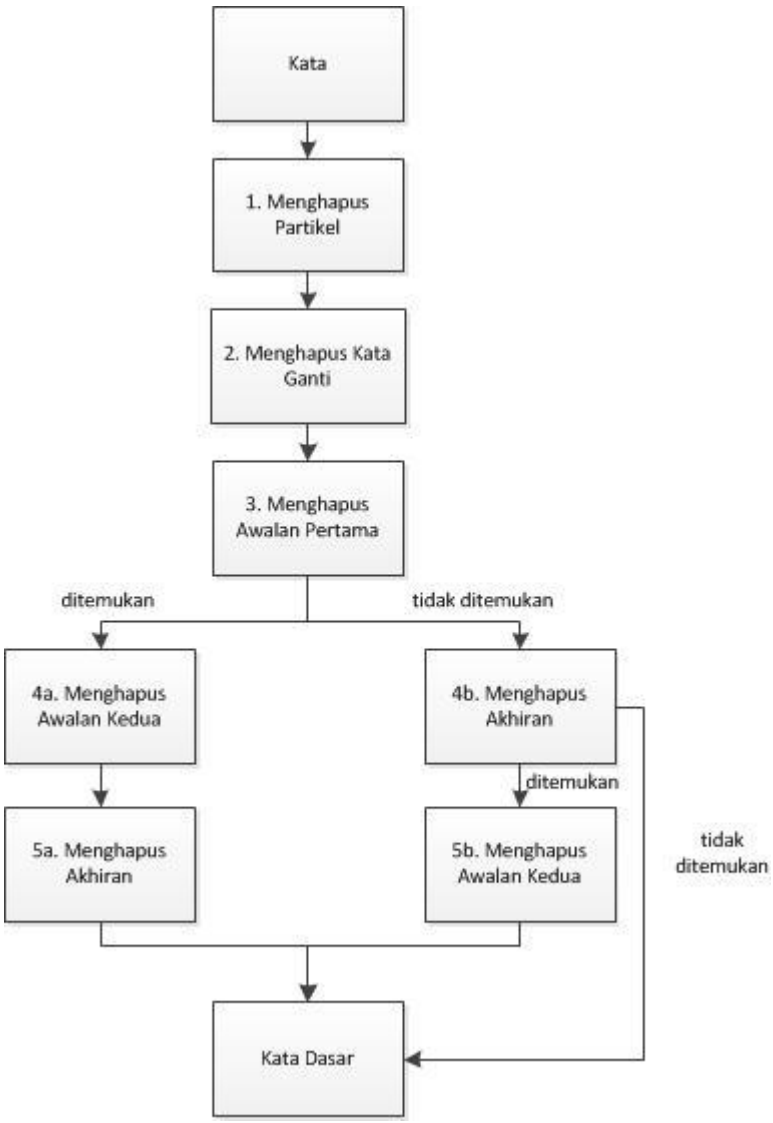
2.2.1.5 *Stopword Removal*

Stopword removal adalah proses menghilangkan kata-kata yang sering muncul, namun tidak berarti. *Stopword* dikatakan tidak berarti karena tidak memiliki keterkatitan dengan topik tertentu. Untuk mendeteksi apakah suatu kata merupakan suatu *stopword* atau bukan adalah menggunakan kamus *stopword* yang sudah ditentukan sebelumnya. Contoh *stopword* pada bahasa Indonesia adalah, di, ke, dari, pada, dan lain-lain.

2.2.1.6 *Stemming*

Stemming adalah proses pemetaan dari penguraian berbagai bentuk kata baik itu *prefix*, *sufix*, maupun gabungan antara *prefix* dan *sufix* (*confix*), menjadi bentuk kata dasarnya [13]. Proses *Stemming* merupakan proses pencarian akar kata (*root word*) dari tiap kata yaitu dengan mengembalikan suatu kata berimbuhan ke bentuk dasarnya (*stem*). Untuk pemrosesan pada bahasa Indonesia, proses *stemming* dilakukan dengan menghilangkan imbuhan yang mengawali dan mengakhiri kata sehingga diperoleh bentuk dasar dari kata tersebut [12]. Sebagai contoh, kata bersama, kebersamaan, menyamai, akan di-*stem* ke *root word*-nya yaitu "sama". Algoritma yang digunakan pada proses ini adalah algoritma Porter Stemmer bahasa Indonesia.

Algoritma ini dikembangkan oleh Fadillah Z. Tala pada tahun 2003. Berupa algoritma *stemmer* untuk bahasa. Algoritma ini mengadopsi dari algoritma *English Porter Stemmer* yang dikembangkan oleh Frakes pada tahun 1992. Dipilihnya algoritma Porter Stemmer untuk dikembangkan sebagai algoritma *stemmer* untuk bahasa Indonesia, karena pemikiran dasar dari algoritma Porter Stemmer cocok dengan struktur morfologi kata-kata di dalam bahasa Indonesia [14]. Perbedaan algoritma ini dengan algoritma yang telah dikembangkan oleh Nazief & Adriani yaitu tidak adanya *dictionary* sehingga algoritma ini dapat dikatakan murni berbasis rule [14]. Berdasarkan hasil penelitian [15] menyimpulkan bahwa proses *stemming* dokumen teks berbahasa Indonesia menggunakan algoritma Porter Stemmer membutuhkan waktu yang lebih singkat dibandingkan *stemming* dengan menggunakan algoritma Nazief & Adriani. Adapun gambar desain dari algoritma Porter Stemmer bahasa Indonesia dapat dilihat pada gambar 2.4 berikut.



Gambar 2.3 Desain Algoritma Porter Stemmer Bahasa Indonesia

Berikut penjelasan dari Gambar 2.4.

1. Menghapus partikel (*particle*) dengan aturan sesuai Tabel 2.1.
2. Menghapus kata ganti (*possesive pronoun*) dengan aturan sesuai Tabel 2.2.
3. Menghapus awalan pertama (*first order prefix*) dengan aturan sesuai Tabel 2.3. Jika ditemukan maka lanjutkan ke langkah 4a, jika tidak ditemukan maka lanjutkan ke langkah 4b.
4. a. Menghapus awalan kedua (*second order prefix*) dengan aturan sesuai Tabel 2.4, kemudian lanjutkan ke langkah 5a.

b. Menghapus akhiran (*suffix*) dengan aturan sesuai Tabel 2.5, jika ditemukan maka lanjutkan ke langkah 5b, jika tidak ditemukan maka kata tersebut diasumsikan sebagai kata dasar.

5. a. Menghapus akhiran (*suffix*) dengan aturan sesuai Tabel 2.5, dan kata akhir diasumsikan sebagai kata dasar.

   b. Menghapus awalan kedua (*second order prefix*) dengan aturan sesuai Tabel 2.4, dan kata akhir diasumsikan sebagai kata dasar.

Berdasarkan uraian diatas, menurut aturan yang ada pada algoritma Porter Stemmer bahasa Indonesia terdapat 5 kelompok aturan yang dapat dilihat pada Tabel 2.1 sampai Tabel 2.5 berikut [14]:

Tabel 2.1 Kelompok Aturan Pertama : Pembentukan Partikel

Akhiran	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
-kah	NULL	2	NULL	bukukah → buku
-lah	NULL	2	NULL	pergilah → pergi
-pun	NULL	2	NULL	bukupun → buku
-tah	NULL	2	NULL	apatah → apa

Tabel 2.2 Kelompok Aturan Kedua : Pembentukan Kata Ganti

Akhiran	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
-ku	NULL	2	NULL	bukuku → buku
-mu	NULL	2	NULL	bukumu → buku
-nya	NULL	2	NULL	bukunya → buku

Tabel 2.3 Kelompok Aturan Ketiga : Pembentukan Awalan Pertama

Awalan Pertama	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
meng-	NULL	2	NULL	mengukur → ukur
meny-	s	2	V…	menyapu → sapu
men-	NULL	2	NULL	menduga → duga
men-	t	2	V…	menuduh → tuduh
mem-	p	2	V…	memilah → pilah

Awalan Pertama	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
mem-	NULL	2	NULL	membaca → baca
me-	NULL	2	NULL	merusak → rusak
peng-	NULL	2	NULL	pengukur → ukur
peny-	s	2	V…	penyapu → sapu
pen-	NULL	2	NULL	penduga → duga
pen-	t	2	V…	penari → tari
pem-	p	2	V…	pemilah → pilah
pem-	NULL	2	NULL	pembaca → baca
di-	NULL	2	NULL	diukur → ukur
ter-	NULL	2	NULL	tersapu → sapu
ke-	NULL	2	NULL	kekasih → kasih

Tabel 2.4 Kelompok Aturan Keempat : Pembentukan Awalan Kedua

Awalan Kedua	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
ber-	NULL	2	NULL	berlari → lari
bel-	NULL	2	Ajar	belajar → ajar
be-	NULL	2	Kerja	bekerja → kerja
per-	NULL	2	NULL	perjelas → jelas
pel-	NULL	2	Ajar	pelajar → ajar
pe-	NULL	2	NULL	pekerja → kerja

Tabel 2.5 Kelompok Aturan Kelima : Pembentukan Akhiran

Akhiran	Pengganti	Kondisi Ukuran	Kondisi Tambahan	Contoh
-kan	NULL	2	Awalan bukan anggota {ke, peng}	tarikkan → tarik
-an	NULL	2	Awalan bukan anggota {di, meng, ter}	perjanjian → janji
-i	NULL	2	Awalan bukan anggota {ber, ke, peng}	mendapati → dapat

Kondisi ukuran adalah jumlah minimum suku kata dalam sebuah kata, karena dalam bahasa Indonesia kata dasar setidaknya mempunyai 2 suku kata.

Maka kondisi ukuran dalam proses *stemming* bahasa Indonesia adalah dua. Kondisi tambahan V dimaksudkan kata setelah awalan berupa huruf vokal.

Namun demikian, terdapat masalah dalam porter stemmer bahasa Indonesia yaitu struktur morfologi dalam bahasa Indonesia memiliki tingkat kerumitan yang lebih tinggi dari pada bahasa Inggris. Seperti misalnya, kesulitan dalam membedakan suatu kata yang mengandung imbuhan baik prefix maupun sufiks dengan suatu kata dasar yang salah satu suku katanya merupakan bagian dari imbuhan [14], contoh:

duduklah → duduk (dilakukan proses *stemming*)

sekolah → seko (dilakukan proses *stemming*, seharusnya tidak)

Untuk menangani masalah pada Porter Stemmer bahasa Indonesia perlu ditambahkan beberapa aturan dalam algoritma agar memberikan hasil yang lebih maksimal dan untuk mempermudah proses *stem* maka dibuatlah beberapa kamus kecil, antara lain sebagai berikut [14]:

a. Kamus kata dasar yang dilekati partikel, untuk menyimpan kata dasar yang memiliki suku kata terakhir (partikel infleksional) serta kata tersebut tidak mendapat imbuhan apapun.
   Seperti: masalah
b. Kamus kata dasar yang dilekati partikel berprefiks untuk menyimpan kata dasar yang memiliki suku kata terakhir (partikel infleksional) dan mempunyai prefiks.
   Seperti: menikah
c. Kamus kata dasar yang dilekati kata ganti milik, untuk menyimpan kata dasar yang memiliki suku kata terakhir (kata ganti infleksional) serta kata dasar tersebut tidak mendapatkan imbuhan apapun.
   Seperti: bangku.
d. Kamus kata dasar yang dilekati kata ganti milik berprefiks, untuk menyimpan kata dasar yangmemiliki suku kata terakhir (kata ganti infleksional) dan mempunyai prefiks.
   Seperti: bersuku.

e. Kamus kata dasar yang dilekati prefiks pertama, untuk menyimpan kata dasar yang memiliki suku kata pertama (prefiks derivasional pertama) serta kata dasar tersebut tidak mendapatkan imbuhan apapun.
   Seperti: median.
f. Kamus kata dasar yang dilekati prefiks pertama bersufiks, untuk menyimpan kata dasar yang memiliki suku kata pertama (prefiks derivasional pertama) dan mempunyai sufiks derivasional.
   Seperti: terapan.
g. Kamus kata dasar yang dilekati prefiks kedua, untuk menyimpan kata dasar yang memiliki suku kata pertama (prefiks derivasional kedua) serta kata dasar tersebut tidak mendapatkan imbuhan apapun.
   Seperti: percaya
h. Kamus kata dasar yang dilekati prefiks kedua bersufiks, untuk menyimpan kata dasar yang memiliki suku kata pertama (prefiks derivasional) dan mempunyai sufiks derivasional.
   Seperti: perasaan.
i. Kamus kata dasar yang dilekati sufiks, untuk menyimpan kata dasar yang memiliki suku kata terakhir (sufiks derivasional).
   Seperti: pantai

2.2.2 *Term Frequency-Invers Document Frequency* (TF-IDF)

Metode *Term Frequency-Inverse Document Frequency* (TF-IDF) adalah cara pemberian bobot hubungan suatu kata (*term*) terhadap dokumen. Untuk dokumen tunggal tiap kalimat dianggap sebagai dokumen. Metode ini menggabungkan dua konsep untuk perhitungan bobot, yaitu *Term frequency* (TF) merupakan frekuensi kemunculan kata (*t*) pada kalimat (*d*). *Document frequency* (DF) adalah banyaknya kalimat dimana suatu kata (*t*) muncul. Frekuensi kemunculan kata di dalam dokumen yang diberikan menunjukkan seberapa penting kata itu di dalam dokumen tersebut. Frekuensi dokumen yang mengandung kata tersebut menunjukkan seberapa umum kata tersebut. Bobot kata semakin besar jika sering muncul dalam suatu dokumen dan semakin kecil jika muncul dalam banyak dokumen [16]. Pada

aloritma TF-IDF digunakan rumus untuk menghitung bobot (W) masing masing dokumen terhadap kata kunci dengan rumus yaitu :

$$W_{dt} = tf_{dt} * IDF_t \tag{2.1}$$

Dimana:

d : Dokumen ke-d

t : Kata ke-t dari kata kunci

W : Bobot dokumen ke-d terhadap kata ke-t

tf : Banyaknya kata yang dicari pada sebuah dokumen

IDF : Inversed Document Frequency

IDF : log2 (D/df)

D : Total dokumen

df : Banyak dokumen yang mengandung kata yang dicari

Setelah bobot (W) masing-masing dokumen diketahui, maka dilakukan proses sorting/pengurutan dimana semakin besar nilai W, semakin besar tingkat similaritas dokumen tersebut terhadap kata kunci, demikian sebaliknya.

Inverse Document Frequency memperhatikan kemunculan *term* pada kumpulan dokumen. Pada metode ini, *term* yang dianggap bernilai/berharga adalah *term* yang jarang muncul pada koleksi/kumpulan dokumen [1]. Persamaan IDF adalah sebagai berikut:

$$IDF(t) = log \frac{N}{df(t)} \tag{2.2}$$

Dimana *df(t)* adalah banyak dokumen yang mengandung *term t*.

TF*IDF merupakan kombinasi metode TF dengan metode IDF. Sehingga persamaan TF*IDF adalah sebagai berikut:

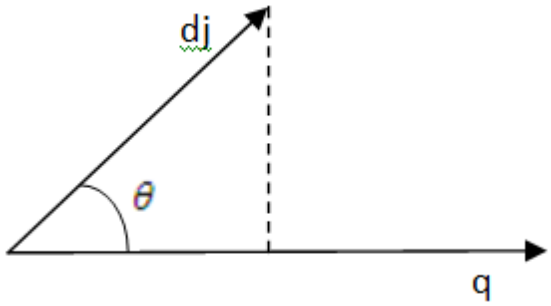
$$TF * IDF(d,t) = TF(d,t) * IDF(t) \tag{2.3}$$

Perhitungan bobot query relevance merupakan bobot hasil perbandingan kemiripan (similaritas) antara query yang dimasukkan oleh user terhadap keseluruhan kalimat. Sedangkan bobot similarity kalimat, merupakan bobot hasil perbandingan kemiripan antar kalimat.

2.2.3 *Vector Space Model*

Vector Space Model (VSM) adalah metode untuk melihat tingkat kedekatan atau kesamaan *(smilarity)* term dengan cara pembobotan term. Dokumen dipandang sebagi sebuah vektor yang memiliki *magnitude* (jarak) dan *direction* (arah). Pada *Vector Space Model*, sebuah istilah direpresentasikan dengan sebuah dimensi dari ruang vektor. Relevansi sebuah dokumen ke sebuah *query* didasarkan pada similaritas diantara vektor dokumen dan vektor *query* [1].

Sebuah dokumen (d_j) dan sebuah *query* (q) direpresentasikan sebagai vektor t-dimensi seperti pada Gambar 2.4.



Gambar 2.4 Vector Space Model

Cara kerja dari *vector space model* adalah dengan menghitung nilai cosines sudut dari dua vektor, yaitu vektor *query* terhadap vektor tiap dokumen. Proses perhitungan vektor space model melalui tahapan perhitungan jarak *query* menggunakan persamaan (2.4) dan dokumen, menggunakan persamaan (2.5).

$$|q| = \sqrt{\sum_{j=1}^{t}(W_{iq})^2} \tag{2.4}$$

Dengan $|q|$ adalah jarak *query*, dan $W_{i,q}$ adalah bobot *query* dokumen ke-i, maka jarak *query* ($|q|$) dihitung untuk didapatkan jarak *query* dari bobot *query* dokumen ($W_{i,q}$) yang terambil oleh sistem. Jarak *query* bisa dihitung dengan persamaan akar jumlah kuadrat dari *query*.

$$|d_j| = \sqrt{\sum_{i=1}^{t}(W_{ij})^2} \tag{2.5}$$

Dengan $|d_j|$ adalah jarak dokumen, dan W_{ij} adalah bobot dokumen ke-i, maka jarak dokumen ($|d_j|$) dihitung untuk didapatkan jarak dokumen dari bobot dokumen (W_{ij}) yang terambil oleh sistem. Jarak dokumen bisa dihitung dengan persamaan akar jumlah kuadarat dari dokumen.

Perhitungan pengukuran similiaritas *query document*, menggunakan persamaan (2.6).

$$\text{sim}(q, d_j) = \sum_{i=1}^{t} W_{iq} \, . W_{ij} \tag{2.6}$$

Dengan W_{ij} adalah bobot term dalam dokumen, W_{iq} adalah bobot query, dan Sim (q, d_j) adalah Similaritas antara *query* dan dokumen. Similaritas antara *query* dan dokumen atau *inner product/Sim* (q, d_j) digunakan untuk mendapatkan bobot dengan didasarkan pada bobot *term* dalam dokumen (W_{ij}) dan bobot query (W_{iq}) atau dengan cara menjumlah bobot *q* dikalikan dengan bobot dokumen.

Pengukuran *Cosine Similarity* (menghitung nilai kosinus sudut antara dua vector) menggunakan persamaan (2.7)

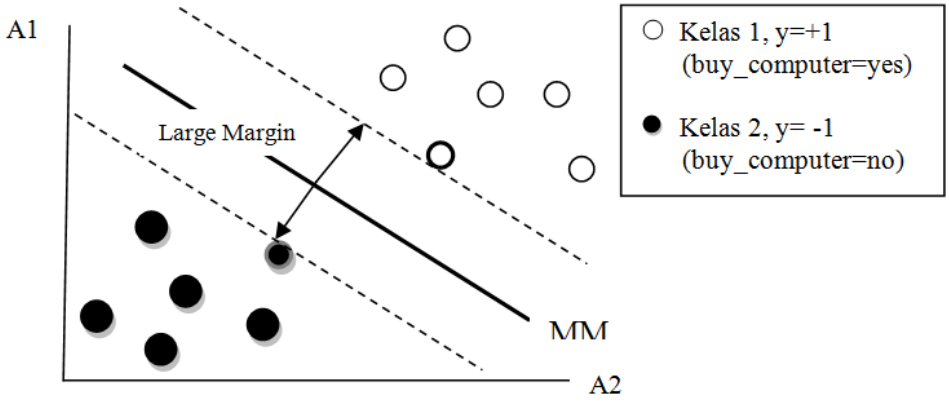
$$\text{sim}(q, d_j) = \frac{q - d_j}{|q| * |d_j|} = \frac{\sum_{i=1}^{t} W_{iq} \, . W_{ij}}{\sqrt{\sum_{j=1}^{t} (W_{iq})^2} * \sqrt{\sum_{i=1}^{t} (W_{ij})^2}} \tag{2.7}$$

Similaritas antara *query* dan dokumen atau $sim(q, d_j)$ berbanding lurus terhadap jumlah bobot query (q) dikali bobot dokumen (d_j) dan berbanding terbalik terhadap akar jumlah kuadrat q (|q|) dikali akar jumlah kuadrat dokumen ($|d_j|$).

2.2.4 *Support Vector Machine*

Support Vector Machine (SVM) disampaikan pada tahun 1992 oleh Vladimir Vapnik dan rekannya, Bernhard Boser dan Isabelle Guyon, meskipun dasar untuk SVM telah ada sejak 1960-an [2]. Metode *Support Vector Machine* (SVM) merupakan salah satu metode *machine learning* yang memaksimumkan akurasi prediksi dengan mencari bidang pembatas (*hyperplane*) terbaik dari dua kelas dalam ruang fitur**.** Metode SVM membutuhkan *training* set positif dan negatif. *Training* set positif dan negatif ini dibutuhkan SVM untuk membuat keputusan terbaik dalam memisahkan data positif dengan data negatif di ruang n-dimensi atau pembatas (*hyperplane*). Metode ini lebih dikenal dengan metode klasifikasi *supervised learning* untuk mencari garis pemisah *hyperplane* dengan

mengoptimalkan *hyperplane*, dan memaksimalkan *margin* antara dua kelas. Data yang paling dekat dengan bidang pembatas disebut *support vector* [17]. Metode ini menjadikan SVM sebagai metode baru yang menjanjikan untuk klasifikasi data, baik linier maupun nonlinier.



Gambar 2.5 Pemisahan Dua Kelas Data Dengan Margin Maksimum

Metode SVM dalam mengklasifikasikan kalimat ke dalam kelas positif dan negatif, digunakan untuk memilih kandidat kalimat yang akan dimasukkan ke dalam suatu ringkasan berupa rangkuman. Kalimat dengan nilai bobot *query relevance* > 0 diberi label positif, sedangkan kalimat dengan bobot *query relevance* = 0 diberi label negatif, hal ini dikarenakan kalimat tersebut tidak memiliki kerelevanan dengan *query*. Untuk kelas positif berlabel +1 yang artinya kalimat tersebut berpengaruh besar terhadap hasil ringkasan, sedangkan untuk kelas negatif berlabel -1 yang artinya kalimat tersebut tidak berpengaruh besar terhadap hasil ringkasan. Klasifikasi kalimat dengan metode ini dilihat dari data uji dengan vektor kalimat, jika nilai vektor melebihi nilai *hyperplane* maka kalimat tersebut masuk ke dalam kelas positif dan jika tidak maka kalimat tersebut masuk ke dalam kelas negatif.

Data pada ruang input (*input space*) berdimensi d dinotasikan dengan $x_i = \in \Re^d$ sedangkan label kelas dinotasikan dengan $y_i \in \{-1, +1\}$ untuk i = 1, 2,…,n. Dimana n adalah banyaknya data. Diasumsikan kedua kelas -1 dan +1 dapat terpisah secara linear bidang pembatas [17], maka persamaan bidang pembatasnya didefinisikan pada persamaan (2.8) berikut:

$$w * x_i + b = 0 \quad (2.8)$$

Data x_i yang terbagi ke dalam dua kelas, yang termasuk kelas -1 (sampel negatif) didefinisikan sebagai vektor yang memenuhi pertidaksamaan (2.9) berikut:

$$w * x_i + b < 0 \text{ untuk } y_i = -1 \quad (2.9)$$

Sedangkan yang termasuk kelas +1 (sampel positif) memenuhi pertidaksamaan (2.10) berikut:

$$w * x_i + b > 0 \text{ untuk } y_i = +1 \quad (2.10)$$

Dimana:

x_i = data *input*

y_i = label yang diberikan

w = nilai dari bidang normal

b = posisi bidang relatif terhadap pusat koordinat

Parameter w dan b adalah parameter yang akan dicari nilainya. Bila label data y_i = -1, maka pembatas menjadi persamaan (2.11) berikut:

$$w * x_i + b \leq -1 \quad (2.11)$$

Bila label data y_i = +1, maka pembatas menjadi persamaan (2.12) berikut:

$$w * x_i + b \geq +1 \quad (2.12)$$

Margin terbesar dapat dicari dengan cara memaksimalkan jarak antar bidang pembatas kedua kelas dan titik terdekatnya, yaitu 2/|w|. Hal ini dirumuskan sebagai permasalahan *quadratic programming* (QP) *problem* yaitu mencari titik minimal persamaan (2.13) dengan memperhatikan persamaan (2.14) berikut:

$$min\ \tau(w) = \frac{1}{2}||w||^2 \quad (2.13)$$

$$y_i(w * x_i + b) - 1 \geq 0, (i = 1, .., n) \quad (2.14)$$

Permasalahan ini ini dapat dipecahkan dengan berbagi teknik komputasi. Lebih mudah diselesaikan dengan mengubah persamaan (2.13) ke dalam fungsi *Lagrangian* pada persamaan (2.15), dan menyederhanakannya menjadi persamaan (2.16) berikut:

$$L(w, b, a) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} a_i(y_i\big((w^T x_i + b) - 1\big)\big) \quad (2.15)$$

$$L(w,b,a) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} a_i y_i (w^T x_i + b) + \sum_{i=1}^{n} a_i \quad (2.16)$$

Dimana a_i adalah *lagrange multiplier* yang bernilai nol atau positif ($a_i \geq 0$). Nilai optimal dari persamaan (2.16) dapat dihitung dengan meminimalkan L terhadap w, b dan a. Dapat dilihat pada persamaan (2.17) sampai (2.19) berikut:

$$\frac{\partial L}{\partial w} = w - \sum_{i=1}^{n} a_i \, y_i x_i = 0 \quad (2.17)$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{n} a_i \, y_i = 0 \quad (2.18)$$

$$\frac{\partial L}{\partial a} = \sum_{i=1}^{n} a_i y_i (w^T x_i + b) - \sum_{i=1}^{n} a_i = 0 \quad (2.19)$$

Maka masalah *Lagrange* untuk klasifikasi dapat dinyatakan pada persamaan (2.20) berikut:

$$Min\ L(w,b,a) = \frac{1}{2}||w||^2 - \sum_{i=1}^{n} a_i y_i (w^T x_i + b) - \sum_{i=1}^{n} a_i \quad (2.20)$$

Dengan memperhatikan persamaan (2.21) dan (2.22) berikut:

$$w - \sum_{i=1}^{n} a_i \, y_i x_i = 0 \quad (2.21)$$

$$\sum_{i=1}^{n} a_i y_i = 0 \quad (2.22)$$

Model persamaan (2.20) diatas merupakan model primal *Lagrange*. Sedangkan dengan memaksimalkan L terhadap a_i, persamannya menjadi persamaan (2.23) berikut:

$$Max \sum_{i=1}^{n} a_i - \frac{1}{2} \sum_{i=1,j=1}^{n} a_i a_j y_i {y_j}^T x_i x_j^T \quad (2.23)$$

Dengan memperhatikan persamaan (2.24) berikut:

$$\sum_{i=1}^{n} a_i y_i = 0,\ a_i \geq 0\ (i,j = 1,..,n) \quad (2.24)$$

Untuk mencari nilai x_i dan y_i dapat dilakukan ketika sudah didapatkan nilai tiap kata (*term*) dari pembobotan tf-idf dan perhitungan *cosine similarity*. Hasil dari pembobotan tf-id diubah ke dalam bentuk format data svm, sedangkan hasil perhitungan *cosine similarity* menjadi label data svm.

Untuk mendapatkan nilai a_i, langkah pertama adalah mengubah setiap kalimat menjadi nilai vektor (*support vector*) = $\binom{x}{y}$. Kemudian nilai vektor dari setiap kalimat dimasukkan ke persamaan (2.25) kernel *trick phi* berikut:

$$K_i = \phi \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \sqrt{x_n^2 + y_n^2} - x + (x-y)^2 \\ \sqrt{x_n^2 + y_n^2} - y + (x-y)^2 \end{bmatrix} \quad (2.25)$$

Nilai x didapatkan dari persamaan (2.26) kernel linear untuk x berikut:

$$\sum_{i=1,j=1}^{1} x_i x_j^T, (i,j = 1,..,n) \quad (2.26)$$

Nilai y didapatkan dari persamaan (2.27) kernel linear untuk y berikut:

$$\sum_{i=1,j=1}^{1} y_i y_j^T, (i,j = 1,..,n) \quad (2.27)$$

Untuk mendapatkan jarak tegak lurus yang optimal dengan mempertimbangkan vektor positif, maka hasil perhitungan dari substitusi nilai x dan nilai y ke persamaan (2.25) diberi nilai bias = 1 [17]. Kemudian cari parameter a_i, dengan terlebih dahulu mencari nilai fungsi setiap kalimat menggunakan persamaan (2.28), lalu mencari nilai a_i pada persamaan linear menggunakan persamaan (2.29) dengan memperhatikan $i,j = 1,..,n$ berikut:

$$\sum_{i=1,j=1}^{n} a_i K_i^T K_j \quad (2.28)$$

$$\sum_{i=1,j=1}^{n} a_i K_i^T K_j = y_i \quad (2.29)$$

Setelah parameter a_i didapatkan, kemudian masukkan ke persamaan (2.30) berikut:

$$\widetilde{W} = \sum_{i=1}^{n} a_i K_i \quad (2.30)$$

Hasil yang didapatkan menggunakan persamaan (2.30), selanjutnya digunakan persamaan (2.31) untuk mendapatkan nilai w dan b:

$$y = wx + b \quad (2.31)$$

Sedemikian sehingga didapatkanlah nilai w dan nilai b atau nilai *hyperplane* untuk mengklasifikasikan kedua kelas. Hasil dari klasifikasi kalimat menggunakan metode SVM akan digunakan untuk memilih kandidat kalimat ringkasan.

2.3 Evaluasi Ringkasan

Metode untuk mengevaluasi hasil ringkasan merupakan topik yang cukup sulit, baik evaluasi terhadap ringkasan yang dihasilkan dari mesin peringkas otomatis ataupun ringkasan yang manual dibuat oleh *abstractor* yang profesional, dikarenakan tidak terdapat definisi ringkasan ideal [18]. Secara umum terdapat dua metode untuk mengevaluasi hasil dari ringkasan [9], yaitu :

1. Intrinsik

   Metode evaluasi intrinsik mengukur kualitas sistem peringkas teks itu sendiri yang difokuskan pada seberapa koheren dan informatif dari suatu ringkasan, biasanya dibandingkan dengan bantuan ringkasan yang dibuat oleh manusia.

2. Ekstrinsik

   Metode ekstrinsik mengukur bagaimana pengaruhnya (efektivitas dan akseptabilitas) hasil ringkasan tersebut pada sistem lain misalnya pencarian dokumen, *indexing*, atau klasifikasi dokumen.

Metode evaluasi yang digunakan pada penelitian ini adalah metode intrinsik dengan *Recall*, *Precision* dan *F-Measures. Recall*, *Precision* dan *F-Measures* merupakan perhitungan nilai performansi yang berhubungan. Adapun pengertian beserta rumus dari perhitungan *recall*, *precision* dan *f-measure* untuk kasus mengevaluasi ringkasan dengan parameter [18]:

1. *Correct* merupakan jumlah kalimat yang berhasil di ekstrak sistem sesuai dengan kalimat yang diekstrak manusia.
2. *Wrong* merupakan jumlah kalimat yang diekstrak sistem tetapi tidak terdapat dalam kalimat yang diekstrak manusia
3. *Missed* merupakan jumlah kalimat yang diekstrak manusia tetapi tidak terdapat dalam kalimat yang diekstrak sistem.

2.3.1 *Recall*

Recall (r) merupakan istilah yang digunakan untuk kalimat terpanggil yang relevan dengan pernyataan atau vektor *query*. Perbandingan jumlah informasi relevan yang didapatkan sistem dengan jumlah seluruh informasi relevan yang ada dalam koleksi informasi. *Recall* berhubungan dengan kemampuan dalam menemukan kalimat yang relevan. Hal ini berarti *recall* adalah bagian dari proses evaluasi yang dapat digunakan sebagai alat ukur relevan sebuah ringkasan. Adapun rumus untuk menghitung *recall* dapat dilihat pada persamaan (2.31) berikut:

$$r = \frac{correct}{correct+missed} \tag{2.32}$$

2.3.2 *Precision*

Precision (p) biasanya menjadi salah satu ukuran yang digunakan untuk menilai evektivitas kalimat yang di hasilkan dalam sebuah rangkuman. *Precision*

adalah jumlah kalimat relevan dari total jumlah kalimat yang ditemukan oleh sistem. Perbandingan jumlah informasi relevan yang didapatkan sistem dengan jumlah seluruh informasi yang terambil oleh sistem baik yang relevan maupun tidak. Adapun, pengukuran tingkat ketepatan (*precision*) dirumuskan pada persamaan (2.31) berikut:

$$p = \frac{correct}{correct+wrong} \tag{2.33}$$

2.3.3 *F-Measure*

F-Measure (F_1) merupakan salah satu perhitungan evaluasi yang mengkombinasikan *recall* dan *precision*. Hubungan antara *recall* dan *precision* yang mempresentasikan akurasi sistem. Nilai *recall* dan *precision* pada suatu keadaan dapat memiliki bobot yang berbeda. Ukuran yang menampilkan timbal balik antara *recall* dan *precision* adalah *f-measure* yang merupakan bobot *harmonic mean* dari *recall* dan *precision*. Perhitungan *f-measure* dapat dirumuskan pada persamaan (2.32) berikut:

$$F_1 = \frac{2*precision*recall}{precission+recall} \tag{2.34}$$

2.4 Analisis Terstruktur

Analisis terstruktur akan dibahas tentang pengertian dan perangkat pemodelan analisis terstruktur.

2.4.1 Pengertian Analisis Terstruktur

Analisis Terstruktur (Structured Analysis), merupakan salah satu teknik analisis yang mengunakan pendekatan berorientasi fungsi. Analisis ini terfokus pada aliran data dan proses bisnis perangkat dan perangkat lunak. Analisis ini disebut process oriented. Analisis struktur sederhana dalam konsep [5].

Teknik ini mempunyai sekumpulan petunjuk dan perangkat komunikasi grafis yang memungkinkan analis sistem mendefinisikan spesifikasi fungsional perangkat lunak secara terstruktur. Pada metode ini, semua fungsi sistem direpresentasikan sebagai sebuah proses transformasi informasi, dan disusun secara hirarkis sesuai tingkat abstraksinya (sistem maupun perangkat lunak) yang hasilnya ditujukan untuk entitas-entitas eksternal. Prinsip dari teknik ini adalah dekomposisi

fungsi dari sistem berdasarkan aliran data dan proses-prosesnya untuk mendapatkan produk analisis yang dapat diubah dan diperbaiki secara mudah (highly maintainable). Dalam bukunya itu, DeMarco mendefinisikan Analisis Terstruktur sebagai teknik untuk mendeskripsikan spesifikasi sistem baru melalui Data Flow Diagrams, Data Dictionary, Structured English, dan Data Structure Diagrams. Spesifikasi sistem tersebut dinyatakan dalam suatu dokumen yang disebut Spesifikasi Terstuktur (Structured Specification).

2.4.2 Perangkat Pemodelan Analisis Terstruktur

Perangkat Pemodelan Analisis Terstruktur adalah alat bantu pemodelan yang digunakan untuk menggambarkan hasil pelaksanaan Analisis Terstruktur. Perangkat Analisis Terstruktur antara lain adalah :

1. Diagram Konteks

Diagram konteks adalah diagram yang terdiri dari suatu proses dan menggambarkan ruang lingkup suatu sistem. Diagram konteksmerupakan level tertinggi dari DFD yang menggambarkan seluruh input ke sistem atau output dari sistem. Ia akan memberi gambaran tentang keseluruhan sistem. Sistem dibatasi oleh boundary (dapat digambarkan dengan garis putus). Dalam diagram konteks hanya ada satu proses. Tidak boleh ada store dalam diagram konteks. Diagram konteks berisi gambaran umum (secara garis besar) sistem yang akan dibuat. Secara kalimat, dapat dikatakan bahwa diagram konteks ini berisi siapa saja yang memberi data (dan data apa saja) ke sistem, serta kepada siapa saja informasi (dan informasi apa saja) yang harus dihasilkan sistem.

2. Diagram Aliran Data atau *Data Flow Diagram* (DFD)

Diagram untuk menggambarkan aliran data dalam sistem, sumber dan tujuan data, proses yang mengolah data tersebut, dan tempat penyimpanan datanya. Keuntungan dari diagram arus data adalah memungkinkan pengembangan sistem dari level yang paling tinggi dan memecah menjadi level yang lebih rendah.Beberapa simbol yang digunakan DFD adalah sebagai berikut :

a. Entitas Luar (*External Entity*)

Suatu yang berada diluar sistem, tetapi ia memberikan data kedalam sistem atau memberikan data dari sistem, disimbolkan dengan suatu kotak notasi.

External Entity tidak termasuk bagian dari sistem. Bila sistem informasi dirancang untuk suatu bagian lain yang masih terkait menjadi external entity.

b. Proses (*Process*)

Proses merupakan apa yang dikerjakan oleh sistem. Proses dapat mengolah data atau aliran data masuk menjadi aliran data keluar. Proses berfungsi mentransformasikan suatu atau beberapa data keluaran sesuai dengan spesifikasi yang diinginkan. Setiap proses memiliki satu atau beberapa masukan serta menghasilkan satu atau beberapa data kelurahan. Proses sering juga disebut bubble.

c. Arus Data (*Data Flow*)

Arus data merupakan tempat mengalirnya informasi dan digambarkan dengan garis yang menghubungkan komponen dari sistem. Arus data ditunjukan dengan arah panah dan garis diberi nama atas arus data yang mengalir. Arus data ini mengalir diantara proses, data store dan menunjukan arus data dari data yang berupa masukan untuk sistem atau hasil proses sistem.

d. Simpanan Data (*Data Store*)

Simpanan data merupakan tempat penyimpanaan data yang ada dalam sistem. Data store dapat disimbolkan dengan dua garis sejajar atau dua garis dengan salah satu sisi samping terbuka. Proses dapat mengambil data dari atau memberikan data ke simpanan data (database).

3. Kamus Data (*Data Dictionary*)

Merupakan suatu tempat penyimpanan (gudang) dari data dan informasi yang dibutuhkan oleh suatu sistem informasi. Kamus data digunakan untuk mendeskripsikan rincian dari aliran data atau informasi yang mengalir dalam sistem, elemen-elemen data, file maupun basis data (tempat penyimpanan) dalam DFD.

4. Diagram Entitas-Relasi atau Entity-Relationship Diagram (ERD)

ERD merupakan notasi grafis dalam pemodelan data konseptual yang mendeskripsikan hubungan antara penyimpanan. ERD digunangan untuk memodelkan struktur data dan hubungan antar data, karena hal ini relative kompleks. Dengan ERD kita dapat menguji model dengan mengabaikan proses yang dilakukan. ERD menggunakan sejumlah notasi dan simbol untuk

menggambarkan struktur dan hubungan antar data, pada dasarnya ada 3 macam simbol yang digunakan:

a. Entity

Entity adalah suatu objek yang dapat di identifikasi dalam lingkaran pemakai, sesuatu yang penting bagi pemakai dalam konteks sistem yang akan dibuat.

b. Atribut

Entity mempunyai elemen yang disebut atribut dan berfungsi mendeskripsikan karakter entity.

c. Relasi

Relasi merupakan gugusan entitas yang berhubungan antar entitas atau beberapa entitas.

2.5 *Software* Pendukung

Pada penelitian yang dilakukan diperlukan *software* pendukung yang membantu dalam pembangunan aplikasi. Adapun *software* pendukung, diantaranya:

2.5.1 Bahasa Pemrograman C# (C Sharp)

C# (C sharp) merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET Framework. Bahasa pemrograman ini dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya seperti Java, Delphi, Visual Basic, dan lain-lain) dengan beberapa penyederhanaan. Menurut standar ECMA-334 C# *Language Specification*, nama C# terdiri atas sebuah huruf Latin C (U+0043) yang diikuti oleh tanda pagar yang menandakan angka # (U+0023). Tanda pagar # yang digunakan memang bukan tanda kres dalam seni musik (U+266F), dan tanda pagar # (U+0023) tersebut digunakan karena karakter kres dalam seni musik tidak terdapat di dalam keyboard standar.

2.5.2 *Microsoft Visual Studio*

Microsoft Visual Studio merupakan sebuah perangkat lunak yang dapat digunakan untuk melakukan pembuatan maupun pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya, dengan bentuk aplikasi *console*, aplikasi *windows*, ataupun aplikasi *web*. *Visual Studio* mencakup *kompiler, SDK, Integrated Development Environment (IDE)*, dan dokumentasi (umumnya berupa *MSDN Library*). Kompiler yang dimasukkan ke dalam paket Visual Studio antara lain adalah : *Visual C++, Visual C#, Visual Basic, Visual Basic .NET, Visual InterDev, Visual J++, Visual J#, Visual FoxPro, dan Visual SourceSafe*.



Gambar 2.6 Logo *Microsoft Visual Studio*

2.5.3 *DevExpress Universal*

DevExpress Universal merupakan *tools* yang dapat digunakan sebagai editor untuk tampilan pada aplikasi yang akan dibuat. Menu *ribbon* seperti *office 2007/2010, autohide docking panel, report* dan *preview* (termasuk *export* ke berbagai file format), *datagrid* dengan berbagai macam *view, auto seach, auto sort* dan *drag drop auto grouping* serta masih banyak lagi fitur yang lainnya yang disediakan pada *tools* ini.

2.5.4 MySql

MySQL merupakan sistem manajemen basis data SQL yang sangat terkenal dan bersifat *open source*. MySQL dibangun, didistribusikan, dan didukung oleh MySQL AB. MySQL AB merupakan perusahaan komersial yang dibiayai oleh pengembang MySQL.MySQL mempunyai dua macam lisensi yaitu lisensi yang bersifat *open source* dengan menggunakan GNU *General Public License* dan lisensi kedua berupa *Standard Commercial License*.

Dalam konteks bahasa SQL, informasi disimpan dalam tabel-tabel yang secara logis merupakan struktur dua dimensi yang tersimpan atas baris-baris data

(*row* atau *record*) yang berada dalam satu atau lebih kolom (*column*). Baris pada tabel disebut *insance* dari data sedangkan kolom sering disebut sebagai *atributes* atau *field*y.