BAB 2

TINJAUAN PUSTAKA

2.1 Tinjauan Tempat Penelitian

PT eBdesk Indonesia didirikan pada tahun 1998, namun secara legal perusahaan ini baru diresmikan pada tahun 2001, sebagai perusahaan produsen perangkat lunak yang bergerak di bidang teknologi informasi. Saat awal berdiri, PT eBdesk Indonesia fokus dalam mengembangkan produk portal perusahaan berdasarkan teknologi internet dan sampai saat ini, PT eBdesk Indonesia tetap fokus untuk mengembangkan produk-produk yang berkelanjutan sesuai dengan perubahan teknologi dan tren. PT eBdesk Indonesia adalah salah satu perusahaan pertama yang mengembangkan *corporate portal*, dimana *corporate portal* itu sendiri adalah sebuah konsep baru ketika itu.

Produk pertama PT eBdesk Indonesia yang diperkenalkan adalah *eBdesk Corporate Portal*. Aplikasi ini mengintegrasikan berbagai informasi dan aplikasi dari dalam atau luar perusahaan. Ini adalah *platform* untuk menyebarkan sepenuhnya *portal web interface* yang memberikan kolaborasi antara karyawan, manajer, pemilik, mitra dan pelanggan. Implementasi yang sukses membuat PT eBdesk Indonesia mengembangkan lebih banyak segmentasi produk dalam manajemen informasi untuk memberikan nilai lebih tehadap kepuasan pelanggan.

Pada mulanya produk dan *framework* PT eBdesk Indonesia dikembangkan dengan C / C++ karena merupakan bahasa pemrograman yang paling komprehensif di industri IT pada masanya. Melalui pengalamannya dalam menggunakan bahasa pemrograman C / C++, PT eBdesk Indonesia terus mengadopsi struktur *development framework* yang lebih baik untuk memberikan produk yang lebih kuat. Hal ini menjadi titik awal yang baik ketika era Java *framework* datang, sehingga saat ini produk-produk PT eBdesk Indonesia berjalan diatas Java *framework.* [23]

PT eBdesk Indonesia 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 yaitu XYBASE (Malaysia), PT eBdesk Indonesia (Indonesia), Patimas (Malaysia), TMS, SSC Solution (Thailand) dan BlueBridge *Software*.

Saat ini PT eBdesk Indonesia sedang mengembangkan sebuah produk bernama *eBdesk Media Management*. *eBdesk Media Management* adalah suatu aplikasi berbasis *web* yang dapat memudahkan *user* untuk mangambil isi berita dari beberapa situs internet. Aplikasi ini juga dapat memonitoring semua perubahan isi berita dalam jangka waktu tertentu.

eBdesk Media Management dapat menjadi alat bantu untuk memantau berita, *blog*, forum, *video-sharing* dan situs media sosial seperti Twitter dan Facebook yang membahas sebuah topik. Secara otomatis *Media Management* juga dapat menemukan tokoh, organisasi, media, lokasi dan objek lainnya yang terlibat di dalam repositori.

eBdesk Media Management dapat menjadi media analisis, dimana dapat memberikan *timeline*, tokoh utama, media yang membahas suatu isu, lokasi, sentimen dari setiap tanggapan dan perbandingan tanggapan tokoh, bahkan mampu menganalisa lebih dalam dengan *text mining*, ontologi, hubungan antar objek, dan metode teks analisa lainnya.

2.2 *E-commerce*

Electronic Commerce (*e-commerce*) merupakan konsep baru yang biasa digambarkan sebagai proses jual beli barang atau jasa pada *World Wide Web Internet* [15] atau proses jual beli atau pertukaran produk, jasa dan informasi melalui jaringan informasi termasuk internet [18]. *E-commerce* merupakan kegiatan bisnis yang dijalankan secara elektronik melalui suatu jaringan internet atau kegiatan jual beli barang atau jasa melalui jalur komunikasi digital.

2.2.1 Tipe *e-commerce*

Jika dilihat dari jenis aplikasi yang digunakan, *e-commerce* dapat dikategorikan menjadi empat tipe, sebagai berikut [5]:

1. *I-Market*

   Suatu tempat atau arena di dunia maya dimana calon pembeli dan penjual saling bertemu untuk melakukan transaksi secara elektronis melalui medium Internet, dari definisi tersebut terlihat bahwa tipe bisnis yang terjadi adalah *Business to Consumer*.

2. *Customer Care*

   Suatu usaha dari perusahaan untuk menjalin hubungan interaktif dengan pelanggan yang telah dimilikinya sebagian besar aplikasi yang dipergunakan bersifat *Business to Consumer*.

3. *Vendors Management*

   Perusahaan melakukan proses pemesanan, pengadaan, dan pembeliaan bahan-bahan yang dibutuhkan dari berbagai pemasok dan vendor melalui Internet dan para rekanan ini akan mengirimkannya kepada perusahaan sesuai dengan kebutuhan.Tipe *Business to Business* merupakan *platform* transaksi yang diterapkan dalam *tipe e-commerce* ini.

4. *Extended Supply Chain*

   Melakukan optimisasi *supply chain* perusahaan dengan cara menjalin hubungan dengan seluruh rekanan atau pihak-pihak lain yang terlibat langsung dalam proses penciptaan produk atau jasa melalui jalur elektronis semacam Internet. Jelas terlihat bahwa seperti halnya tipe *e-commerce Vendor Management*, prinsip *Business to Business* merupakan *platform* yang diterapkan dalam pengembangan *e-commerce* terkait.

2.2.2 Mekanisme Transaksi *e-commerce*

Pembeli yang akan berbelanja di toko *online* dapat menggunakan fasilitas *shopping cart*. *Shopping cart* adalah sebuah *software* di situs *web* yang mengizinkan pelanggan untuk melihat toko yang anda buka kemudian memilih

item barang untuk diletakkan dalam kereta dorong yang kemudian membelinya saat melakukan *check out*.

Konsep *shopping cart* ini meniru kereta belanja yang biasanya digunakan orang untuk berbelanja di pasar swalayan. *Shopping cart* biasanya berupa formulir dalam *web* dan dibuat dengan kombinasi CGI (*Common Gateway Interface*), *database*, dan HTML (*Hyper Text Markup Language*), dimana barang-barang yang dimasukkan ke *shopping cart* masih dapat dibatalkan, jika pembeli berniat untuk membatalkan membeli barang tersebut. Jika pembeli ingin membayar untuk barang yang dipilih, maka pembeli harus mengisi formulir transaksi, biasanya formulir ini menanyakan identitas pembeli. Setelah pembeli mengadakan transaksi, barang akan dikirimkan melalui jasa pos atau jasa pengiriman lain langsung ke alamat pembeli.

2.2.3 Metode Pembayaran di *e-commerce*

Terdapat 3 metode pembayaran yang biasa digunakan dalam transaksi menggunakan *e-commerce*:

1. *Online Procesing Credit Cart*

   Metode ini cocok digunakan untuk produk yang bersifat retail dimana pasarnya adalah seluruh dunia. Pembayaran dilakukan secara *real time* (proses verifikasi saat itu juga).

2. *Money Transfer*

   Cara ini lebih aman untuk menerima pembayaran dari konsumen mancanegara, namun memerlukan biaya tambahan bagi konsumen dalam bentuk *fee* bagi pihak penyedia jasa *money transfer* untuk mengirim sejumlah uang ke Negara lain.

3. *Cash on Delivery*

   Pembayaran dengan bayar di tempat ini hanya bisa dilakukan jika konsumen berada dalam satu kota yang sama dengan penyedia jasa.

2.3 *Market Basket Analysis*

Market basket analysis merupakan salah satu tipe analisis data yang paling sering digunakan dalam dunia pemasaran. Tujuan dari *market basket analysis* adalah untuk menentukan produk-produk apa saja yang paling sering dibeli atau digunakan sekaligus oleh para pelanggan. *Market basket analysis* adalah salah satu cara yang digunakan untuk menganalisis data penjualan dari suatu perusahaan. Proses ini menganalisis *buying habits* konsumen dengan menemukan asosiasi antar *item-item* yang berbeda yang diletakkan konsumen dalam *shopping basket*. Hasil yang telah didapatkan ini nantinya dapat dimanfaatkan oleh perusahaan untuk mengembangkan strategi pemasaran dengan melihat *item-item* mana saja yang sering dibeli secara bersamaan oleh konsumen. [3]

Informasi mengenai produk-produk yang biasanya dibeli secara bersama-sama oleh para pelanggan dapat memberikan "wawasan" tersendiri bagi para pengelola untuk membuat lebih banyak produk demi meningkatkan kepuasan pelanggan. Misalnya, marketing bisa saja memanfaatkan informasi seperti ini untuk menggabungkan modul-modul yang umumnya dipesan ke dalam sebuah paket produk. Hal ini bertujuan untuk memudahkan "mayoritas" pembeli agar semakin mudah membeli produk dengan modul-modul yang berkaitan sekaligus.

2.4 *Data Mining*

Data mining adalah proses mencari pola atau informasi menarik dalam data terpilih dengan menggunakan teknik atau metoda tertentu. Teknik, metoda atau algoritma dalam *data mining* sangat bervariasi. Pemilihan metoda atau algoritma yang tepat sangat tergantung pada tujuan dan proses secara keseluruhan. Karakteristik dari *data mining* adalah jumlah data yang besar dan harus dianalisa dengan teknik yang otomatis, *noisy, incomplete data,* data yang bersifat *heterogenous* [19].

Teknik analisa *data mining* pada umumnya diorientasikan untuk dapat menerjemahkan data dalam jumlah yang besar, dengan tujuan dapat menghasilkan keputusan dan kesimpulan yang terjamin keakuratannya [13].

Berikut adalah tahapan-tahapan untuk mendapatkan *knowledge* dari proses *data mining:*

1. *Selection*

   Selection adalah proses memilih dan menyaring data berdasarkan beberapa kriteria, misalkan berdasarkan jenis barang.

2. *Preprocessing*

   Preprocessing adalah mempersiapkan data dengan cara membersihkan data, informasi atau *field-field* yang tidak dibutuhkan, yang nantinya akan memperlambat kinerja proses *query.*

3. *Transformation*

   Pada proses *transformation*, data-data yang telah melalui proses *select* dan *pre-processing* tidak serta merta bisa langsung digunakan, tetapi data ditransformasikan terlebih dahulu ke bentuk yang lebih *navigable* dan *useable*.

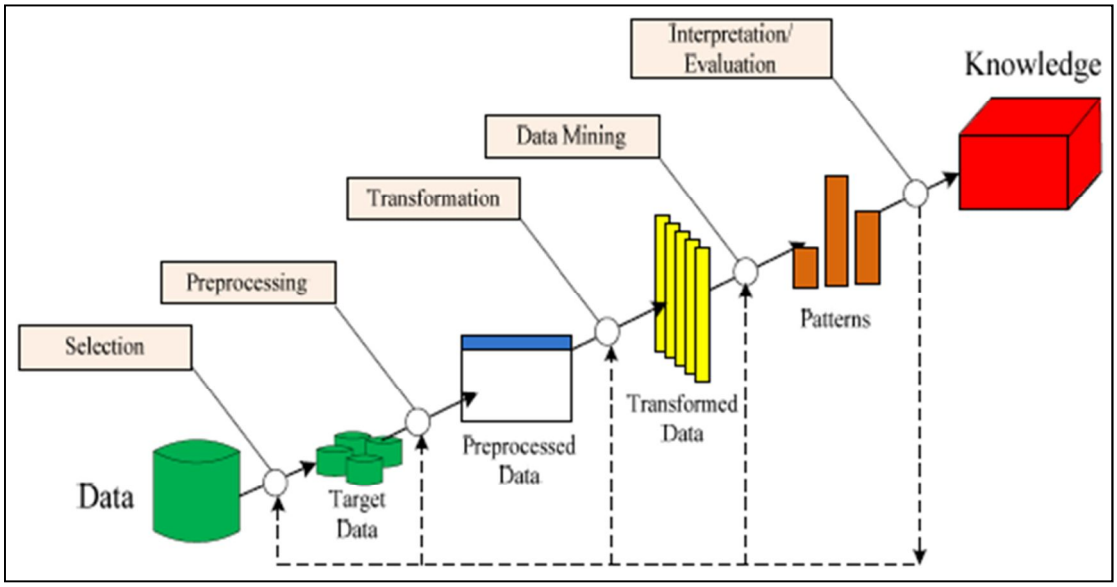
4. *Data mining*

   Pada bagian ini, dipusatkan untuk mendapatkan pola-pola dari data (*extiction of data*). Setidaknya ada 3 teknik yang paling sering digunakan pada data mining, yaitu: *classification, clustering* dan *association rule mining.*

5. *Interpretation and evaluation*

   Dalam proses ini *pattern* atau pola-pola yang telah diidentifikasi oleh sistem kemudian diterjemahkan atau diinterpretasikan ke dalam bentuk *knowledge* yang lebih mudah dimengerti oleh user untuk mengambil keputusan.

Jika digambarkan, maka tahapan-tahapan untuk mendapatkan *knowledge* dari proses data mining adalah sebagai berikut ini:



Gambar 2.1 Tahapan Untuk Mendapatkan *Knowledge* Dari Proses Data Mining

2.5 *Association Rule Mining*

Association rule mining adalah teknik *mining* untuk menentukan aturan assosiatif antara suatu kombinasi *item*. Contoh dari aturan assosiatif dari analisa pembelian di suatu pasar swalayan adalah bisa diketahui berapa besar kemungkinan seorang pelanggan membeli roti bersamaan dengan susu. Dengan pengetahuan tersebut, pemilik pasar swalayan dapat mengatur penempatan barangnya atau merancang kampanye pemasaran dengan memakai kupon diskon untuk kombinasi barang tertentu. Penting tidaknya suatu aturan assosiatif dapat diketahui dengan menggunakan dua parameter, yaitu *support* dan *confidenc*. [12] Menurut Gregorius S. Budhi dan kawan-kawan "*Association rule mining* adalah suatu prosedur untuk mencari hubungan antar *item* dalam suatu *dataset* yang ditentukan." [3]

Association rule meliputi dua tahap [3], yaitu:

1. Mencari kombinasi yang paling sering terjadi dari suatu *itemset.*
2. Mendefinisikan *condition* dan *result* (untuk *conditional association rule*).

Umumnya, ada dua ukuran kepercayaan (*interestingness measure*) yang digunakan dalam menentukan suatu *association rule* yaitu:

1. *Support:* suatu ukuran yang menunjukan seberapa besar tingkat dominasi suatu *item* atau *itemset* dari keseluruhan transaksi. Ukuran ini menentukan apakah suatu *item* atau *itemset* layak untuk dicari *confidence factor*-nya.
2. *Confidence:* suatu ukuran yang menunjukan hubungan antar dua atau lebih *item* secara *conditional* (misal, seberapa sering *item* B dibeli jika membeli *item* A).

Kedua ukuran ini nantinya berguna dalam menentukan *interesting association rules* atau *strong association rule*, yaitu untuk dibandingkan dengan batasan yang ditentukan oleh *user*. Batasan tersebut umumnya terdiri dari minimum *support* dan minimum *confidence*. Bila memenuhi kedua batasan maka sebuah *rule* dapat disebut *interesting rule* atau *strong rule.*

2.6 FP-Tree

FP-tree adalah struktur penyimpanan data yang dipadatkan. FP-tree dibangun dengan memetakan setiap data transaksi ke dalam setiap lintasan tertentu dalam FP-tree. Setiap transaksi yang dipetakan mungkin ada transaksi yang memiliki *item* yang sama, maka lintasannya memungkinkan untuk saling menimpa. Semakin banyak data transaksi yang memiliki *item* yang sama, maka proses pemadatan dengan struktur data FP-tree semakin efektif. Kelebihan dari FP-tree adalah hanya memerlukan dua kali *scanning* data transaksi yang terbukti sangat efisien.

Misal I = {A1, A2, ..., An} adalah kumpulan dari *item* dan *data base* transaksi DB = {T1, T2, ... Tn}, dimana Ti (i € [1..n]) adalah sekumpulan transaksi yang mengandung *item* dari I. Sedangkan *support* adalah penghitung (*counter*) frekuensi kemunculan transaksi yang mengandung suatu pola. Suatu pola dikatakan sering muncul (*frequent pattern*) apabila *support* dari pola tersebut tidak kurang dari suatu konstanta ξ (batas ambang minimum dari *support*) yang telah didefinisikan sebelumnya. Permasalahan mencari pola *frequent* dengan batas

ambang minimum *support count* ξ inilah yang dicoba untuk dipecahkan oleh FP-tree dengan menggunakan algoritma FP-growth.

Adapun FP-tree adalah sebuah pohon dengan definisi sebagai berikut:

a. FP-tree dibentuk oleh sebuah akar yang diberi label *null*, sekumpulan cabang yang terdiri dari *item-item* tertentu, dan sebuah tabel *frequent header*.
b. Setiap simpul dalam FP-tree mengandung 3 informasi penting, yaitu label *item* (menginformasikan jenis *item* yang direpresentasikan simpul tersebut), *support count* (merepresentasikan jumlah lintasan transaksi yang melalui simpul tersebut) dan *pointer* penghubung yang menghubungkan simpul-simpul dengan label *item* sama antar lintasan, ditandai dengan garis panah putus-putus.

Kasus contoh 1, misalkan diberikan tabel data transaksi sebagai berikut, dengan minimum *support count* ξ = 20% dan identitas transaksi dinamakan dengan variabel TID.

Tabel 2.1 Tabel Data Transaksi

TID	**Transaksi**
1	a,b
2	b,c,d,g,h
3	a,c,d,e,f
4	a,d,e
5	a,b,z,c
6	a,b,c,d
7	a,r
8	a,b,c
9	a,b,d
10	b,c,e

Frekuensi kemunculan tiap *item* dapat dilihat pada tabel 2.2 berikut:

Tabel 2.2 Frekuensi Kemunculan Tiap *Item*

Item	**Frekuensi**
a	8
b	7
c	6
d	5
e	3
f	1
g	1
h	1
r	1
z	1

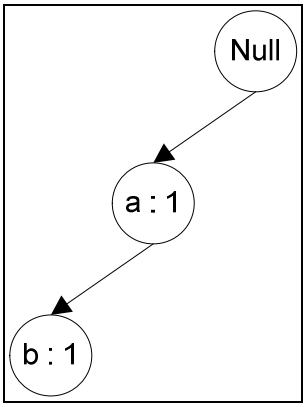
Setelah dilakukan pemindaian pertama didapat *item* yang memiliki frekuensi di atas *support count* ξ = 20% adalah a,b,c,d, dan e. Kelima *item* inilah yang akan berpengaruh dan akan dimasukkan ke dalam FP-tree, selebihnya (r, z, g dan h) dapat dibuang karena tidak berpengaruh signifikan.

Tabel 2.3 berikut mendata kemunculan *item* yang *frequent* dalam setiap transaksi, diurutkan berdasarkan yang frekuensinya paling tinggi.

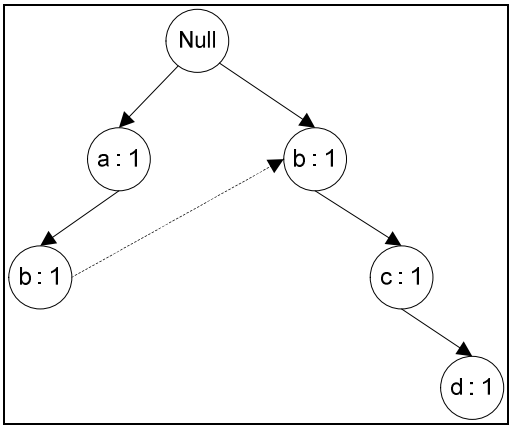
Tabel 2.3 Tabel Data Transaksi Setelah Pemindaian

TID	***Item***
1	{a,b}
2	{b,c,d}
3	{a,c,d,e}
4	{a,d,e}
5	{a,b,c}
6	{a,b,c,d}
7	{a}
8	{a,b,c}
9	{a,b,d}
10	{b,c,e}

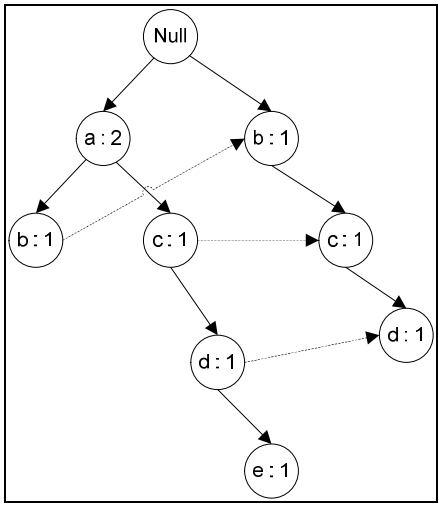
Berikut ilustrasi mengenai pembentukan FP-tree dalam bentuk gambar:



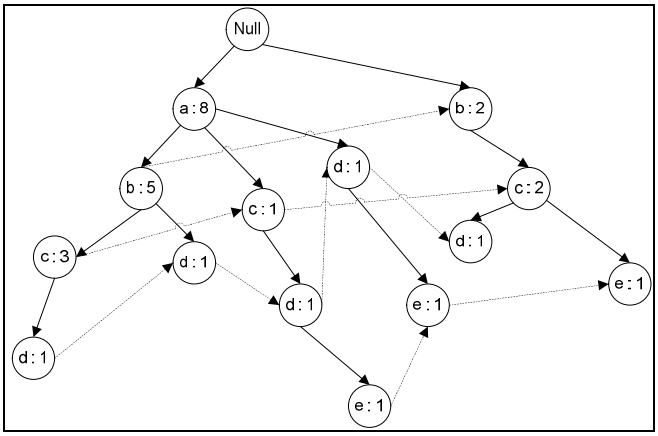
Gambar 2.2 Hasil Pembentukan FP-Tree Setelah Pembacaan TID 1



Gambar 2.3 Hasil Pembentukan FP-Tree Setelah Pembacaan TID 2



Gambar 2.4 Hasil Pembentukan FP-Tree Setelah Pembacaan TID 3



Gambar 2.5 Hasil Pembentukan FP-Tree Setelah Pembacaan TID 10

Diberikan 10 data transaksi dengan 5 jenis *item* seperti pada tabel 2.3 di atas. Gambar 2.2 sampai 2.5 menunjukkan proses terbentuknya FP-tree setiap TID dibaca. Setiap simpul pada FP-tree mengandung nama sebuah *item* dan *counter support* yang berfungsi untuk menghitung frekuensi kemunculan *item* tersebut dalam tiap lintasan transaksi.

FP-tree yang merepresentasikan data transaksi pada tabel 2.1 dibentuk dengan cara sebagai berikut:

1. Kumpulan data dipindai pertama kali untuk menentukan *support count* dari setiap *item*. *Item* yang tidak *frequent* dibuang, sedangkan *frequent*

item dimasukkan dan disusun dengan urutan menurun, seperti yang terlihat pada tabel 2.2.

2. Pemindaian kedua, yaitu pembacaan TID pertama {a, b} akan membuat simpul a dan b, sehingga terbentuk lintasan transaksi Null→a→b. *Support count* dari setiap simpul bernilai awal 1.
3. Setelah pembacaan transaksi kedua {b, c, d}, terbentuk lintasan kedua yaitu Null→b→c→d. Support *count* masing-masing *count* juga bernilai awal 1. Walaupun b ada pada transaksi pertama, namun karena *prefix* transaksinya tidak sama, maka transaksi kedua ini tidak bisa d*item*patkan dalam satu lintasan.
4. Transaksi ketiga memiliki *prefix* transaksi yang sama dengan transaksi pertama, yaitu a, maka lintasan transaksi ketiga dapat ditimpakan di a, sambil menambah *support count* dari a dan selanjutnya membuat lintasan baru sesuai dengan transaksi ketiga (lihat gambar 2.5).
5. Proses ini dilanjutkan sampai FP-tree berhasil dibangun berdasarkan tabel data transaksi yang diberikan.

2.7 FP-Growth

Setelah tahap pembangunan FP-tree dari sekumpulan data transaksi, akan diterapkan algoritma FP-growth untuk mencari *frequent itemset* yang memenuhi syarat. Algoritma FP-growth dibagi menjadi tiga langkah utama, yaitu :

1. Tahap Pembangkitan *Conditional Pattern Base*

   Conditional Pattern Base merupakan sub-*database* yang berisi *prefix path* (lintasan *prefix*) dan *suffix pattern* (pola akhiran). Pembangkit *conditional pattern base* didapatkan melalui FP-tree yang telah dibangun sebelumnya.
2. Tahap Pembangkitan *Conditional* FP-tree

   Pada tahap ini, *support count* dari setiap *item* pada setiap *conditional pattern base* dikumpulkan, lalu setiap *item* yang memiliki jumlah *support count* lebih besar sama dengan minimum *support count* ξ akan dibandingkan dengan *conditional /* FP-tree.

3. Apabila *Conditional* FP-tree merupakan lintasan tunggal (*single path*), maka didapatkan *frequent itemset* dengan melakukan kombinasi *item* untuk setiap *conditional* FP-tree. Jika bukan lintasan tunggal, maka dilakukan pembangkit FP-growth secara rekursif.

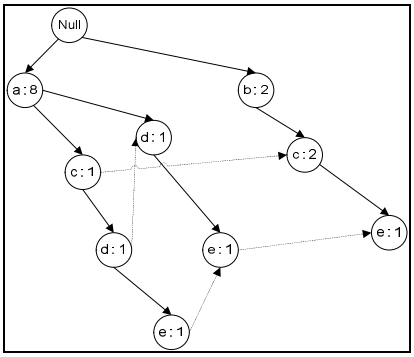
Ketiga tahap tersebut merupakan langkah yang akan dilakukan untuk mendapatkan *frequent itemset.* Algoritma FP Growth dapat dilihat pada gambar 2.6.

Input: FP-tree *Tree*
Outout: R_t sekumpulan lengkap pola frequent
Method: FP-growth (*Tree*,null)
Procedure: FP-growth (*Tree*,α)
{
01: **if** *Tree* mengandung *single path* P;
02: **then** untuk tiap kombinasi (dinotasikan β) dari
node-node dalam *path* P **do**
03: bangkitkan pola β α dengan Support = minimum
support dari node-node dalam β;
04: **else** untuk tiap a_i dalam header dari *Tree* **do** {
05: bangkitkan pola
06: bangun $\beta = a_i$ α dengan
support = a_i . support
07: **if** *Tree* β=Ø
08: **then** panggil FP-growth (*Tree*, β) }
}

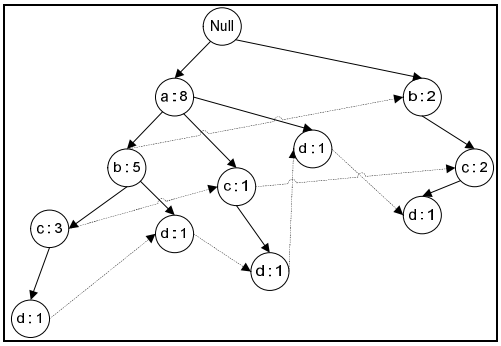
Gambar 2.6 Algoritma FP Growth

Akan dicoba menerapkan algoritma FP-growth pada kasus contoh 1. Langkah-langkah yang harus d*item*puh akan dijelaskan pada bagian berikut ini.

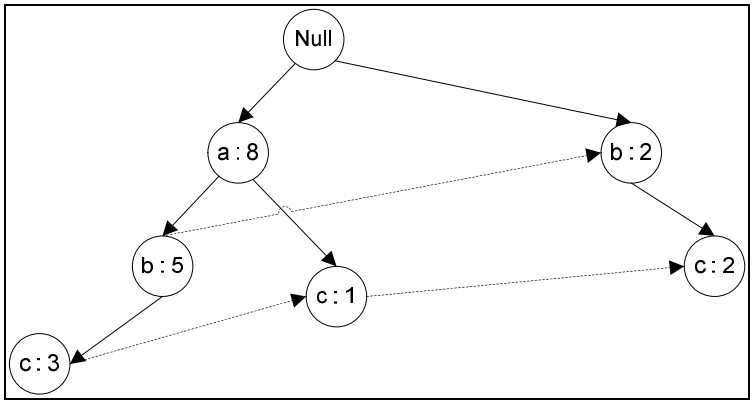
Untuk menemukan *frequent itemset* dari contoh 1, maka perlu ditentukan cabang pohon dengan lintasan yang berakhiran dengan *support count* terkecil, yaitu e. Berturut-turut ditentukan juga yang berakhiran di d, c, b dan a. Proses pembentukan dapat dilihat pada gambar 2.7 berikut:



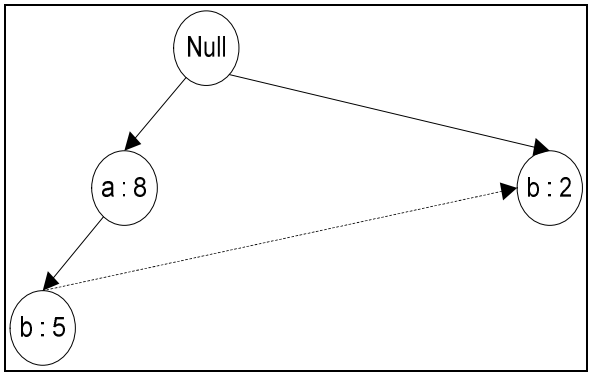
Gambar 2.7 Lintasan Yang Mengandung Simpul e



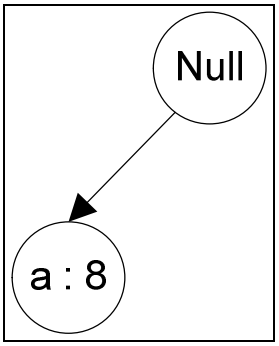
Gambar 2.8 Lintasan Yang Mengandung Simpul d



Gambar 2.9 Lintasan Yang Mengandung Simpul c



Gambar 2.10 Lintasan Yang Mengandung Simpul b



Gambar 2.11 Lintasan Yang Mengandung Simpul a

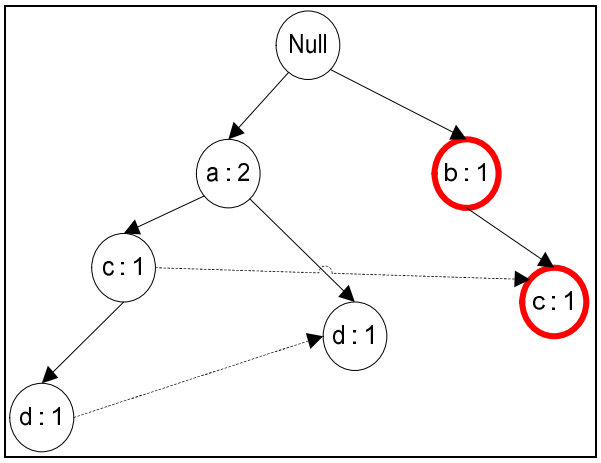
Algoritma FP-growth menemukan *frequent itemset* yang berakhiran *suffix* tertentu dengan menggunakan metode *divide and conquer* untuk memecah *problem* menjadi *subproblem* yang lebih kecil.

Contohnya, jika kita ingin menemukan semua *frequent itemset* yang berakhiran e. Oleh karena itu, kita harus mengecek apakah *support count* dari e memenuhi minimum *support count* ξ=2, karena *support count* dari e adalah 3 dan $3 \geq \xi$, maka e adalah *item* yang *frequent*.

Setelah mengetahui bahwa *item* e adalah *item* yang *frequent*, maka *subproblem* selanjutnya adalah menemukan *frequent itemset* dengan akhiran de, ce, be, dan ae. Dengan menggabungkan seluruh solusi dari *subproblem* yang ada, maka himpunan semua *frequent itemset* yang berakhiran *item* e akan didapatkan.

Untuk lebih memperjelas, dapat dilihat contoh menemukan *frequent itemset* yang berakhiran dengan *item* e di bawah ini:

1. Langkah pertama yang dilakukan adalah membangun sebuah cabang FP-tree dengan hanya menyertakan lintasan yang berakhir di e.
2. *Support count* dari *item* e dihitung dan dibandingkan dengan minimum *support count*. Karena memenuhi, maka {e} termasuk *frequent itemset*, karena *support count* dari *item* e = 3.
3. Karena *item* e *frequent*, maka perlu dipecahkan *subproblem* untuk menemukan *frequent itemset* yang berkahiran dengan de, ce, be, dan ae. Sebelum memecahkan *subproblem* ini, maka cabang FP-tree tersebut harus diubah terlebih dahulu menjadi *conditional* FP-tree. *Conditional* FP-tree mirip dengan FP-tree biasa, namun *conditional* FP-tree dimaksudkan untuk mencari *frequent itemset* yang berakhiran *item* tertentu.

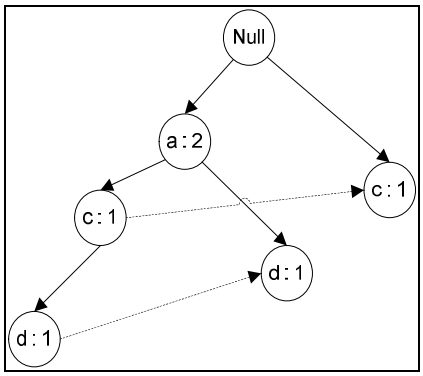


Gambar 2.12 Lintasan Setelah Simpul e Dibuang

4. *Conditional* FP-tree dapat dibentuk dengan cara :
   a. Setiap lintasan yang tidak mengandung e dibuang. Pada contoh, lintasan terkanan, terdapat lintasan yang tidak mengandung e, yaitu null→b→c. Lintasan ini dapat dibuang dengan cara mengurangi *support count* menjadi 1, sehingga lintasan tersebut hanya mengandung transaksi {b, c, e}, seperti pada gambar 2.12.

b. Setelah semua lintasan berakhir di e, maka simpul e dapat dibuang, karena setiap nilai *support count* pada simpul orang tuanya telah mencerminkan transaksi yang berakhir di e. *Subproblem* selanjutnya yang harus dipecahkan adalah mencari lintasan *frequent itemset* yang berakhir di de, ce, be, dan ae.

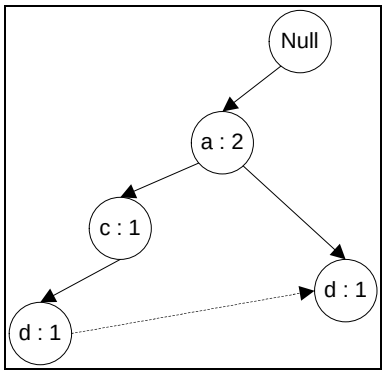
c. Karena nilai *support count* dari b adalah 1, yang berarti transaksi yang mengandung b dan e hanya 1 transaksi, makasimpul b dan lintasan yang mengandung be dapat dibuang, karena jika *item* b tidak *frequent*, maka setiap transaksi yang berakhiran be juga tidak *frequent*. Terbentuk Conditional FP-tree untuk e, seperti pada gambar 2.13 berikut:



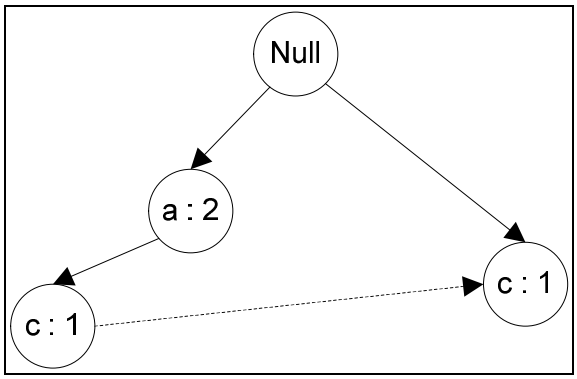
Gambar 2.13 *Conditional* FP-tree Untuk e

5. FP-tree menggunakan *Conditional* FP-tree untuk membangun pohon lintasan *prefix* untuk menemukan *frequent itemset* yang berakhir dengan pasangan *item* de,ce, dan ae.

6. Untuk lintasan *prefix* de, yang dibentuk dari *Conditional FP-tree* untuk *item* e dapat dilihat pada gambar 2.14.



Gambar 2.14 Pohon *Prefix* Yang Berakhiran de



Gambar 2.15 Pohon *Prefix* Yang Berakhiran ce

7. Dengan menjumlahkan *support count* dari d, yang tidak lain adalah jumlah *frequent itemset* yang berakhir di de, didapat bahwa {d, e} juga termasuk dalam *frequent itemset.*
8. Selanjutnya algoritma FP-tree akan mengulangi langkah yang sama dengan langkah ketiga, sehingga didapatkan *conditional FP-tree* untuk de hanya berisi satu daun, yaitu a, dengan *support count* 2. Sehingga {a, d, e} termasuk dalam *frequent itemset.*
9. *Subproblem* berikutnya yaitu dengan menemukan *frequent itemset* yang berakhiran dengan ce. Didapat {c, e} juga merupakan *frequent itemset*, begitupula dengan {a, e}.

Setelah memeriksa *frequent itemset* untuk beberapa akhiran (*suffix*), maka didapat hasil yang dirangkum dalam tabel berikut:

Tabel 2.4 Hasil *Frequent Item*set

Suffix	***Frequent Itemset***
e	{e}, {d,e}, {a,d,e}, {c,e}, {a,e}
d	{d}, {c,d}, {b,c,d}, {a,c,d}, {b,d}, {a,b,d}, {a,d}
c	{c}, {b,c}, {a,b,c}, {a,c}
b	{b}, {a,b}
a	{a}

Dengan metode *divide and conquer* ini, maka pada setiap langkah rekursif, algoritma FP-growth akan membangun sebuah *conditional* FP-tree baru yang telah diperbaharui nilai *support count*, dan membuang lintasan yang mengandung *item-item* yang tidak *frequent* lagi.

Setelah didapatkan *frequent itemset*, selanjutnya adalah membuat *rule* dengan cara menghitung *confidence*-nya. Dari 19 *itemset* yang dihasilkan pada tabel 2.4, tidak semua dihitung karena *rule* yang dihasilkan adalah jika membeli barang A, maka akan membeli barang B, maka *itemset* yang dihitung minimal berisi dua item. Maka yang akan dihitung *confidence*-nya adalah 14 subsets, yaitu: {a,d,e}, {b,c,d}, {a,c,d}, {a,b,d}, {a,b,c}, {d,e}, {c,e}, {a,e}, {c,d}, {b,d}, {a,d}, {b,c}, {a,c} dan {a,b}.

Hanya kombinasi yang lebih besar sama dengan *minimum confidence* yang akan diambil atau *strong association rule* nya saja. Adapun rumus menghitung *confidence* adalah:

$$\text{Confidence} \quad A \rightarrow B \qquad \frac{support_count \quad A \cup}{support_count \quad A} \tag{2.1}$$

Dimana ($A \cup$) pada rumus 2.1 bukan menunjukan transaksi yang mengandung A atau B, melainkan menunjukan transaksi yang mengandung A dan B. Sebagai contoh akan dihitung *confidence* untuk *itemset* {a,d,e} sebagai berikut:

- a^d → e = 2/4 = 50%
- e → a^d = 2/3 = 66,7%

- a → d^e = 2/8 = 25%
- d^e → a = 2/2 = 100%
- a^e → d = 2/2 = 100%
- d → a^e = 2/5 = 40%
- a → d = 4/8 = 50%
- d → a = 4/5 = 80%
- d → e = 2/5 = 40%
- e → d = 2/3 = 66,7%
- a → e = 2/8 = 25%
- e → a = 2/3 = 66,7%

Jika dimisalkan *minimum confidence* adalah 75 %, *minimum confidence* ini diambil diatas 75% agar tingkat akurasinya tinggi, maka yang termasuk *strong association rule* adalah d^e → a, a^e → d dan d → a, yang artinya jika konsumen membeli item d dan e, maka konsumen membeli item a juga, jika konsumen membeli item a dan e, maka konsumen membeli item d juga dan jika konsumen membeli item d, maka konsumen membeli item a juga. Berikut tabel yang berisi hasil lengkap pola-pola atau *rules* yang dihasilkan:

Tabel 2.5 *Strong Association Rule*

Jika membeli	***Maka akan membeli***	***Confidence***
d dan e	a	100%
a dan e	d	100%
d	a	80%
c	b	83,3%
a dan c	b	75%

2.8 PHP

PHP (*Hypertext Preprocessor*) adalah sebuah bahasa pemrograman di sisi *server*. Ketika mengakses sebuah URL, maka *web browser* akan melakukan request ke sebuah *web server*. Misalnya kita me-*request* sebuah file PHP dari http://www.koder.web/index.php, maka *web server* akan melakukan parsing

terhadap file PHP tersebut. PHP *parser* yang menjalankan kode-kode PHP yang terdapat pada file index.php lalu mengirimkan hasilnya ke *web browser*.

Ketika ingin menggunakan PHP, hal pertama yang harus disiapkan adalah sebuah *web server*, seperti yang sudah dibahas sebelumnya bahwa PHP merupakan bahasa pemrograman disisi *server*. Banyak sekali jenis *web server* yang dapat digunakan, antara lain Apache, IIS, iPlanet, Omnni, Xintami dan lain-lain.

PHP dapat diperoleh secara gratis dengan cara men-*download* dari situs resmi PHP di http://www.php.net/download.php.

Beberapa dasar-dasar PHP yang dipakai pada aplikasi ini antara lain:

1. Elemen dasar PHP yang mencakup pengenal, tipe data, variabel, konstanta dan juga operator.
2. Pernyataan kontrol digunakan terkait dengan pengambilan keputusan. Ada banyak cara dalam melakukan pernyataan kontrol. Diantaranya dengan menggunakan pernyataan if, switch, operator “?”, while, do-while, for, break, continue, exit dan beberapa sintak alternatif lainnya.
3. *Array* merupakan suatu struktur data atau wadah yang dapat menyusun sekumpulan variabel yang memiliki tipe data yang sama.
4. Fungsi merupakan blok kode yang ditujukan untuk melaksanakan suatu tugas tertentu. Fungsi ada yang memiliki parameter dan ada juga yang tidak memiliki parameter. Pada PHP sebenarnya sudah tersedia banyak fungsi-fungsi standar PHP. Akan tetapi dapat juga dibuat suatu fungsi bentukan sesuai dengan keinginan.

2.9 *Object Oriented Programming* (OOP)

Object Oriented Programming (OOP) merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi pada paradigma ini dibungkus dalam kelas-kelas atau objek-objek. Bandingkan dengan logika pemrograman terstruktur, setiap objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya. OOP diciptakan untuk mengatasi keterbatasan pada bahasa pemrograman tradisional. Konsep dari OOP sendiri

adalah semua pemecahan masalah dibagi ke dalam objek. Dalam konsep OOP data dan fungsi-fungsi yang akan mengoperasikannya digabungkan menjadi satu kesatuan yang dapat disebut sebagai objek.

2.10 UML (*Unified Modeling Language*)

Unified Modelling Language (UML) adalah sebuah standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem. [2]

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#, VB.NET atau PHP. [20]

Berikut akan dijelaskan 4 macam diagram yang akan digunakan dalam pembangunan aplikasi ini, yaitu *use case diagram, sequence diagram, activity diagram,* dan *class diagram.*

2.10.1 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 atau sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu. [20]

Use case diagram dapat sangat membantu bila sedang menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem.

Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang

di-*include* akan dipanggil setiap kali *usecase* yang meng-*include* dieksekusi secara normal. Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*.

Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

2.10.2 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 atar 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. [20]

Masing-masing objek, termasuk aktor, memiliki *lifeline* vertikal. *Message* digambarkan sebagai garis berpanah dari satu objek ke objek lainnya. Pada fase desain berikutnya, *message* akan dipetakan menjadi operasi/metoda dari *class*. *Activation bar* menunjukkan lamanya eksekusi sebuah proses, biasanya diawali dengan diterimanya sebuah *message*. Untuk objek-objek yang memiliki sifat khusus, standar UML mendefinisikan *icon* khusus untuk objek *boundary*, *controller* dan *persistent entity*.

2.10.3 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. [20]

2.10.4 Activity Diagram

Activity diagram menggambarkan berbagai alur 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 behaviour internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas.

Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan behaviour pada kondisitertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasiyang dapat berupa titik, garis horizontal atau vertikal. *Activity diagram* dapat dibagi menjadi beberapa *object swimlane* untuk menggambarkan objek mana yang bertanggung jawab untuk aktivitas tertentu. [20]

2.11 *Framework CodeIgniter* dan Konsep *Model View Controller* (MVC)

Framework adalah sebuah struktur konseptual dasar yang digunakan untuk memecahkan sebuah permasalahan, bahkan isu-isu kompleks yang ada. Sebuah *framework* telah berisi sekumpulan arsitektur atau konsep-konsep yang dapat mempermudah dalam pemecahan sebuah permasalahan. *Framework* hanya menjadi sebuah konstruksi dasar yang menopang sebuah konsep atau sistem yang bersifat "*essential support*" (penting tapi bukan komponen utama). [4]

CodeIgniter adalah sebuah *web application framework* yang bersifat *open source* digunakan untuk membangun aplikasi PHP dinamis. [4] CodeIgniter sangat ringan, terstruktur, mudah dipelajari, dokumentasi lengkap dan dukungan yang luar biasa dari forum CodeIgniter. Selain itu CodeIgniter juga memiliki fitur-fitur lainnya yang sangat bermanfaat, antara lain:

1. Menggunakan *pattern* MVC.

   Dengan menggunakan *pattern* MVC ini, struktur kode yang dihasilkan menjadi lebih terstruktur dan memiliki standar yang jelas.

2. URL Friendly

   URL yang dihasilkan sangat *url friendly*. Pada CodeIgniter diminimalisasi penggunaan $_GET dan digantikan dengan URI.

3. Kemudahan

   Kemudahan dalam mempelajari, membuat *library* dan *helper*, memodifikasi serta meng-integrasikan *library* dan *helper*.

Jika membandingkan antara CodeIgniter dengan *framework-framework* lainnya maka beberapa poin yang membuat CodeIgniter unggul adalah:

1. Kecepatan

   Berdasarkan hasil *benchmark* CodeIgniter merupakan salah satu *framework* PHP tercepat yang ada saat ini.

2. Mudah dimodifikasi dan beradaptasi

   Sangat mudah memodifikasi *behavior framework* ini. Tidak membutuhkan *server requirement* yang macam-macam serta mudah mengadopsi *library* lainnya.

3. Dokumentasi lengkap dan jelas

   Bahkan tanpa buku ini pun CodeIgniter sebenarnya telah menyediakan sebuah panduan yang lengkap mengenai CodeIgniter. Semua informasi yang anda butuhkan tentang CodeIgniter ada disana.

4. *Learning Curve* Rendah

   CodeIgniter sangat mudah dipelajari. Dalam pemilihan *framework* hal ini sangat penting diperhatikan karena kita harus juga memperhatikan skill dari seluruh anggota team. Jika sebuah *framework* sangat sulit dipelajari maka akan beresiko untuk memperlambat *team development*.

MVC adalah konsep dasar yang harus diketahui sebelum mengenal CodeIgniter. MVC adalah singkatan dari *Model View Controller*. MVC sebenarnya adalah sebuah *pattern* atau teknik pemrograman yang memisahkan *bisnis logic* (alur pikir), *data logic* (penyimpanan data) dan *presentation logic* (antarmuka aplikasi) atau secara sederhana adalah memisahkan antara desain, data dan proses. [4] Adapun komponen-komponen MVC antara lain:

1. Model

   Model berhubungan dengan data dan interaksi ke *database* atau *webservice*. Model juga merepresentasikan struktur data dari aplikasi yang bisa berupa basis data maupun data lain. Biasanya di dalam model akan berisi *class* dan fungsi untuk mengambil, melakukan *update* dan menghapus data *web*site. Sebuah aplikasi *web* biasanya menggunakan basis data dalam menyimpan data, maka pada bagian model biasanya akan berhubungan dengan perintah-perintah query SQL.

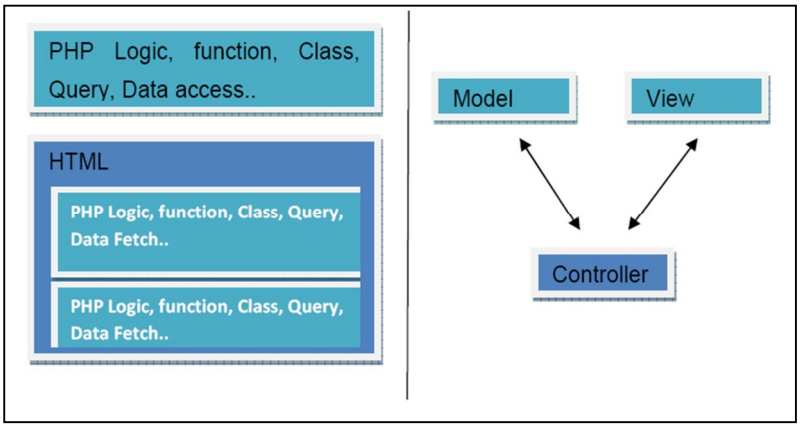
2. *View*

   View berhubungan dengan segala sesuatu yang akan ditampilkan ke *end-user*. Bisa berupa halaman *web*, rss, javascript dan lain-lain. Kita harus menghindari adanya logika atau pemrosesan data di *view*. Di dalam *view* hanya berisi variabel-variabel yang berisi data yang siap ditampilkan. *View* dapat dikatakan sebagai halaman *website* yang akan dibuat dengan menggunakan HTML dan bantuan CSS atau JavaScript. Di dalam *view* jangan pernah ada kode untuk melakukan koneksi ke basisdata. *View*

hanya dikhususkan untuk menampilkan data-data hasil dari model dan *controller*.

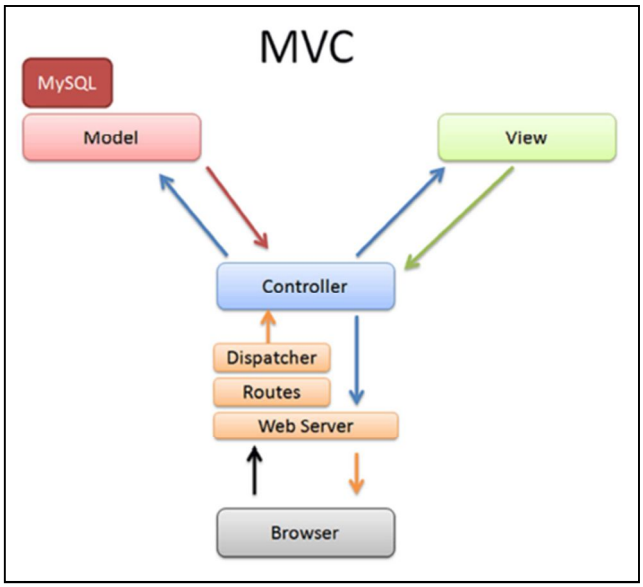
3. *Controller*

   Controller bertindak sebagai penghubung data dan *view*. Di dalam *controller* inilah terdapat *class-class* dan fungsi-fungsi yang memproses permintaan dari *view* ke dalam struktur data di dalam model. *Controller* juga tidak boleh berisi kode untuk mengakses basis data karena tugas mengakses data telah diserahkan kepada model. Tugas *controller* adalah menyediakan berbagai *variabel* yang akan ditampilkan di *view*, memanggil model untuk melakukan akses ke basis data, menyediakan penanganan kesalah atau *error*, mengerjakan proses logika dari aplikasi serta melakukan validasi atau cek terhadap input.



Gambar 2.16 Perbandingan PHP Biasa Dengan CodeIgniter

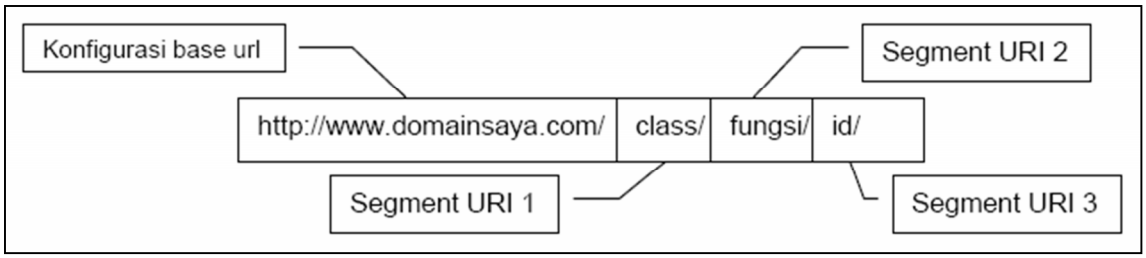
Pada gambar 2.16, pada eksekusi PHP biasanya kita akan me-*load* semua *library* dan fungsi yang dibutuhkan kemudian digabungkan ke dalam HTML untuk di eksekusi oleh PHP. Untuk kasus sederhana cara tersebut masih baik-baik saja, tetapi ketika aplikasi tersebut menjadi kompleks atau rumit maka kita akan sulit memeliharanya jika tidak didukung oleh arsitektur *software* yang bagus. Hal tersebut bisa terjadi disebabkan oleh kode yang sama namun dibuat berulang-ulang, kode tidak konsisten dan lain-lain.



Gambar 2.17 Alur Kerja CodeIgniter

Jika dipetakan, alur kerja CodeIgniter akan tampak seperti gambar 2.17. *Browser* berinteraksi melalui *controller*. *Controller*-lah yang akan menerima dan membalas semua *request* dari *browser*. Untuk data maka *controller* akan meminta ke model dan untuk *user interface* atau *template* akan meminta ke *view*. Jadi "otak" dari aplikasi ada di *controller*, "muka" aplikasi ada di *view* dan "data" ada di model. Ketika *browser* meminta sebuah halaman *web* maka router akan mencarikan *controller* mana yang harus menangani request tersebut. Setelah itu barulah si *controller* menggunakan model untuk mengkases data dan *view* untuk menampilkan data tersebut.

Segmen-segmen pada URL pada CodeIgniter mencerminkan *controller* yang dipanggil. Contoh: http://www.domainsaya.com/*class*/fungsi/id maka domain tersebut dapat dipecah menjadi bagian-bagian seperti pada gambar 2.18.



Gambar 2.18 Bagian – bagian URL pada CodeIgniter

Adapun bagian-bagian URL diatas adalah:

1. Konfigurasi *Base* URL

   Bagian base url merupakan url yang kita masukkan pada konfigurasi base_url yang merupakan url paling dasar untuk mengakses *web* atau aplikasi kita.

2. Segmen URL 1

   Segmen url 1 yaitu *class*. *Class* tersebut merupakan nama kelas *controller* yang akan kita panggil. Apabila segmen ini kosong maka akan digantikan dengan *default controller* yang telah disetting di konfigurasi router.php.

3. Segmen URL 2

   Segmen url 2 yaitu fungsi dari *class controller* yang telah kita panggil tadi. Apabila segmen kedua ini kosong maka fungsi yang dipanggil adalah fungsi index dari *class controller* tersebut.

4. Segmen URL 3

   Segmen url 3 biasanya berisi parameter dari fungsi. Jika fungsi dari *controller* yang dipanggil mempunyai parameter maka parameternya harus dimasukkan sebagai segmen url sesuai urutan.

2.12 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 atau seleksi

dan pemasukan data yang memungkinkan pengoperasian data dikerjakan dengan mudah secara otomatis.

Kelebihan MySQL dibandingkan RDBMS lainnya adalah sebagai berikut:

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 user 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 olpertor dan fungsi secara penuh yang mendukkung perintah SELECT dan WHERE dalam query
7. **Security.** memiliki lapisan sekuritas, seperti level subnetmask, nama host dan izin akses user disertai dengan password enkripsi.
8. **Scalability dan Limits.** Mampu menangani database 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. [22]