BAB 2
TINJAUAN PUSTAKA

Bab tinjuan pustaka berisi tentang tinjauan tempat penelitian, konsep dasar aplikasi *text mining*, penentuan indeks, metode tf-idf, *vector space model*, konsep berorientasi objek, basis data, tinjauan perangkat lunak, dan bahasa pemrograman JAVA yang digunakan untuk membangun aplikasi.

2.1 Tinjauan Tempat Penelitian

PT. eBdesk didirikan pada tahun 1998, sebagai perusahaan produsen perangkat lunak yang bergerak di bidang teknologi informasi. Ebdesk telah mengembangkan produk portal perusahaan, berdasarkan teknologi *internet*. Ebdesk tetap fokus untuk mengembangkan produk-produk yang berkelanjutan sesuai dengan perubahan teknologi dan tren. Ebdesk adalah salah satu perusahaan pertama yang mengembangkan *corporate portal*, yang mana *corporate portal* itu sendiri adalah sebuah konsep baru ketika eBdesk didirikan.

eBdesk Corporate Portal mengintegrasikan berbagai informasi dan aplikasi, dari perusahaan di dalam atau di luar, di antarmuka web pribadi tunggal. Ini adalah platform untuk menyebarkan sepenuhnya *portal web interface* yang memberikan koneksi mulus antara karyawan, manajer, pemilik, mitra, dan pelanggan.

PT. eBdesk adalah anggota Asia Software Alliance (SAA), merupakan koalisi dari pemenang-pemenang penghargaan perusahaan software dan jasa. SAA adalah aliansi berbasis ICT global pertama dari jenisnya yang menawarkan solusi

yang komprehensif *end-to-end* dan layanan untuk berbagai segmen industri. Saat ini ada 6 anggota SAA termasuk XYBASE (Malaysia), eBdesk (Indonesia), Patimas (Malaysia), TMS, SSC Solution (Thailand), BlueBridge Software.

Saat ini PT eBdesk sedang mengembangkan sebuah produk bernama eBdesk Collaboration, yaitu berupa sebuah aplikasi yang memungkinkan penggunanya berkolaborasi untuk mencapai suatu tujuan. Beberapa fitur utama dalam aplikasi eBdesk Collaboration adalah *sharing* dokumen, galeri, blog dan forum.

Dalam aplikasi eBdesk Collaboration ini terdapat fasilitas untuk menyimpan dokumen. Untuk setiap dokumen bisa terdiri dari satu atau beberapa *file* yang di-*upload* kedalam aplikasi, disimpan pula ke dalam *database* berupa judul dan deskripsi singkat mengenai dokumen yang dimasukkan oleh pengguna. Proses pencarian dokumen yang dilakukan pada aplikasi eBdesk Collaboration saat ini baru sebatas mencari dokumen berdasarkan judul dan deskripsi singkat yang disimpan ke dalam *database*.

Untuk lebih menambah nilai jual dari aplikasi eBdesk Collaboration ini, oleh karena itu ingin dibuatkan sebuah fitur baru yaitu pencarian dokumen yang bisa menelusuri *query* pencarian sampai ke dalam isi *file* yang di-*upload* nya itu sendiri.

2.2 Istilah-istilah yang Digunakan Dalam Penulisan

a. *Term* : kata, frase

b. *File* : berkas tertulis (surat, akta, dsb) dari waktu yang lampau, disimpan kedalam media elektronik, disimpan dan dipelihara di tempat khusus untuk referensi. Contoh : Kontrak Kerja PT A.docx
c. Dokumen : kumpulan / gabungan dari *file*. Sebagai contoh, dokumen pendukung skripsi, didalamnya terdapat *file-file* : Jadwal Kegiatan Proposal.docx, Kerangka Penulisan Skripsi.docx, Sistem Evaluasi dan Seleksi Proposal.docx, Syarat Pengajuan Proposal.docx, dll. rapkan dapat memenuhi keinginan pengguna dari kumpulan *file* yang ada
d. *Query*: pertanyaan yang dimasukkan pengguna ke dalam aplikasi untuk mencari *file*

2.3 Text Mining

Text mining adalah salah satu bidang khusus dari *data mining*. *Text mining* dapat didefinisikan sebagai suatu proses menggali informasi dari data yang berupa teks dimana sumber data biasanya didapatkan dari dokumen. (Khodra, 2003). Tujuan dari *text mining* adalah untuk mendapatkan informasi yang berguna dari sekumpulan dokumen. Jadi, sumber data yang digunakan pada *text mining* adalah kumpulan teks yang memiliki format yang tidak terstruktur atau minimal semi terstruktur.

Algoritma yang digunakan pada *text mining*, biasanya tidak hanya melakukan perhitungan hanya pada dokumen, tetapi juga pada fitur. Empat macam fitur yang sering digunakan:

a. *Character* merupakan komponen individual, bisa huruf, angka, karakter spesial dan spasi, merupakan *block* pembangun pada level paling tinggi

pembentuk semantik feature, seperti kata, term dan concept.Pada umumnya, representasi *character-based* ini jarang digunakan pada beberapa teknik pemrosesan teks.

b. *Words* adalah satuan kata yang digunakan untuk diproses.
c. *Terms* merupakan *single word* dan frasa *multiword* yang terpilih secara langsung dari corpus. Representasi *term-based* dari dokumen tersusun dari subset term dalam dokumen.
d. *Concept* merupakan *feature* yang digenerate dari sebuah dokumen secara manual, *rule-based*, atau metodologi lain. Pada tugas akhir ini, konsep digenerate dari argumen atau kata benda yang sudah diberi label pada suatu dokumen.

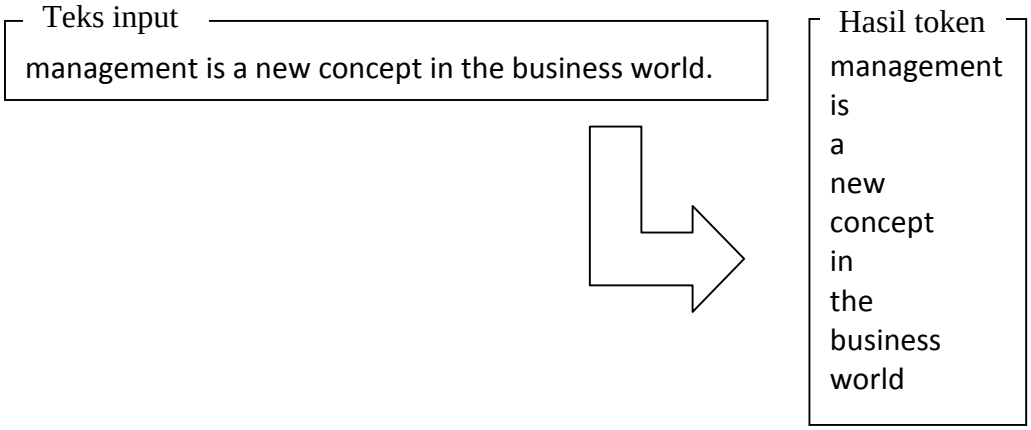
Text mining bisa juga diartikan sebagai proses analisis teks untuk menemukan informasi baru dari sekumpulan teks berbahasa alami yang tidak terstruktur. Singkatnya adalah pencarian pola tertentu pada suatu text. Dalam text mining ada 5 tahapan umum yang biasanya terdapat pada *text mining* yaitu *tokenizing, filtering, stemming, tagging* dan *Analyzing.* (Harlian, 2006)

2.4 Proses Penentuan Indeks

Terdapat beberapa tahapan untuk melakukan proses penentuan indeks sampai tersimpan didalam basis data yaitu menyimpan ID *file* dan menyimpan *term index* melalui memecah isi *file* kedalam *tokenizing* (*break into token*), melakukan *filtering, stemming, tagging,* dan *term weighting*. Langkah-langkah yang dilakukan dalam proses penentuan index adalah :

a. Tokenizing

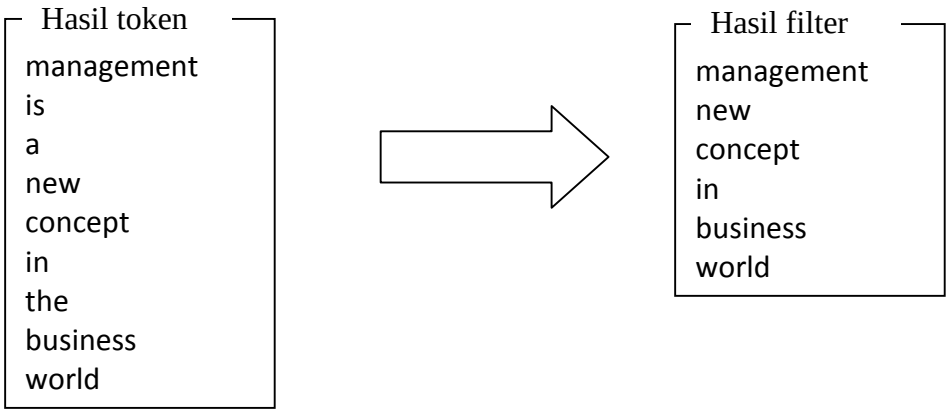
Merupakan proses memecah (*parsing*) *file* menjadi *token-token* yaitu dengan memotong menjadi *term*. Contoh hasil *tokenizing* terlihat pada Gambar 2.1 sebagai berikut :



Gambar 2.1 Hasil Tokenizing [Milkha, 2006]

b. Filtering

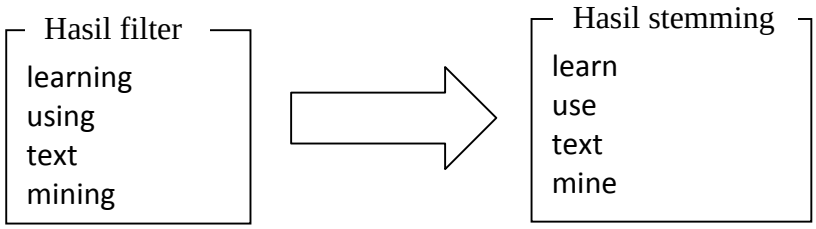
Merupakan proses menghilangkan kata-kata umum yang sering ditampilkan dalam *file* seperti : *is*, *for*, *in*, dan sebagainya. Contoh hasil *filtering* terlihat pada Gambar 2.2 sebagai berikut :



Gambar 2.2 Hasil *Filtering* [Milkha, 2006]

c. Stemming

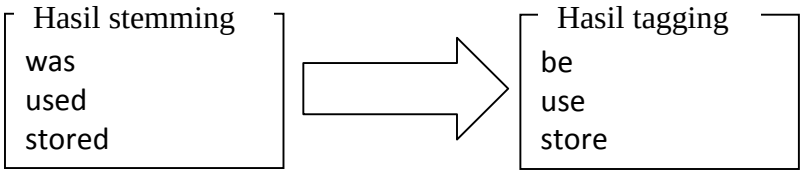
Merupakan proses pembuangan *prefix* (awalan) dan *suffix* (imbuhan akhiran) secara morfologi dari suatu kata berimbuhan menjadi kata dasar. Sebelum dilakukan proses *stemming*, terlebih dahulu dilakukan pengecekan kedalam kamus kata dasar. Kata yang tidak ada dalam kamus akan dilakukan proses *stemming*. Contoh hasil *stemming* terlihat pada Gambar 2.3 sebagai berikut :



Gambar 2.3 Hasil *Stemming* [Milkha, 2006]

d. Tagging

Tahapan *tagging* adalah tahap mencari bentuk awal/root kata dari tiap kata dalam bentuk lampau atau kata hasil *stemming*. Contoh hasil *tagging* terlihat pada Gambar 2.4 sebagai berikut :



Gambar 2.4 Hasil *Tagging* [Milkha, 2006]

e. Term weighting

Merupakan proses pembobotan *term* yang ada di *file* (seberapa sering kemunculan *term* pada *file*).

Contoh dari proses penentuan indeks bisa dilihat pada ilustrasi dibawah ini :

1. Teks yang diproses adalah ”Text mining, sometimes alternately referred to as text data mining”
2. Dilakukan proses *tokenizing* terhadap teks, maka dihasilkan daftar kata-kata sebagai berikut :

text
mining
sometimes
alternately
referred
to
as
text
data
mining

3. Setelah proses *tokenizing*, dilakukan proses *filtering* yang berfungsi untuk menghilangkan kata-kata umum. Dalam kasus ini yang menjadi kata umum adalah ”as” dan ”to”. Daftar kata-kata umum bisa dilihat pada lampiran. Setelah proses *filtering* maka dihasilkan daftar kata-kata sebagai berikut :

text
mining
sometimes
alternately
referred
text
data
mining

4. Selanjutnya dilakukan proses *stemming*. Sehingga dihasilkan hasil proses dari *stemming* sebagai berikut :

text
mine
sometim
altern

refer
text
data
mine

5. Selain dilakukan proses *stemming*, untuk kata yang tidak ada dalam kamus maka dilakukan proses *tagging*. Sehingga dihasilkan daftar sebagai berikut:

text
mine
sometim
altern
refer
text
data
mine

6. Proses terakhir adalah menghitung *term weighting* dari kata-kata yang telah diproses. Sehingga dihasilkan hasil perhitungannya sebagai berikut :

Kata	Frekuensi
text	2
mine	2
sometim	1
altern	1
refer	1
data	1

2.4.1 Porter Stemmer

Algoritma yang paling umum digunakan untuk melakukan stemming bahasa Inggris dan efektif digunakan adalah algoritma Porter (Porter 1980). Algoritma porter terdiri dari 5 tahap pemotongan kata, yang digunakan secara sekuensial. Dalam tiap tahap tersebut ada berbagai ketentuan untuk memotong kata. Aturan-aturan tersebut disimbolkan sebagai berikut :

1. m, biasa disebut ukuran. Bisa diartikan lebih jauh bahwa m adalah jumlah kemunculan huruf vokal yang diikuti dengan konsonan.

```
Algoritma untuk menghitung m :

Function countM(input term: array[] of char) → integer
{mengembalikan nilai m}
  Deklarasi
      m, N : integer
      lastIsVowel : boolean

  Begin
      m ← 0
      N ← Arraysize(term)
      lastIsVowel ← false
      If(N < 2)
            return 0

      For(i = 1 to N)
          if(term[i] != 'a' && term[i] != 'i' && term[i] != 'u' &&  term[i] != 'e' &&
          term[i] != 'o')
              then
              begin
                    if(lastIsVowel) then
                          m ← m + 1
                    lastIsVowel ← false
              end
              else lastIsVowel ← true
      Endfor
      return m
End;
```

Contoh :

a. m akan bernilai 0 jika *term* yang dimasukkan kedalam fungsi adalah TR, EE, TREE, Y, BY
b. m akan bernilai 1 jika *term* yang dimasukkan kedalam fungsi adalah TROUBLE, OATS, TREES, IVY
c. m akan bernilai 2 jika *term* yang dimasukkan kedalam fungsi adalah TROUBLES, PRIVATE, OATEN, ORRERY

2. *S adalah aturan bahwa kata diakhiri dengan huruf S
3. *v* adalah di dalam kata mengandung huruf vokal
4. *d adalah akhiran kata merupakan konsonan rangkap (*double*)
5. *o adalah kata berakhiran KVK (konsonan - vokal - konsonan), dimana huruf konsonan yang kedua bukan W, X, atau Y

Aturan dalam menghapus akhiran dituliskan dengan : (kondisi) S1 -> S2. Ini berarti jika kata berakhiran S1, dan sebelum S1 memenuhi kondisi yang diberikan maka S1 diganti dengan S2.

Tahap-tahap pada algoritma Porter Stemmer adalah sebagai berikut :

1. Menghapus akhiran dari kata jamak
    a. SSES -> SS (caresses)
    b. IES -> I (ponies)
    c. SS -> SS (caress)
    d. S -> (cats)
2. Menghapus infleksi lisan
    a. (m>0) EED -> EE (agreed, feed)
    b. (*v*) ED -> (plastered, bled)
    c. (*v*) ING -> (motoring, sing)
3. Jika melakukan penghapusan ED atau ING pada tahap ke 2, maka dilanjutkan dengan tahap ini
    a. AT -> ATE (conflat*ed*)
    b. BL -> BLE (troubl*ed*)

c. IZ -> IZE (siz*ed*)

d. (*d dan bukan (*L atau *S atau *Z)) -> (tann)

e. (m=1 dan *o) -> E (fil*ing*, fail*ing*, slow*ing*)

4. Mengganti akhiran Y dengan I

   (*v*) Y -> I (happy, sky)

5. Menghapus satu akhiran dari beberapa akhiran

   a. (m>0) ATIONAL -> ATE (relational)

   b. (m>0) TIONAL -> TION (conditional, rational)

   c. (m>0) ENCI -> ENCE (valenci)

   d. (m>0) ANCI -> ANCE (hesitanci)

   e. (m>0) IZER -> IZE (digitizer)

   f. (m>0) ABLI -> ABLE (conformabli)

   g. (m>0) ALLI -> AL (radically)

   h. (m>0) ENTLI -> ENT (differently)

   i. (m>0) ELI -> E (vilely)

   j. (m>0) OUSLI -> OUS (analogously)

   k. (m>0) IZATION -> IZE (vietnamization)

   l. (m>0) ATION -> ATE (predication)

   m. (m>0) ATOR -> ATE (operator)

   n. (m>0) ALISM -> AL (feudalism)

   o. (m>0) IVENESS -> IVE (decisiveness)

   p. (m>0) FULNESS -> FUL (hopefulness)

   q. (m>0) OUSNESS -> OUS (callousness)

r. (m>0) ALITI -> AL (formaliti)

s. (m>0) IVITI -> IVE (sensitiviti)

t. (m>0) BILITI -> BLE (sensibility)

6. Menghapus satu akhiran dari beberapa akhiran

a. (m>0) ICATE -> IC (triplicate)

b. (m>0) ATIVE -> (formative)

c. (m>0) ALIZE -> AL (formalize)

d. (m>0) ICITI -> IC (electriciti)

e. (m>0) ICAL -> IC (electrical)

f. (m>0) FUL -> (hopeful)

g. (m>0) NESS -> (goodness)

7. Menghapus akhiran terakhir

a. (m>1) AL -> (revival)

b. (m>1) ANCE -> (allowance, dance)

c. (m>1) ENCE -> (inference, fence)

d. (m>1) ER -> (airliner, employer)

e. (m>1) IC -> (gyroscopic, electric)

f. (m>1) ABLE -> (adjustable, mov(e)able)

g. (m>1) IBLE -> (defensible,bible)

h. (m>1) ANT -> (irritant,ant)

i. (m>1) EMENT -> (replacement)

j. (m>1) MENT -> (adjustment)

k. (m>1 and (*S or *T)) ION -> (adoption)

l. (m>1) OU -> (homologous)

m. (m>1) ISM -> (communism)

n. (m>1) ATE -> (activate)

o. (m>1) ITI -> (angulariti)

p. (m>1) OUS -> (homologous)

q. (m>1) IVE -> (effective)

r. (m>1) IZE -> (bowdlerize)

8. Menghapus akhiran e

   a. (m>1) E -> (probate, rate)

   b. (m=1 dan bukan *o) E -> (cease)

9. Mengurangi akhiran L rangkap

   (m>1 dan *LL) -> L (controll, roll)

2.4.2 Term Weigthing

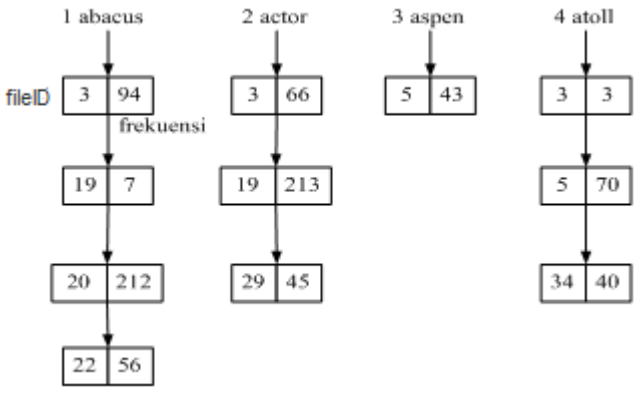
Term adalah suatu kata atau suatu kumpulan kata yang merupakan ekspresi verbal dari suatu pengertian. Dalam *information retrieval* sebuah *term* perlu diberi bobot, karena semakin sering suatu *term* muncul pada suatu *file*, maka kemungkinan *term* tersebut semakin penting dalam *file*.

Pembobotan *term* (*term weighting*) merupakan operasi yang dibutuhkan untuk membantu suatu proses *information retrieval* yaitu dengan menghitung kemunculan frekuensi suatu *term* pada sebuah *file*. Pembobotan *term* dibutuhkan dalam menentukan peringkat *file*. Misalkan *term* "*read*" dalam *file* A muncul sebanyak 5 kali. Maka bobot *term "read"* dalam *file* A adalah 5.

2.4.3 Inverted Index

Ide dasar dari *inverted index* adalah membuat *dictionary* dari *term-term* (Manning, 2008). Untuk setiap *term*, terdapat *list* yang merekam *file* dimana *term* tersebut berada yang disebut dengan *posting* dan *list* dari *posting* disebut *posting list*.

Didalam *file* koleksi, setiap *file* memiliki serial number yang unik, yang disebut dengan *file identifier* (fileID), dimana fileID dinyatakan dengan bilangan *integer* yang bertambah. *Input* dari proses penentuan indeks merupakan *list* dari *token* yang telah dinormalisasi dari setiap *file*, dimana *list* merupakan pasangan antar *term* dan fileID. Inti dari tahap proses penentuan indeks adalah melakukan pengurutan *term* secara alphabetis dan setiap *posting list* diurutkan berdasarkan docID. Kemunculan *term* yang sama dari satu *file* digabungkan dan kemunculan *term* yang sama dari *file* yang berbeda dikelompokkan (Manning, 2008). Contoh *inverted index* dapat dilihat pada Gambar 2.5.



Gambar 2.5 Contoh inverted index [Firdaus, 2008]

Gambar 2.5 menunjukkan *list* dari *inverted index*. Setiap *list* terdiri dari *term* dan *posting list*, dimana setiap *posting list* berisi fileID dan frekuensi kemunculan *term* dari *term* yang berada didalam fileID. Nilai frekuensi dari kemunculan term dalam file dituliskan dengan simbol $tf_{d,t}$. Terlihat di gambar ada 4 *term* yaitu : abacus, actor, aspen, dan atoll. Untuk *posting list* abacus file dengan ID = 3, memiliki frekuensi kemunculan *term* abacus sebesar 94 kali, atau dengan kata lain $tf_{3,abacus}$ = 94; $tf_{19,abacus}$ = 7; $tf_{20,abacus}$ = 212; $tf_{22,abacus}$ = 56. Dan begitu pun seterusnya untuk *term* yang lain.

2.5 Proses Pencarian

Setelah proses penentuan indeks, tahap selanjutnya adalah proses pencarian. Proses pencarian merupakan suatu proses mencari ke dalam basis data berdasarkan kata kunci yang dimasukkan oleh pengguna. Tahapan dalam proses pencarian adalah :

a. Tokenizing

Tahap awal dalam melakukan pencarian adalah memecah (*tokenizing*) *query*, *keyword* yang dimasukkan pengguna dijadikan sebagai *query* untuk melakukan pencarian kedalam *database*.

b. Filtering

Hasil dari proses *tokenizing* adalah *query token* (berisi *query token* yang sudah dipotong berdasarkan *token*), kemudian dilakukan *filtering* yaitu menghilangkan kata-kata yang umum.

c. Stemming

Setelah *query* tersebut melewati proses *filtering*, kemudian *query* tersebut melalui proses *stemming* yaitu suatu proses membuang awalan, akhiran dari kata. Hal ini dilakukan karena yang disimpan di dalam indeks *database* adalah kata dasar saja. Sebelum dilakukan proses *stemming*, terlebih dahulu dilakukan pengecekan kedalam kamus kata dasar.

d. Tagging

Setelah *query* tersebut melewati proses *stemming*, kemudian *query* tersebut melalui proses *tagging* adalah tahap mencari bentuk awal/root kata dari tiap kata dalam bentuk lampau atau kata hasil *stemming.*

e. Ranking

Tahap yang terakhir adalah melakukan perangkingan, dari *file-file* yang didapatkan. Perangkingan disusun berdasarkan bobot masing masing *file* terhadap kata kunci, *file* yang paling besar bobotnya menjadi *file* dengan *ranking* teratas.

2.5.1 Ranking

Ranking (perangkingan) merupakan pencarian *file-file* yang relevan terhadap *query* dan mengurutkan *file* tersebut berdasarkan kesesuaiannya dengan *query*.

Aplikasi pencarian menerima *query* dari pengguna, kemudian melakukan perangkingan terhadap *file* pada koleksi berdasarkan kesesuaiannya dengan *query*. Hasil perangkingan yang diberikan kepada pengguna merupakan *file* yang menurut sistem relevan dengan *query* dan metode untuk perangkingan tersebut

dilakukan dengan menghitung bobot masing-masing *file* terhadap *query*. Cara menghitung bobot masing-masing *file* terhadap query adalah dengan menggunakan metode tf-idf. Setelah dilakukan pembobotan, *file* hasil pencarian akan ditampilkan sesuai bobot dari *file*.

2.6 Metode TF-IDF

Metode tf-idf adalah suatu metode atau formula yang digunakan untuk menghitung bobot masing-masing *file* terhadap kata kunci (Harlian, Milkha. 2006). Metode TF/IDF dikenal sebagai algorithma yang sederhana namun relevan dalam melakukan pencocokan kata pada sebuah dokumen. Karena kelebihan itulah metode tf-idf banyak digunakan untuk pemberian bobot. *Term frequency* * *inverse document frequency* atau biasa disingkat dengan *tf-idf.*

Term frequency (*tf*) adalah frekuensi dari kemunculan sebuah *term* dalam *file* yang bersangkutan. Oleh sebab itu, *tf* memiliki nilai yang bervariasi dari suatu *file* ke *file* yang lain bergantung pada tingkat kepentingan sebuah *term* dalam sebuah *file* yang diberikan.

Inverse document frequency (*idf*) adalah suatu statistik yang mengkarakteristikkan sebuah *term* dalam keseluruhan koleksi *file*. *Idf* merupakan sebuah perhitungan dari bagaimana *term* didistribusikan secara luas pada koleksi *file* yang bersangkutan. Semakin sedikit *file* yang mengandung *term* yang dimaksud, maka nilai *idf* semakin besar. Jika setiap *file* dalam koleksi mengandung term yang bersangkutan, maka nilai dari idf dari term tersebut adalah nol. Hal ini menunjukkan bahwa sebuah *term* yang muncul pada setiap *file* dalam

koleksi tidak berguna untuk membedakan *file* berdasarkan topik tertentu. Nilai *idf* dalam sebuah *term t* dirumuskan dalam persamaan berikut:

$$idf_t \quad \log \frac{}{df_t} \tag{2.1}$$

$$tfidf_{,t} \quad tf_{,t} \cdot idf_t \tag{2.2}$$

Keterangan:

N	:	Jumlah *file*
df_t	:	Jumlah *file* yang mengandung *term* yang bersangkutan
$tf_{,t}$	:	Frekuensi kemunculan *term* pada *file* yang bersangkutan, dimana frekuensi ini sudah dihitung pada proses sebelumnya dan disimpan ke dalam basis data, lihat sub bab 2.4.3 Inverted Index
idf_t	:	Nilai *invers document frequency* (idf) dari sebuah *term.* Yaitu statistik yang mengkarakteristikkan sebuah *term* dalam keseluruhan koleksi *file*
$tfidf_{,t}$	:	Bobot sebuah term setelah dihitung menggunakan metode tf*idf

Perhitungan bobot dari *term* tertentu dalam sebuah *file* dengan menggunakan tf*idf menunjukkan bahwa deskripsi terbaik dari *file* adalah *term* yang banyak muncul dalam *file* tersebut dan sangat sedikit muncul pada *file* yang lain. Demikian juga sebuah *term* yang muncul dalam jumlah yang sedang dalam porsi yang cukup dalam *file* koleksi yang diberikan menjadi deskriptor yang baik. Bobot terendah akan di berikan pada yang yang muncul sangat jarang pada beberapa *file* dan term yang muncul pada hampir atau seluruh *file*.

Jika hasil dari proses pembobotan tf*idf menunjukkan bahwa ada *file-file* yang memiliki nilai bobot yang sama, maka diperlukan perhitungan lain untuk

menentukan peringkat dari *file-file* yang memiliki nilai sama. Perhitungan tersebut dilakukan dengan menggunakan metode *vector space model*. Dimana metode ini mengukur kemiripan antara suatu *file* dengan suatu *query*.

Contoh simulasi perhitungan nilai tf*idf bisa dilihat pada bagian dibawah ini :

query		
q	ant dog	
file	text	terms
f1	ant ant bee	ant bee
f2	dog bee dog hog dog ant dog	ant bee dog hog
f3	cat gnu dog eel fox	cat dog eel fox gnu

Hitung *term frekuensi (tf)*

	ant	bee	cat	dog	eel	fox	gnu	hog
f1	2	1						
f2	1	1		4				1
f3			1	1	1	1	1	

Hitung *document frequency (df)*

term	df
ant	2
bee	2
cat	1
dog	2
eel	1
fox	1
gnu	1
hog	1

Hitung *invers document frequency (idf)*

term	df	idf
ant	2	log(3/2) = 0,176
bee	2	log(3/2) = 0,176
cat	1	log(3/1) = 0,477
dog	2	log(3/2) = 0,176
eel	1	log(3/1) = 0,477
fox	1	log(3/1) = 0,477
gnu	1	log(3/1) = 0,477
hog	1	log(3/1) = 0,477

Hitung tf-idf

term	idf	tf_{f1}	tf_{f2}	tf_{f3}	tf-idf_{f1}	tf-idf_{f1}	tf-idf_{f1}
ant	log(3/2) = 0,176	2	1		0,334	0,176	
dog	log(3/2) = 0,176		4	1		0,704	0,176
tf-idf					0,334	0,88	0,176

Maka, nilai tf-idf berdasarkan query diatas adalah :

f1 : 0,334
f2 : 0,88
f3 : 0,176

2.7 Vector Space Model

Vector space model merupakan solusi atas permasalah yang dihadapi jika menggunakan metode tf-idf. Karena pada metode tf-idf terdapat kemungkinan antar *file* memiliki bobot yang sama, sehingga ambigu untuk diurutkan.

Vector space model merupakan model yang digunakan untuk mengukur kemiripan antara suatu *file* dengan suatu *query*. Pada model ini, *query* dan *file* dianggap sebagai vektor-vektor pada ruang *n*-dimensi, dimana n adalah jumlah dari seluruh term tunggal. (Harjono & David, 2005).

Inti perhitungan dari metode ini adalah dengan menghitung nilai cosinus sudut dari dua vektor, yaitu bobot dari tiap *file* dan bobot dari *query*. Menggunakan *consine similiarity* untuk mengukur kemiripan *file* terhadap *query*.

Berikut ini adalah tahapan untuk menghitung nilai *similarity* :

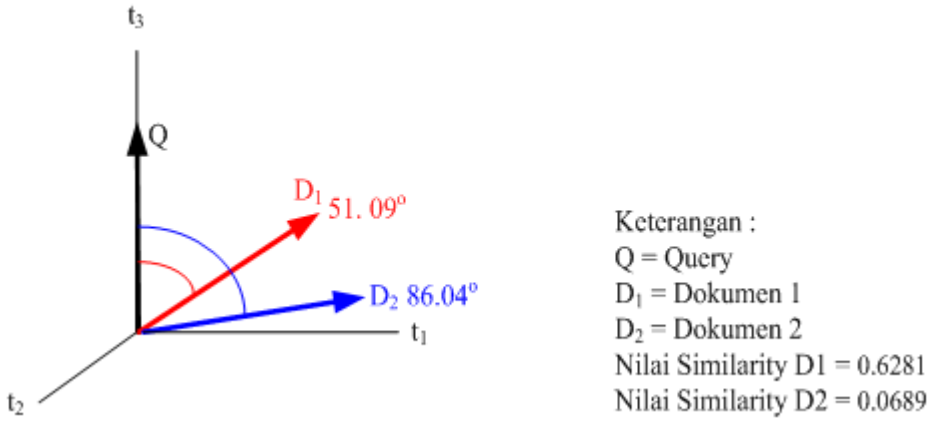
1	Hitung *Length*	:	$\|x\| = \sqrt{x_1^2 + x_2^2 + x_3^2 + \dots x_n^2}$
2	Hitung *Inner Product*	:	$x_1.x_2 = x_{11}.x_{21} + x_{12}.x_{22} + \dots x_{1n}.x_{2n}$
3	Hitung nilai *Cosine of the angl*e	:	$\text{Cos } \theta = \frac{x_1.x_2}{\|x_1\|.\|x_2\|}$

$x_1, x_2, x_3 \dots x_n$ = *term*

$x_1.x_2$ = *term* yang sesuai dengan *query*

Cos θ = sudut antara vektor *query* dan vektor dokumen.

Contoh sudut antara vektor *query* dan vektor dokumen dapat dilihat pada gambar 2.6.



Gambar 2.6 Representasi grafis sudut vector dokumen dan query

Pada Gambar 2.6 menunjukkan bahwa besar sudut antara Q dan D_1 lebih kecil dibandingkan antara Q dan D_2 sehingga dokumen 1 lebih relevan dibandingkan dokumen 2. Hal ini disebabkan semakin "dekat" atau bahkan "sama" suatu vektor dokumen dengan vektor *query* maka dokumen dapat dipandang semakin relevan dengan *query*. Pada gambar 2.9 didapatkan hasil dari nilai *similarity* D_1 = 0.6281 dan D_2 = 0,0689. Untuk D_1 = 0,6281 besar sudutnya adalah 51,09 sedangkan untuk D_2 = 0,0689 besar sudutnya 86,04. Sehingga semakin tinggi nilai *similarity* maka besar sudutnya akan semakin kecil dan akan semakin relevan.

Contoh perhitungan nilai cosinus similarity bisa dilihat pada bagian dibawah ini :

query		
q	ant dog	
document	text	terms
f1	ant ant bee	ant bee
f2	dog bee dog hog dog ant dog	ant bee dog hog
f3	cat gnu dog eel fox	cat dog eel fox gnu

Hitung Length

	ant	bee	cat	dog	eel	fox	gnu	hog	length
q	1			1					$\sqrt{2}$
f1	2	1							$\sqrt{5}$
f2	1	1		4				1	$\sqrt{19}$
f3			1	1	1	1	1		$\sqrt{5}$

Hitung Inner Product

f1 1*2 + 1*0 = 2
f2 1*1 + 1*4 = 5
f3 1*0 + 1*1 = 1

Hitung nilai Cosine of the angle

	f1	f2	f3
q	2 / √10 = 0,63	5 / √38 = 0,81	5 / √10 = 0,32

2.8 Konsep Berorientasi Objek

Desain berorientasi objek adalah sebuah teknik OOP yang memusatkan desain pada objek dan kelas berdasarkan pada skenario dunia nyata. Hal ini menegaskan keadaan, perilaku dan interaksi dari objek. Selain itu juga menyediakan manfaat akan kebebasan pengembangan, meningkatkan kualitas,

mempermudah pemeliharaan, mempertinggi kemampuan dalam modifikasi dan meningkatkan penggunaan kembali software. (JEDI, 2007)

Dalam konsep berorientasi objek dikenal beberapa istilah sebagai berikut:

2.8.1 Kelas

Kelas adalah cetak biru (rancangan) dari objek. Ini berarti kita bisa membuat banyak objek dari satu macam kelas. Kelas mendefinisikan sebuah tipe dari objek. Di dalam kelas kita dapat mendeklarasikan variabel dan menciptakan objek (instansiasi). Sebuah kelas mempunyai anggota (*member*) yang terdiri atas atribut dan *method*. (Common Labz, 2008)

2.8.2 Objek

Objek secara lugas dapat diartikan sebagai instansiasi atau hasil ciptaan dari suatu kelas yang telah dibuat sebelumnya. Dalam pengembangan program orientasi objek lebih lanjut, sebuah objek dapat dimungkinkan terdiri atas objek-objek lain. Seperti halnya objek mobil terdiri atas mesin, ban, kerangka mobil, pintu, karoseri dan lain-lain. Atau, bisa jadi sebuah objek merupakan turunan dari objek lain sehingga mewarisi sifat-sifat induknya. (Common Labz, 2008)

2.8.3 Atribut

Atribut menunjuk pada elemen data dari sebuah objek. Atribut menyimpan informasi tentang objek.

2.8.4 Method

Method dikenal juga sebagai suatu fungsi dan prosedur. Dalam OOP, *method* digunakan untuk memodularisasi program melalui pemisahan tugas dalam suatu class. Pemanggilan *method* menspesifikasikan nama *method* dan menyediakan informasi (parameter) yang diperlukan untuk melaksanakan tugasnya. (Common Labz, 2008)

2.8.5 Constructor

Constructor adalah tipe khusus *method* yang digunakan untuk menginstansiasi atau menciptakan sebuah objek. Nama *constructor* adalah sama dengan nama kelasnya. Selain itu, constructor tidak bisa mengembalikan suatu nilai (*not return value*) bahkan *void* sekalipun. (Common Labz, 2008)

2.8.6 Package

Package menunjuk pada pengelompokkan kelas dan/atau *sub package*. Strukturnya dapat disamakan dengan direktorinya. (JEDI, 2007)

2.9 Unified Modelling Languange

Unified Modelling Language (UML) adalah sebuah "bahasa" yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. (Common Labz, 2008)

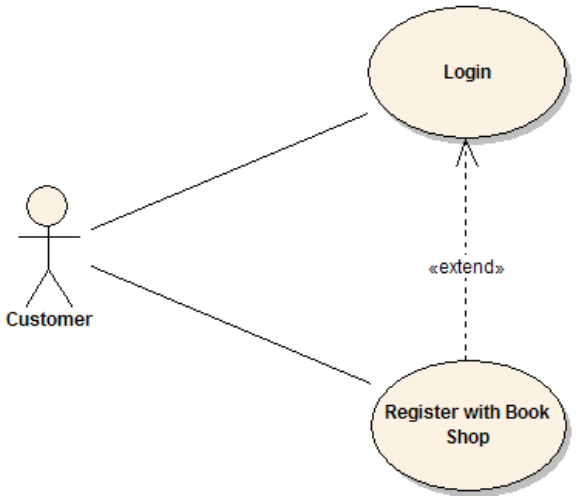
Dengan menggunakan UML kita dapat membuat model untuk semua jenis aplikasi piranti lunak, dimana aplikasi tersebut dapat berjalan pada piranti keras,

sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan *class* dan *operation* dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa-bahasa berorientasi objek seperti C++, Java, C# atau VB.NET. (Darwiyanti & Wahono, 2003)

Pada UML dikenal beberapa diagram, diantaranya sebagai berikut:

2.9.1 Use Case Diagram

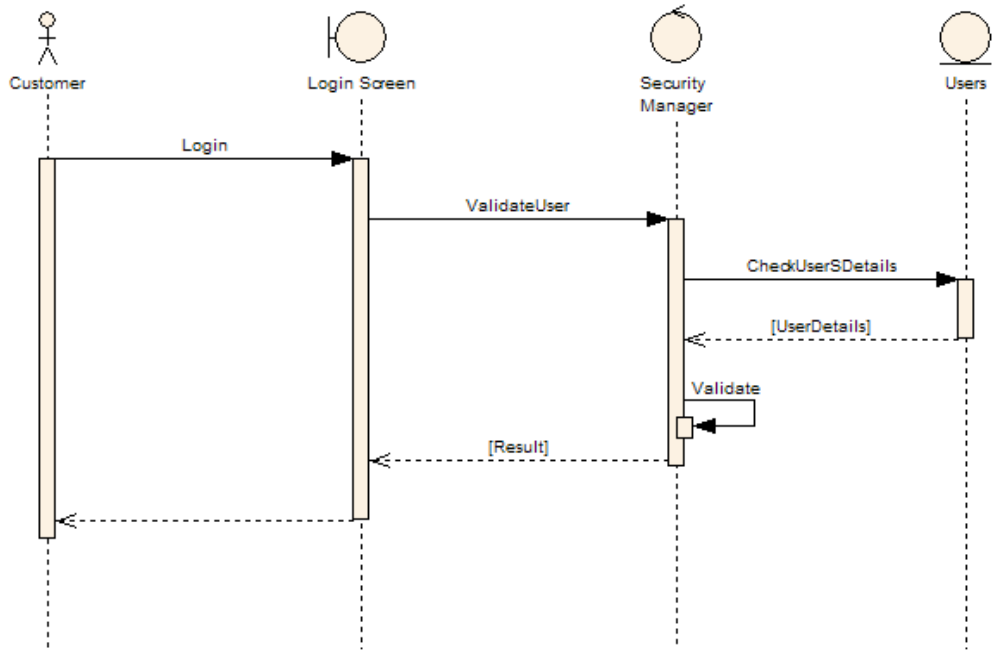
Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. (Darwiyanti & Wahono, 2003)



Gambar 2.7 Contoh *use case diagram*

2.9.2 Sequence Diagram

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, display, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atas dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah event untuk menghasilkan output tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan. (Darwiyanti &Wahono, 2003)

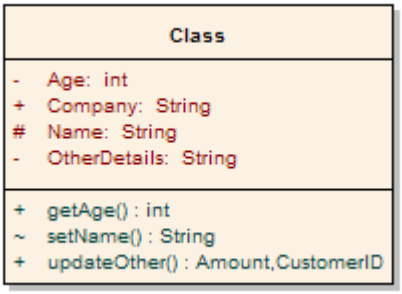


Gambar 2.8 Contoh *sequence diagram*

2.9.3 Class Diagram

Class (Kelas) adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. Kelas menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi).

Diagram kelas menggambarkan struktur dan deskripsi kelas, paket dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi, dan lain-lain. (Darwiyanti & Wahono, 2003)



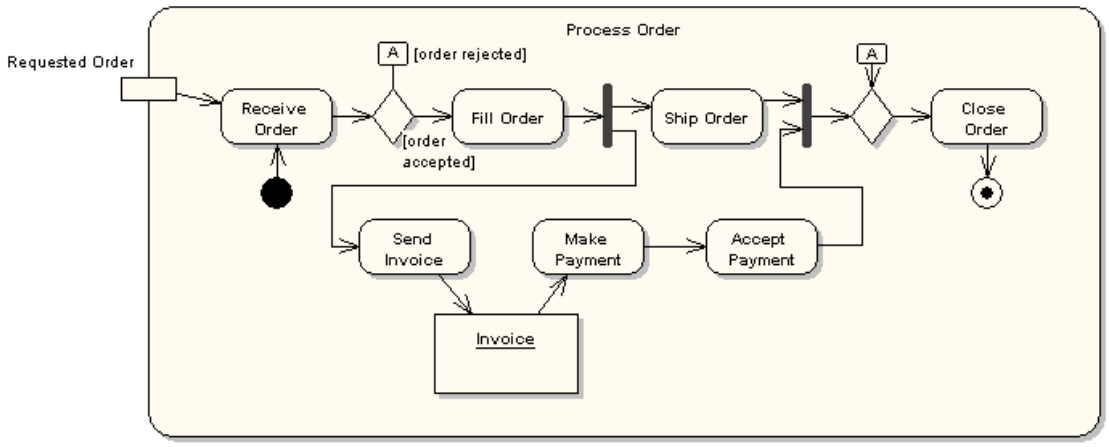
Gambar 2.9 Contoh diagram kelas

2.9.4 Activity Diagram

Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram*

juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan perlakuan internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum. (Darwiyanti &Wahono, 2003)



Gambar 2.10 Contoh *activity diagram*

2.10 JAVA

Java dibuat dan diperkenalkan pertama kali oleh sebuah tim *Sun Microsystem* yang dipimpin oleh **Patrick Naughton** dan **James Gosling** pada tahun 1991 dengan *code* nama **Oak.** Tahun 1995 Sun mengubah nama Oak tersebut menjadi **Java**. Ide pertama kali kenapa java dibuat adalah karena adanya

motivasi untuk membuat sebuah bahasa pemrograman yang bersifat *portable* dan *platform independent* (tidak tergantung mesin dan sistem operasi) yang dapat digunakan untuk membuat piranti lunak yang dapat ditanamkan (*embedded*) pada berbagai macam peralatan elektronik consumer biasa, seperti microwave, remote control, telepon, card reader dan sebagainya. (Common Labz, 2008)

Java adalah bahasa pemogramman yang sederhana dan tangguh. Berikut ini adalah beberapa karakteristik dari Java sesuai dengan definisi Sun.

1. **Sederhana.** Bahasa pemrograman Java menggunakan sintaks mirip dengan C++ namun sintaks pada Java telah banyak diperbaiki terutama menghilangkan penggunaan pointer yang rumit dan multiple inheritance. Java juga menggunakan *automatic memory allocation* dan *memory garbage collection.*
2. **Berorientasi objek.** Java mengunakan pemrograman berorientasi objek yang membuat program dapat dibuat secara modular dan dapat dipergunakan kembali. Pemrograman berorientasi objek memodelkan dunia nyata kedalam objek dan melakukan interaksi antar objek-objek tersebut.
3. **Dapat didistribusi dengan mudah.** Java dibuat untuk membuat aplikasi terdistribusi secara mudah dengan adanya *libraries networking* yang terintegrasi pada Java.
4. ***Interpreter***. Program Java dijalankan menggunakan interpreter yaitu Java Virtual Machine (JVM). Hal ini menyebabkan *source code* Java yang telah

dikompilasi menjadi Java *bytecodes* dapat dijalankan pada platform yang berbeda-beda.

5. ***Robust.*** Java mempuyai reliabilitas yang tinggi. *Compiler* pada Java mempunyai kemampuan mendeteksi error secara lebih teliti dibandingkan bahasa pemrograman lain. Java mempunyai *runtime-Exception handling* untuk membantu mengatasi error pada pemrograman.
6. **Aman.** Sebagai bahasa pemrograman untuk aplikasi internet dan terdistribusi, Java memiliki beberapa mekanisme keamanan untuk menjaga aplikasi tidak digunakan untuk merusak sistem komputer yang menjalankan aplikasi tersebut.
7. ***Architecture Neutral*.** Program Java merupakan *platform independent*. Program cukup mempunyai satu buah versi yang dapat dijalankan pada platform yang berbeda dengan Java Virtual Machine.
8. ***Portabel*.** *Source code* maupun program Java dapat dengan mudah dibawa ke platform yang berbeda-beda tanpa harus dikompilasi ulang.
9. ***Performance*.** Performance pada Java sering dikatakan kurang tinggi. Namun performance Java dapat ditingkatkan menggunakan kompilasi Java lain seperti buatan Inprise, Microsoft ataupun Symantec yang menggunakan Just In Time Compilers (JIT).
10. ***Multithreaded*.** Java mempunyai kemampuan untuk membuat suatu program yang dapat melakukan beberapa pekerjaan secara sekaligus dan simultan.

11. **Dinamis.** Java didesain untuk dapat dijalankan pada lingkungan yang dinamis. Perubahan pada suatu class dengan menambahkan properties ataupun method dapat dilakukan tanpa menggangu program yang menggunakan class tersebut.

2.11 MySQL Server 5

MySQL adalah salah satu Relational Database Management System (RDBMS) yang didistribusikan secara gratis di bawah lisensi GPL (General Public License) yang dibuat oleh MySQLAB. MySQL merupakan turunan dari konsep database SQL (*Structured Query Languange*) untuk pemilihan/seleksi dan pemasukan data yang memungkinkan pengoperasian data dikerjakan dengan mudah secara otomatis.

Pada MySQL terdapat beberapa *storage engine* (mesin penyimpanan data) yang berbeda-beda dan dapat ditentukan oleh proggramer database tersebut. Gambar di bawah ini merupakan arsitektur MySQL:

Kelebihan MySQL:

1. ***Portability*.** Berjalan stabil pada berbagai sistem operasi (Windows,Linux, Mac OS, Solaris dsb)
2. ***Free dan Open Source*.** Didistribusikan secara gratis dan *open source*.
3. ***Multiuser*.** dapat digunakan oleh beberapa pengguna dalam waktu yang bersamaan tanpa mengalami masalah atau konflik
4. ***Performance Tuning*.** Memiliki kecepatan yang baik dalam menangani *query* sederhana.

5. ***Column Types*.** memiliki tipe kolom yang kompleks, seperti : signed/unsigned integer, float, double, char, varchar, blob, time, datetime, timestamp, year, set serta enum
6. ***Command dan Functions*.** memiliki operator dan fungsi secara penuh yang mendukung perintah SELECT dan WHERE dalam *query*
7. ***Security*.** memiliki lapisan sekuritas, seperti level subnetmask, nama *host* dan izin akses pengguna disertai dengan password enkripsi.
8. ***Scalability dan Limits*.** Mampu menangani basis data dalam skala besar dengan jumlah records lebih dari 50 juta dan 60 ribu tabel serta 5 miliar baris.
9. ***Connectivity*.** Dapat melakukan koneksi dengan *client* menggunakan protokol TCP/IP, Unix soket atau Named Pipes.
10. ***Localization*.** Dapat mendeteksi pesan kesalah (*error code*) pada *client* dengan menggunakan lebih dari dua puluh bahasa.
11. ***Interface*.** Memiliki interface terhadap berbagai aplikasi dan bahasa pemrograman dengan fungsi API (Application Programming Interface).
12. ***Clients* dan *Tools*.** dilengkapi dengan berbagai tool yang dapat digunakan untuk administrasi database
13. **Struktur Tabel.** Memiliki struktur tabel yang lebih fleksibel dalam menangani ALTER TABLE dibandingkan Oracle atau PostgreSQL. (MySQL, 2000)