BAB I

PENDAHULUAN

1.1 LATAR BELAKANG

Basis data (*database*) merupakan mekanisme yang digunakan untuk menyimpan informasi atau data [12]. Bagi setiap institusi maupun perusahaan, basis data sangat penting karena basis data tidak hanya mempercepat perolehan informasi, tetapi juga dapat meningkatkan daya saing terhadap perusahaan lain. Hal inilah yang mendorong banyak perusahaan yang menggunakan pemrosesan manual mulai beralih memanfaatkan basis data yang terkomputasi.

Terdapat beberapa model basis data, salah satu model basis data yang banyak digunakan saat ini adalah model basis data relasional [12]. Dalam pengelolaan basis data tersebut, institusi dan perusahaan menggunakan *Database Managemen System* (DBMS) karena kelebihannya dalam hal pemulihan (*recovery*), kecepatan mengambil data (*performance*), dan keamanan (*security*) [12]. DBMS sendiri merupakan suatu program komputer yang digunakan untuk memasukkan, mengubah, menghapus, memanipulasi, dan memperoleh data / informasi dengan praktis dan efisien [1].

Sebuah DBMS memperlakukan data sesuai dengan sifat dan strukturnya hanya di dalam DBMS itu sendiri. Setelah data tersebut dipanggil, data akan kehilangan sifat dan strukturnya [2]. Hal ini dapat dipecahkan dengan cara mendeskripsikan data kembali, yaitu setelah diambil dengan menggunakan pemrograman, misalkan memakai PHP, ASP, atau yang lain. Namun, cara ini tidak praktis dan bermasalah dalam hal pengembangannya, karena pengembang

harus membuat program deskripsi data setiap kali data ditransferkan. Selain itu, pada umumnya tiap *vendor* DBMS menggunakan tipe data yang berbeda-beda untuk mendeskripsikan data. Sehingga terdapat kesulitan semisal suatu data yang disimpan dalam DBMS Oracle akan disimpan / dikirim dalam DBMS MySQL.

Di tengah masalah ini, lahir sebuah teknologi *cross platform*, yaitu *eXtensible Markup Language* (XML). Format XML berbasis teks, hal inilah yang menyebabkan XML dengan mudah dapat memindahkan data antar *platform* dan dapat berpindah melalui Internet [6]. XML dapat digunakan untuk pertukaran informasi antara sistem-sistem yang terpisah jauh, seperti pada aplikasi *Busines-to-Busines* (B2B). Dengan struktur dan definisi yang jelas, XML dapat dipakai merepresentasikan dan mengkomunikasikan basis data relasional yang tersebar [11].

Dokumen XML yang berukuran besar menyebabkan proses pengambilan data (*query*) menjadi tidaklah mudah, karena setiap proses *query* harus me-*parsing* (mengurai) dokumen XML dari awal yang memerlukan banyak tahapan dan waktu [5]. Dalam mengurai dokumen XML yang berukuran besar, dapat ditangani dengan menggunakan metode parsing SAX untuk lebih efisien [7].

Penyimpanan data XML dalam suatu model basis data relasional yang kemudian diolah dengan menggunakan suatu DBMS merupakan alternatif penyimpanan data untuk mempermudah, mempercepat proses query, dan memberikan banyak keuntungan. Namun Penyimpanan dokumen XML menggunakan RDBMS (DBMS relasional) adalah suatu masalah tersendiri karena seseorang diharuskan untuk dapat menangani perbedaan model data XML yang bersifat hirarki dengan model data relasional yang cenderung bersifat flat.

1.2 PERUMUSAN MASALAH

Dari uraian masalah di atas, masalah yang dapat dirumuskan adalah bagaimana menyimpan model data XML yang bersifat hirarki ke dalam model data relasional yang cenderung bersifat flat dengan menggunakan metode parsing SAX, sehingga mempermudah pertukaran data, yaitu dengan menggunakan XML yang kemudian disimpan dalam DBMS untuk mempermudah, mempercepat proses query, dan memberikan banyak keuntungan dari sisi manajemen data dan keamanannya.

1.3 BATASAN MASALAH

1. Dokumen XML yang akan ditransformasikan ke model basis data relasional adalah XML valid-tipe dengan DTD (*Document Type Definition*) *external*.
2. Aplikasi yang dibuat tidak mengecek validitas suatu dokumen XML.
3. Hasil transformasi berupa file DDL (*Data Definition Language*) dan file DML (*Data Manipulation Language*) yang dapat diterapkan di dalam DBMS MySQL.
4. Perangkat-perangkat lunak yang digunakan adalah bahasa pemrograman PHP dan DBMS yang digunakan adalah MySQL yang terintegrasi dalam software Xampp platform Windows.

1.4 TUJUAN

Membuat aplikasi berbasis web yang dapat mentransformasikan dokumen XML menjadi model basis data relasional dengan menggunakan metode parsing SAX, sehingga hasil transformasi tersebut dapat mempermudah proses query dengan memanfaatkan DBMS seperti MySQL.

1.5 SISTEMATIKA PENULISAN

Sistematika penulisan yang dilaksanakan dalam penulisan Tugas Akhir ini dapat disusun sebagaimana berikut :

BAB I : PENDAHULUAN

Pendahuluan memberikan gambaran sekilas tentang bagaimana latar belakang masalah, perumusan masalah, pembatasan masalah, tujuan, dan sistematika penulisan tugas akhir ini.

BAB II : LANDASAN TEORI

Berisikan dasar teori yang mendasari keseluruhan topik tugas akhir ini, perangkat-perangkat lunak yang akan dipergunakan dalam pembangunan fasilitas transformasi data XML ke basis data relasional.

BAB III : ANALISIS KEBUTUHAN DAN PERANCANGAN

Menjelaskan analisis dari kebutuhan dan perancangan aplikasi transformasi dokumen XML ke dalam model basis data relasional.

BAB IV : IMPLEMENTASI DAN PENGUJIAN

Bab ini berisi tentang implementasi dan pengujian perangkat lunak yang telah dibuat beserta hasilnya.

BAB V : PENUTUP

Bab ini secara khusus memberikan kesimpulan yang dapat diambil dari tugas akhir yang telah disusun.

BAB II

LANDASAN TEORI

2.1 Basis Data

Basis data menurut Stephens dan Plew adalah mekanisme yang digunakan untuk menyimpan informasi atau data [12]. Dengan basis data, pengguna dapat menyimpan data secara terorganisasi. Setelah data tersimpan, informasi harus mudah diambil. Cara data disimpan menentukan seberapa mudah mencari suatu informasi berdasarkan kriteria yang ada. Data pun harus mudah ditambahkan ke dalam basis data, dimodifikasi, dan dihapus.

2.1.1 Bahasa Basis Data

Sistem basis data menyediakan bahasa pendefinisian data (*Data Definition Language*) untuk menentukan skema basis data dan bahasa manipulasi data (*Data Manipulation Language*) untuk menyatakan *query* dan *update* basis data. Pada praktiknya, *Data Definition Language* (DDL) dan *Data Manipulation Language* (DML) bukan merupakan dua bahasa yang terpisah, melainkan membentuk bagian bahasa basis data. Untuk saat ini bahasa basis data yang paling populer adalah *Sructured Query Language* (SQL). Selain SQL terdapat pula bahasa yang lain seperti QUEL dan dBase [3].

2.1.1.1 *Data Definition Language*

Data Definition Language adalah perintah-perintah yang digunakan untuk mendefinisikan skema ke DBMS [1]. Sebagai contoh, pernyataan berikut dalam bahasa SQL mendefinisikan tabel rekening.

```
Create table rekening (no_rekening char(10), saldo integer);
```

Eksekusi pernyataan DDL tersebut akan membuat sebuah tabel rekening dengan 2 kolom, yaitu no_rekening dan saldo.

2.1.1.2 *Data Manipulation Language*

Yang termasuk dalam manipulasi data adalah [12]:

a. Pengambilan informasi yang disimpan dalam basis data.
b. Penyisipan informasi baru dalam basis data.
c. Penghapusan informasi dalam basis data.
d. Modifikasi informasi yang disimpan dalam basis data.

Data Manipulation Language adalah bahasa yang memungkinkan pengguna mengakses atau memanipulasi data seperti yang diatur oleh model data. Ada dua tipe DML, yaitu [12]:

1. DML Prosedural, mengharuskan pengguna untuk menentukan data yang dibutuhkan dan bagaimana mendapatkannya.
2. DML deklaratif / DML nonprosedural, mengharuskan pengguna menentukan data yang dibutuhkan tanpa menentukan bagaimana mendapatkannya.

Komponen DML dari bahasa SQL adalah DML deklaratif.

Query merupakan permintaan yang diberikan oleh user untuk mengambil informasi yang tersimpan dalam basis data. Bagian DML yang terlibat dalam pengambilan informasi disebut bahasa query. Istilah *query* sering disamakan dengan istilah bahasa manipulasi data. Berikut ini diberikan contoh tabel pegawai dan tabel pesan pada tabel 2.1 dan tabel 2.2 yang akan diberikan *query*.

Tabel 2.1 Contoh Tabel Pegawai

KdPegawai	Nama	Alamat	Kota
001	Gerrard	Jl. Kenanga 10	Batang
002	Frank	Jl. Tulua 8	Tegal
003	Torres	Jl. Asri C 9	Blora

Tabel 2.2 Contoh Tabel Pesan

KdBarang	NmBarang	KdPegawai
3346	Sepatu	003
3344	Jaket	001
3321	Barbel	002

Query dalam bahasa SQL berikut mencari nama pegawai dengan KdPegawai = 001:

```
Select Pegawai.Nama from Pegawai where Pegawai.KdPegawai=001
```

Query dapat juga melihat informasi lebih dari satu tabel. Sebagai contoh, *query* berikut ini mencari NmBarang yang dipesan oleh pegawai dengan KdPegawai = 002:

```
Select Pegawai.Nama, Pesan.NmBarang from Pegawai, Pesan where
Pegawai.KdPegawai=Pesan.KdPegawai and Pegawai.KdPegawai=002
```

2.1.2 Model Basis Data Relasional

Ada beberapa tipe basis data, beberapa basis data bertipe sederhana, sedangkan yang lainnya sangat kompleks. Salah satu tipe model basis data yang banyak digunakan adalah model basis data relasional [12].

Unit penyimpanan utama dalam basis data relasional adalah tabel atau kelompok data yang saling berhubungan. Sebuah tabel terdiri atas baris dan kolom. Baris berhubungan dengan record dalam tabel dan kolom mengandung nilai semua baris yang berhubungan dengan *field* tertentu. Tabel dapat dihubungkan satu sama lain melalui nilai kolom yang disebut kunci *(key)* [12].

Hubungan antartabel ditentukan oleh integritas referensial (*referential integrity)* yang memerlukan penggunaan batasan kunci utama *(primary key)* dan kunci tamu *(foreign key)*. Integritas referensial adalah seperangkat aturan yang mengatur hubungan antara kunci utama dengan kunci tamu milik tabel-tabel yang berada dalam suatu basis data relasional untuk menjaga konsistensi data. Tipe batasan lain dapat pula dibuat untuk mengontrol data yang bisa dimasukkan dalam kolom tertentu dan membuat hubungan antartabel.

Keuntungan model basis data relasional adalah [12] :

a. Data dapat diakses secara cepat.
b. Struktur basis data mudah diubah.
c. Data disajikan secara logis sehingga pengguna tidak perlu mengetahui bagaimana data disimpan.
d. Pengguna mudah membuat query yang kompleks untuk mengambil data.
e. Pengguna mudah menerapkan integritas data.
f. Pengguna mudah membuat dan memodifikasi program aplikasi.
g. Bahasa standar (SQL) sudah dibuat.

Kekurangan model basis data relasional adalah [12]:

a. Kelompok informasi atau tabel yang berbeda harus dihubungkan untuk mengambil data.
b. Pengguna harus memahami hubungan antartabel.
c. Pengguna harus belajar SQL.

2.2 *Extensible Markup Language*

Extensible Markup Language (XML) merupakan sebuah subset dari *Generalized Markup Language* (SGML), tetapi dengan standar dan kemampuan ekstra yang telah ditambahkan pada pemrosesannya, dan saat ini teknologi XML merupakan suatu hibrida dari pemrosesan dokumen dan basis data [6]. Sebagai bahasa markup, XML sangat jauh lebih baik daripada HTML yang merupakan bahasa dari *world wide web* (www) yang dipergunakan untuk menyusun dan membentuk dokumen agar dapat ditampilkan pada program *browser* [13]. Ada beberapa alasan atas keunggulan XML. Pertama, para desainer XML membuat pemisah yang jelas antara struktur, isi, dan perwujudan dokumen. XML memiliki fasilitas untuk menangani masing-masing hal tersebut, dan sifat dasar dari fasilitas-fasilitas tersebut tidak dapat dicampurbaurkan [12].

2.2.1 Anatomi Dokumen XML

Sebuah dokumen XML memiliki tiga bagian, yaitu : *prolog*, elemen *root* (*root element*) atau disebut *document element*, dan *epilog* [13]. Namun sebuah dokumen XML, utamanya memiliki dua bagian yaitu prolog dan elemen dokumen atau elemen *root* [13].

2.2.1.1 Prolog

Sebuah prolog dalam sebuah dokumen XML tidaklah harus ada. Akan tetapi, prolog dapat memberikan informasi penting pada program aplikasi (misal browser), tidak hanya apa yang ditampilkan, tetapi juga bagaimana menampilkannya. Prolog dapat juga membantu menjelaskan tujuan dan cakupan dokumen untuk para pembaca manusia.

Terdapat empat konsep dasar dalam prolog, yaitu deklarasi XML, komentar (*comment*), perintah pemrosesan (*processing instruction*), dan deklarasi tipe dokumen. Sehingga bagian-bagian prolog berisi [4]:

1. Deklarasi XML

Berikut ini contoh deklarasi XML yang menyatakan dokumen XML dengan nomor versinya : `<?xml version="1.0" encoding="UTF-8"?>` Pembukaan `<?` Dan penutupan `?>` merupakan pembatas standar dalam XML. Di luar angka versi tersebut, deklarasi XML memungkinkan juga berisi deklarasi pengkodean and deklarasi dokumen *standalone*.

Deklarasi pengkodean memberitahu program XML semisal *browser* tentang kumpulan karakter yang diperlukan untuk memroses dokumen tersebut. Deklarasi dokumen standlone mengambil nilai sederhana, yakni ya / tidak, yang memberitahu prosesor XML apakah dokumen dan aturannya sepenuhnya *self-contrained*, yaitu DTD untuk dokumen diletakkan pada beberapa file atau tidak.

Berikut ini deklarasi XML lengkap, dengan pengkodean dan deklarasi dokumen standalone : `<?xml version="1.0" encoding="UTF-8" standalone="yes"?>`

2. Komentar

Untuk memasukkan deskripsi tentang bagaimana sebuah kode, memperhatikan bagaimana menggunakannya, dan seterusnya, dapat digunakan sebuah komentar yang diawali dengan `<!--` dan diakhiri dengan

-->. Apapun yang terletak di antara tanda <!-- dan --> akan diabaikan oleh processor XML.

3. Perintah pemrosesan

Sebenarnya apa yang tercakup dalam perintah pemrosesan tergantung pada sifat software yang diharapkan mampu memproses dokumen, sebagai contoh : `<?realaudio version="5.0" frequency="5.5kHz" bitrate="16Kbps"?>` atau `<?xml-stylesheet type="text/css" href="pustaka01.cs"?>`

Kata yang mengikuti pembukaan `<?` (dalam kasus ini realaudio) disebut sebagai target perintah pemrosesan. Informasi yang mengikuti target hanya menggambarkan properti target.

4. Deklarasi tipe dokumen

Memberi tahu software pemrosesan tentang definisi tipe dokumen yang digunakan untuk memahami dokumen. Syntax umum dari deklarasi tipe dokumen: `<!DOCTYPE` *`name externalDTDpointer internalDTDsubset`*`>`

Hanya tiga *name* yang diperlukan dari tiga potong kode yang mengikuti kata kunci DOCTYPE :

a. *Name* secara umum hanyalah label untuk tipe dokumen. Secara khusus mengidentifikasi tag yang akan digunakan sebagai elemen root.

b. *externalDTDpointer*, pointer ini terdiri dari kata kunci SYSTEM, diikuti dengan *Uniform resource Identifier* (URI) yang

mengindikasikan di sistem. Sebagai contoh deklarasi doctype yang sederhana dengan referensi pada DTD eksternal : `<!DOCTYPE flixinfo> SYSTEM http://www.flixml.org/flixml.dtd>`

c. *InternalDTDsubset*, digunakan secara opsional atau sebagai tambahan pada DTD eksternal. Apapun yang dapat ditaruh ke dalam DTD eksternal dapat muncul secara internal. Jika terdapat DTD eksternal dan juga terdapat DTD internal, maka DTD internal akan lebih diutamakan oleh prosesor XML. Informasi DTD internal dipasang pada akhir deklarasi doctype dengan karakter `[` dan `]`, sebagaimana dalam contoh berikut : `<DOCTYPE flixinfo SYSTEM http://www.flixml.com/flixml.dtd [<!ELEMENT castinterview ANY>]>`

2.2.1.2 Elemen Root

Elemen root adalah komponen utama suatu dokumen XML. Tata cara penulisan elemen root harus memperhatikan struktur yang dibawa oleh DTD yang bersangkutan. Setiap elemen harus memenuhi beberapa ketentuan berikut [11]:

a. Dibatasi *tag* pembuka "<" dan penutup ">".
b. Nama elemen dan atributnya bersifat *case-sensitive*, yaitu huruf besar dan kecil dianggap beda.
c. Semua nama dalam elemen XML harus diawali huruf dan tidak boleh mengandung : `@`, `#`, `$`, `%`,`^`, `(`, `)`, `+`, `?`, `=`, `;`, `*`, dan kata '`xml`'.
d. Suatu dokumen XML berbentuk struktur *tree* yang hanya boleh memiliki satu akar. Akar simpul tersebut dapat diisi oleh elemen lain (bersarang).

e. Untuk penulisan karakter <, >, &, ", dan `, digunakan *entityreference*. Misalnya, untuk '>' digunakan `&gt`, untuk '`&`' digunakan `&amp`. Untuk mempermudah penulisannya, dapat digunakan CDATA. CDATA adalah suatu cara memasukkan tulisan yang mengandung karakter-karakter yang dapat dikenali sebagai suatu *markup*.

2.2.2 *Document Type Definition*

Document Type Definition (DTD) secara formal menyatakan struktur dan isi elemen (tag, hubungan di antara tag-tag berbeda, dan seterusnya) dari dokumen XML valid yang diberikan.

Dokumen XML dapat dikategorikan menjadi 3 jenis, yaitu [13]:

a. Well Formed XML: jika mengikuti spesifikasi / aturan XML namun tidak memiliki DTD / XML Schema
b. Valid XML: jika mengikuti spesifikasi / aturan XML dan memiliki & sesuai dengan DTD / XML Schema.
c. Invalid XML: jika tidak mengikuti aturan XML & tidak memiliki DTD.

DTD menggunakan tata bahasa umum untuk menspesifikasikan struktur dan nilai-nilai yang diijinkan dalam sebuah dokumen XML. DTD terdiri dari 2 komponen dasar, yaitu ELEMENT dan ATTLIST [17]:

2.2.2.1 ELEMENT

Setiap elemen yang digunakan dalam dokumen XML harus dinyatakan dengan menggunakan tag `<!ELEMEN>` di DTD. Syntax untuk mendeklarasikan sebuah elemen pada DTD adalah seperti berikut :

```
<!ELEMENT ElementName Rule>
```

Komponen *Rule* mendefinisikan aturan untuk isi yang terdapat dalam elemen. Aturan ini mendefinisikan struktur logis dari dokumen XML dan dapat digunakan untuk memeriksa validitas dokumen tersebut. Aturan dapat terdiri dari deklarasi umum dan satu atau lebih elemen.

Terdapat 3 deklarasi konten umum dalam XML : PCDATA, ANY, dan EMPTY

a. PCDATA

Deklarasi PCDATA dapat digunakan bila konten dalam suatu unsur hanya berupa teks (konten yang tidak mengandung unsur anak). Semisal :

```
<!ELEMENT title (#PCDATA)>
<!ELEMENT price (#PCDATA)>
```

b. ANY

Deklarasi ANY dapat mencakup konten teks dan elemen anak (*child element*). Sebagai contoh :

```
<!ELEMENT html ANY>
```

Deklarasi ANY memungkinkan konten apapun di dalam tag elemen, dalam contoh ini tag html dapat mengandung elemen *body* dan *head* ataupun berupa teks biasa :

```
<html> ini adalah dokumen HTML. <head/> <body/></html>
```

c. EMPTY

Deklarasi EMPTY memungkinkan elemen untuk tidak memiliki konten, baik konten elemen anak (*child element*) maupun konten teks biasa. Contoh :

```
<ELEMENT img EMPTY>
```

d. One or More Element

Untuk menghindari penggunaan deklarasi ANY untuk dapat memvalidasi dokumen XML, suatu elemen lebih baik untuk menetapkan konten pasti yang terdapat dalam elemen tersebut. Contoh :

```
<!ELEMENT html (head, body)>
```

Syntax di atas menunjukkan bahwa elemen *html* akan memiliki dua elemen anak : *head* dan *body*. Contoh di atas digunakan tanda penghubung koma untuk memisahkan elemen anak satu dengan lainnya.

Terdapat 2 jenis pemisah antar elemen anak, tanda koma "," dan tanda bar vertikal "|". Tanda koma, bermakna dalam urutan isi elemen anak, yang dapat di baca elemen anak A diikuti oleh elemen anak B. Tanda vertikal bar, bermakna pilihan yaitu elemen induk hanya boleh memiliki salah satu dari dua elemen anak saja.

XML menyediakan beberapa karakter khusus (*marker*) yang mempunyai makna berapa kali elemen anak dapat muncul dalam dokumen XML, karakter-karakter khusus (marker) tersebut dijelaskan dalam tabel 2.3.

Tabel 2.3 Marker DTD

Marker	Makna
?	Elemen anak dapat muncul 0 atau 1 kali
+	Elemen anak dapat muncul 1 kali atau lebih
*	Elemen anak dapat muncul 0 kali atau lebih

Berikut contoh penggunaan *marker* :

```
<!ELEMENT book (author+, year)>
```

2.2.2.2 ATTLIST

Setiap elemen dapat memiliki satu set atribut yang terkait dengannya. Atribut-atribut untuk elemen tersebut dinyatakan dengan menggunakan tag <!ATTLIST>. berikut ini syntax pendeklarasian ATTLIST :

```
<!ATTLIST ElementName AttributeDefinition>
```

ElementName adalah nama elemen untuk atribut-atribut yang dimiliki. *AttributeDefinition* terdiri dari komponen sebagai berikut:

```
AttributeName AttributeType DefaultDeclaration
```

AttributeName adalah nama atribut. *AttributeType* mengacu pada tipe data atribut. *DefaultDeclaration* berisi deklarasi bagian standar dari definisi atribut. Atribut DTD XML dapat memiliki beberapa tipe. Tabel 2.4 menjelaskan tiap tipe atribut yang ada [4]:

Tabel 2.4 Tipe-tipe Atribut

Tipe atribut	Deskripsi dan contoh
CDATA	Mengijinkan nilai atribut berisi (hampir) sembarang data karakter.
	`<!ATTLIST body bgcolor CDATA #REQUIRED>`
Enumerated	Menggunakan daftar nilai yang diperbolehkan.
	`<!ATTLIST font color (red | blue | black) #REQUIRED>`
ENTITY dan ENTITIES	Nama entiti-entiti biner eksternal dikaitkan dengan elemen ini.
ID	Secara unik mengidentifikasi kejadian dari sebuah elemen dalam dokumen yang ditetapkan.
	`<!ATTLIST a linkid ID #REQUIRED>`
IDREF dan IDREFS	Menunjuk elemen-elemen dengan sebuah nilai atribut ID yang ditentukan
	`<!ATTLIST ul headlink IDREF #IMPLIED>`

Tabel 2.4 Tipe-tipe Atribut (lanjutan)

Tipe atribut	Deskripsi dan contoh
NMTOKEN dan NMTOKENS	Membatasi nilai-nilai atribut untuk jenis tertentu dari data karakter
	`<!ATTLIST body background NMTOKEN "blue" foreground NMTOKENS "green, yellow">`

2.3 Menyimpan dan Mencari Dokumen XML

Sistem manajemen basis data seperti RDBMS sangat efisien dalam menangani data berjumlah banyak dan RDBMS menyediakan fasilitas untuk menjaga integritas, konsistensi, dan ketersediaan [12]. Ada tiga pendekatan untuk menyimpan data dalam format XML di dalam sistem basis data, yaitu [12] :

a. Menyimpan dokumen XML sebagai dokumen terstruktur.
b. Menyimpan dokumen XML sebagai objek DOM tree.
c. Menyimpan dokumen XML sebagai himpunan tabel relasional.

Pendekatan pertama digunakan untuk menyimpan dan mengambil dokumen terstruktur dengan menggunakan basis data asli SGML/XML. Misalnya *OpenText* (*LiveLink*) adalah mesin pencarian *full-text*. Keuntungan menggunakan basis data asli adalah tidak harus merancang pemetaan antara dokumen dan tabel. Dokumen XML dapat disimpan begitu saja dalam basis data dan diambil dengan XPath maupun XQuery.

Pendekatan kedua dapat direalisasikan dengan menggunakan basis data berorientasi objek (OODB). Pada kerangka kerja OODB, sebuah objek data disimpan sebagai objek yang menetap dan sebuah aplikasi dapat mengalamatkan objek melalui pointer. Pada OODB berbasis XML, sebuah dokumen XML dapat dinyatakan sebagai objek DOM *tree* dan disimpan dalam tempat penyimpanan

yang bersifat menetap. Salah satu implementasi pendekatan kedua adalah *eXcelon*. Produk berdasarkan pada OODB umum yang disebut *ObjectStore*. Software ObjectStore menyimpan kumpulan objek DOM dan menyediakan fungsi pencarian dengan menggunakan XPath. Keuntungan utama pendekatan kedua adalah sama dengan basis data asli.

Pada pendekatan ketiga, sebuah dokumen XML disimpan dalam sebuah RDBMS. Pendekatan dapat mengatur sekumpulan tabel relasional dan skemanya, yang merupakan salah satu karakteristik penting XML. Oleh karena itu, kita tidak mudah mengubah sebuah dokumen XML menjadi satu atau lebih tabel.

Pendekatan ketiga lebih banyak digunakan karena dua alasan. Pertama, sebagian besar aplikasi yang ada, menyimpan data dalam RDBMS. Untuk membuat aplikasi Web berbasis XML yang terintegrasi dengan sumber daya RDBMS. Kedua, sebagian besar RDBMS komersil, seperti Oracle dan DB2, mampu menangani data dalam jumlah besar dan jumlah akses yang besar pula. Produk-produk RDBMS komersil menyediakan berbagai macam kemampuan manajemen, termasuk backup data dan recovery. Dengan alasan inilah mengapa dibuat aplikasi yang dapat mentransformasikan dokumen XML menjadi model basis data relasional.

Terdapat empat cara untuk mengambil dokumen XML yang tersimpan, yaitu :

a. Menggunakan bahasa *query* yang dibuat khusus untuk hal ini seperti OpenText.
b. Menggunakan XPath. XPath dapat digunakan sebagai bahasa *query* karena bagian dokumen XML dapat dialamatkan dengan menggunakan XPath.

Expresi XPath juga dapat diubah menjadi SQL untuk mencari sebuah basis data di mana dokumen XML dipecah menjadi tabel-tabel dengan menggunakan JDBC (*Java Database Conectivity*).

c. Menggunakan XQuery, yang merupakan standar W3C yang masih dalam pengembangan. XQuery adalah bahasa query standar untuk dokumen XML. XQuery dapat menentukan format keluaran secara flexibel.

d. Menggunakan Structured Query Language (SQL). SQL adalah bahasa umum untuk mengakses RDBMS. Jika sebuah dokumen XML diubah menjadi data untuk disimpan dalam sebuah tabel sebagai nilai kolom atau jika dokumen XML dihasilkan data yang tersimpan dalam basis data, maka dokumen XML dapat diakses menggunakan SQL.

2.4 Transformasi XML

Penyimpanan model data XML yang bersifat hirarki ke dalam model data relasional yang cenderung bersifat flat dapat dipecahkan dengan beberapa pemetaan (*mapping*) sebagai berikut [7]:

a. *Schema mapping* : dengan algoritma ODTDMap, DTD dari XML digunakan sebagai *input*, dan pemetaan DTD tersebut menjadi sebuah skema basis data dan σ-*mapping* yang memberikan setiap elemen / atribut dalam DTD suatu relasi di mana elemen / atribut tersebut akan disimpan.

b. *Data mapping* : membagi-bagi dokumen XML ke tupel / record relasional dan memasukkan ke dalam skema basis data relasional yang dihasilkan pada tahap sebelumnya. Terdapat dua algoritma, yaitu

OXInsert dan SDM (*SAX-based Data Mapping*). Algoritma OXInsert berbasis DOM parser, sedangkan SDM berbasis pada SAX parser. Dalam pengerjaan tugas akhir ini, penulis hanya menggunakan algoritma SDM.

c. *Query mapping* : menerjemahkan sebuah *query* XML menjadi *query* relasional yang setara (yaitu pernyataan SQL atau ekspresi aljabar relasional), mengeksekusi *query* tersebut terhadap database dan mengembalikan hasil *query* kepada user. Jika hasil *query* adalah dikembalikan sebagai dokumen XML, maka algoritma rekonstruksi diperlukan untuk merekonstruksi XML *subtrees* berakar pada node yang sesuai. Dalam tugas akhir ini tidak dibahas *query mapping* ini.

2.4.1 Schema Mapping

Pada tahap ini digunakan algoritma pemetaan skema *ODTDMap* yang menghasilkan skema dari DTD XML. Sebenarnya terdapat beberapa pendekatan, salah satu pendekatan yakni dengan memetakan setiap elemen DTD ke tabel yang berbeda. Kelemahan dari pendekatan ini adalah memungkinkan menghasilkan terlalu banyak tabel. Pendekatan yang lain adalah dengan memetakan semua elemen DTD ke tabel tunggal. Kelemahan dari pendekatan ini akan menghasilkan tabel yang besar. Sedangkan pendekatan yang lebih baik dengan menggunakan algoritma *ODTDMap*, yakni dengan memetakan *child* dan *parent* pada tabel yang sama ketika *child* muncul maksimal sekali dibawah *parent*, operasi ini disebut dengan operasi *inlining* [7]. Dengan menggunakan pendekatan ini, akan mengurangi jumlah tabel pada skema basis data yang dihasilkan [7].

Algoritma ODTDMap terdiri dari tiga langkah utama [7]:

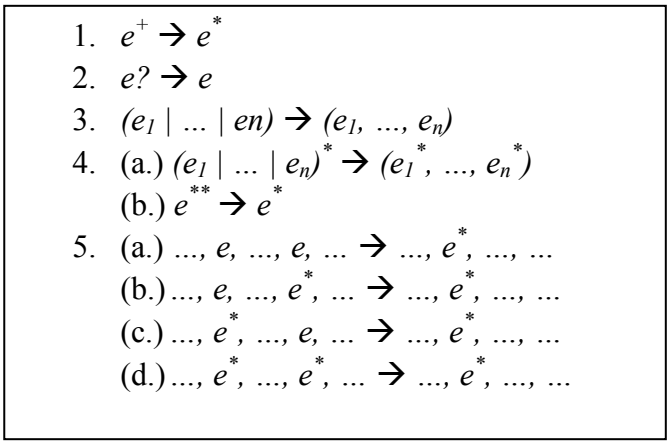
a. *Simplifying DTD* : langkah ini akan menyederhanakan prosedur pemetaan yang komplek dari DTD.
b. *Creating and inlining DTD graph* : dibuat DTD graph dari hasil *simplifying* secara *inlining*.
c. *Generating database schema and σ-mapping* : menghasilkan skema dan σ-mapping berdasarkan *DTD graph* yang telah *inline* (hasil proses inlining).

2.4.1.1 Simplifying DTD

Operator yang terdapat pada DTD dapat diklasifikasikan menjadi dua kelompok berdasarkan hubungan antar elemen parent dan elemen child :

a. Operator yang menggambarkan hubungan one-to-one {‘?’,’,’}
b. Operator yang menggambarkan hubungan one-to-many {‘+’,’*’}

Untuk operator {‘|’}, semisal : `<!ELEMENT a (b|c)>` berarti *a* dapat memiliki unsur elemn *b* atau *c* tetapi tidak memiliki keduanya, dengan demikian dapat dibuat kolom *b* dan *c* dalam satu tabel sesuai dengan elemen *a*, dengan catatan apabila *a*, *b* mengandung elemen, maka nilai kolom *c* adalah null(kosong) dan juga sebaliknya. Dengan pengelompokan tersebut sudah cukup untuk menghasilkan skema relasional pada DTD yang diberikan. Berikut ini aturan *simplifying* DTD pada gambar 2.1 [7]



Gambar 2.1 Aturan *Simplifying DTD*

Dengan menggunakan aturan *Simplifying DTD* di atas, dapat mentransformasikan expresi berikut :

```
<!ELEMENT a ((b+, c*, d?)?, (e?, f, g*, h?)+)?)>
```

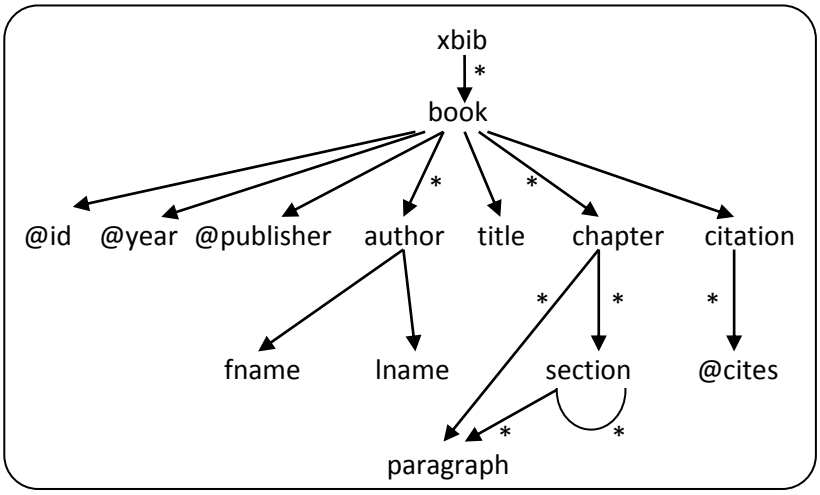
menjadi : `<!ELEMENT a (b*, c*, d, e, f, g*, h*)>`

2.4.1.2 Creating and inlining DTD graph

Dalam langkah ini, dibuat *DTD graph* dari hasil *simplifying DTD* dan dilakukan operasi *inlining* pada *DTD graph*. Struktur dari DTD D dapat diwakili oleh grafik G, G= (V,E), dimana V adalah himpunan *vertex/node* (titik) dan E adalah himpunan dari *edge* (garis). *Node* mewakili elemen dan atribut DTD, dan *edge* mewakili relasi *parent-child*. Setiap *node* diberi label dengan nama elemen terkait atau tipe atribut. Sebuah *edge* diberi label '*' jika *edge* tersebut dapat muncul lebih dari sekali di bawah *parent* dalam dokumen XML. Sebagai contoh, DTD hasil *simplifying* pada gambar 2.2 menghasilkan *DTD graph* seperti pada gambar 2.3.

```
<!ELEMENT xbib (book*)>
<ELEMENT book (title, author+, chapter+,
ciation?)>
<!ATTLIST book id ID #REQUIRED>
<!ATTLIST book year CDATA #REQUIRED>
<!ATTLIST book publisher CDATA #IMPLIED>
<!ELEMENT author (fname, lname)>
<!ELEMENT chapter (paragraph+ | section+)>
<!ELEMENT citation EMPTY>
<!ATTLIST citation cites IDREFS #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT fname (#PCDATA)>
<!ELEMENT lname (#PCDATA)>
<!ELEMENT paragraph (#PCDATA)>
```

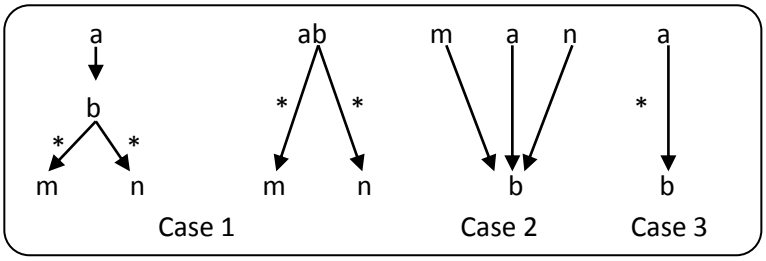
Gambar 2.2 DTD hasil *simplifying*, xbib.dtd



Gambar 2.3 *DTD graph* xbib.dtd

Node yang berisi atribut, didahului dengan tanda "@" dalam *DTD graph*. Untuk atribut bertipe IDREFS atau NMTOKENS, *edge* diberi label dengan '*' dalam *DTD graph*.

Setelah *DTD graph* terbentuk, proses selanjutnya membuat *inlined DTD graph* dengan prosedur *inlining*. Alasannya karena elemen-elemen yang telah *inline* akhirnya akan menghasilkan hubungan tunggal. Berikut aturan transformasi *DTD graph* menjadi inlined *DTD graph* pada gambar 2.4 [7].

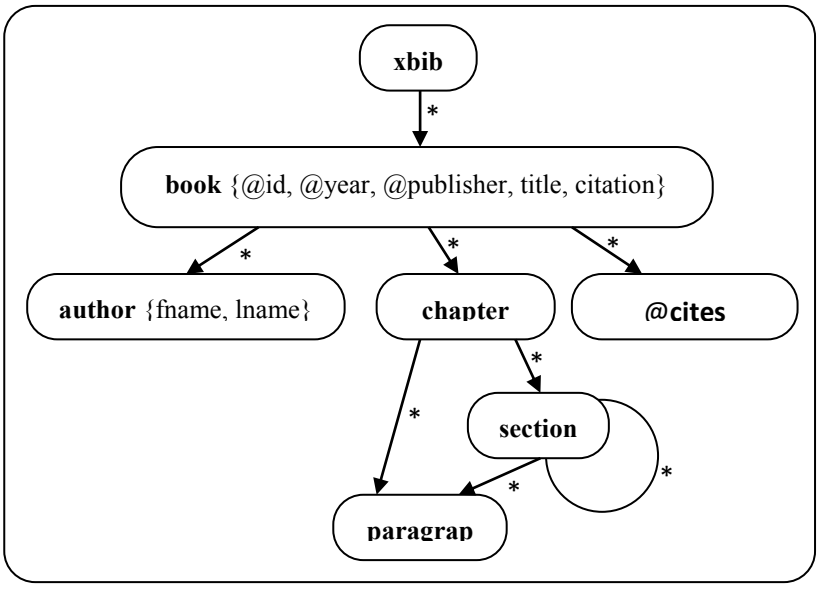


Gambar 2.4 Aturan *inlined DTD graph*

Pada case 1, node *a* berhubungan dengan node *b* oleh garis normal (garis tanpa label '*') dan tak ada garis lain yang menuju *b*. Dalam case ini, *a* dapat menggabungkan node *b* dengan menjaga relasi *parent-child* antara *b* dengan anak-anaknya.

Pada case 2, node *a* berhubungan dengan node *b* oleh garis normal dan terdapat garis lain yang menuju *b*. dalam case ini node *a* dan *b* tak dapat digabungkan karena *b* mempunyai banyak *parent*.

Pada case 3, node *a* berhubungan dengan node b oleh *star edge* (garis dengan label '*'). Dalam case ini node *b* tak dapat digabungkan dengan node *a*.



Gambar 2.5 *Inlined DTD graph*, xbib.dtd

Dengan mengikuti aturan seperti pada gambar 2.4, *DTD graph* xbib.dtd, dari gambar 2.3 dilakukan proses *inlining* menjadi *inlined DTD graph* seperti pada gambar 2.5.

2.4.1.3 Generating database schema and σ-mapping

Langkah akhir dari *schema mapping* adalah menghasilkan skema basis data relasional berdasarkan *inlined DTD graph* yang akan digunakan untuk langkah selanjutnya, yaitu *data mapping*.

Pada dasarnya, skema basis data yang akan dihasilkan mengaitkan setiap elemen *e* dengan ID yang *unique*. Diberikan juga *f.ID* yang *unique* untuk setiap tipe elemen *f* pada kelompok node *inline e*. Alasan di balik diberikannya ID atau *f*.ID untuk setiap elemen adalah untuk dapat menyimpan urutan elemen XML dalam tabel relasional

Atribut *parentID* dikenakan pada setiap elemen yang tidak *inline* untuk menjaga struktur relasi *parent-child* pada DTD graph untuk menangani *query* XML rekursif, tiap elemen *e* diberi atribut *endID* untuk menyimpan ID maksimum *child* dari *e*. atribut *f.endID* juga diberikan pada tiap tipe elemen *f* untuk tujuan yang sama. Berikut ini skema basis data yang dihasilkan dari *inlined DTD graph* pada gambar 2.5.

xbib (ID, endID)
book (ID, parentD, endID, book.id, book.year, book.publisher, title.ID, title.endID, title, citation.ID, citation.endID)
author (ID, parentID, endID, fname.ID, fname.endID, fname, lname.ID, lname.endID, lname)
chapter (ID, parentID, endID)
section (ID, parentID, endID, parentType)
paragraph (ID, parentID, endID, parentType, paragraph)
cites (ID, parentID, cites)

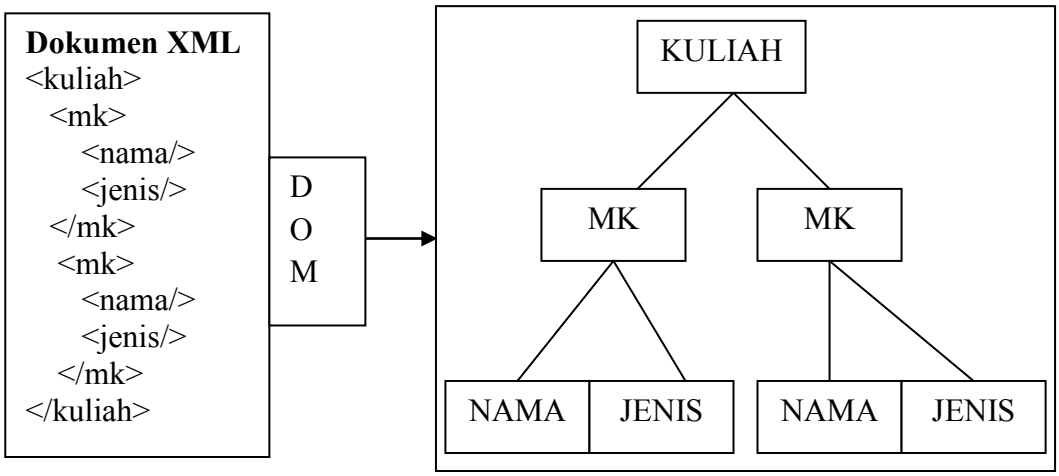
Gambar 2.6 Skema basis data relasional xbib.dtd

2.4.2 Data mapping

Proses *data mapping* membagi-bagi dokumen XML ke tupel / record relasional dan memasukkan ke dalam skema basis data relasional yang dihasilkan pada tahap sebelumnya. Terdapat dua algoritma *data mapping* yang akan dibahas, yakni DOM-based *OXInsert* (DOM parser) dan SAX-based *SDM* (metode *parsing* SAX).

2.4.2.1 DOM-based *OXinsert*

DOM merupakan singkatan dari dari *Document Object Model*. DOM *Parser* menterjemahkan dokumen XML dan menempatkan elemen-elemen yang ditemuinya ketika memproses dokumen ke dalam struktur pohon seperti pada Gambar 2.7 berikut ini.



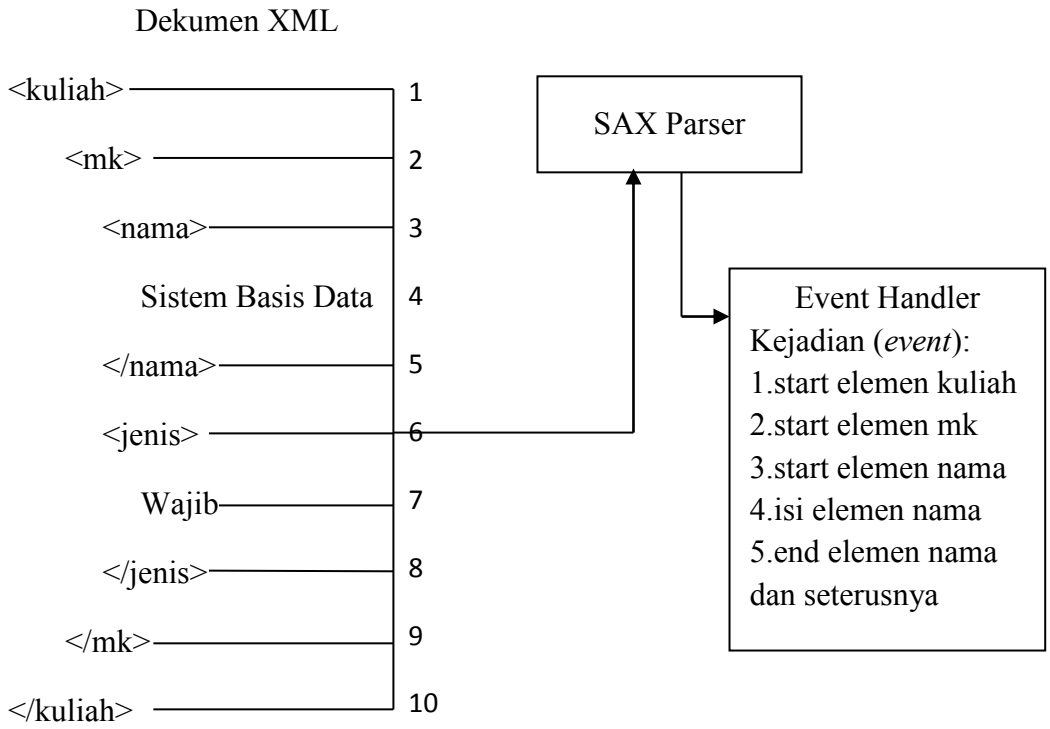
Gambar 2.7 Cara kerja DOM

Pada gambar 2.7 terdapat dokumen XML, elemen *kuliah* memiliki 2 *child* elemen *mk*, pada tiap elemen *mk* memiliki 2 *child* elemen *nama* dan *jenis*. Elemen-elemen tersebut disusun dalam struktur pohon berdasarkan *parent-child*. Informasi mengenai struktur dokumen ini kemudian disimpan ke dalam memori.

Informasi inilah yang digunakan program aplikasi untuk mengakses dokumen XML [2].

2.4.2.2 SAX-based *SDM*

SAX adalah singkatan dari *Simple API for XML*. API sendiri adalah singkatan dari *Application Program Interface*. Seperti namanya, dibanding DOM *Parser*, isi program SAX *Parser* lebih sederhana. SAX *parser* bekerja berdasarkan apa yang disebut *event-based*. Berikut ini gambaran cara kerja SAX *parser* pada Gambar 2.8



Gambar 2.8 Cara kerja SAX

SAX *parser* berjalan menjelajahi dokumen dari awal. Begitu SAX *parser* menemukan deklarasi sebuah elemen, ia akan mencatatnya dan menyimpannya dalam suatu *event-handler*. *Event-handler* inilah yang menyediakan akses ke isi dokumen XML. Cara kerja SAX *parser* diilustrasikan dalam gambar 2.8.

Dari dua jenis metode yang dibahas di atas, tidak ada yang dapat dikatakan lebih unggul dari yang lain. SAX dan DOM dapat dimanfaatkan secara optimal tergantung pada sifat dari dokumen yang hendak di-*parse*. Bila dokumen XMLnya besar, menggunakan DOM akan banyak memakan memori untuk merepresentasikan dokumen dalam struktur pohon. Maka, penggunaan SAX akan lebih menguntungkan dari segi penghematan *resource*. Sebaliknya, untuk dokumen XML berskala kecil, penggunaan DOM akan lebih menguntungkan karena DOM menyediakan struktur yang "siap pakai", sehingga memudahkan program aplikasi dalam mengakses dokumen. Sedang bila kita menggunakan SAX, seorang pemrogram harus bekerja lebih keras dalam mengakses dokumen XML, karena ia harus mendefinisikan sendiri struktur dokumen XMLnya. Sehingga, program aplikasi yang dibuatnya lebih kompleks, akibatnya membebani prosesor yang mengeksekusinya [2].

SAX lebih tepat digunakan untuk menangani dokumen XML yang dinamis, sedang DOM lebih tepat untuk menangani dokumen yang sifatnya statis [2]. Dalam tugas akhir metode parsing SAX yang digunakan karena keunggulannya menangani dokumen XML yang besar.

2.5 Hypertext Preprocessor

Hypertext Preprocessor (PHP) adalah salah satu bahasa script yang berjalan dalam sebuah web server dan berfungsi sebagai pengolah data pada sebuah server [14]. Dengan menggunakan script PHP, sebuah website akan lebih interaktif dan dinamis. Data yang dikirim oleh pengguna website atau komputer *client* akan diolah dan disimpan pada database web server dan dapat ditampilkan kembali

apabila diakses. Untuk menjalankan kode-kode *script* PHP ini, file harus diupload kedalam server. *Upload* adalah proses mentransfer data atau file dari computer klien ke dalam web server sedangkan proser mentransfer data dari webserver ke komputer c*lient* disebut *download*.

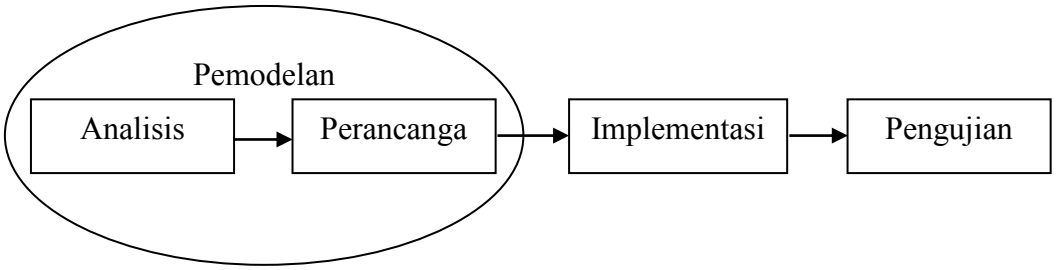
Beberapa keunggulan yang dimiliki script PHP adalah [14]:

a. PHP memilik tingkat akses yang cepat.
b. PHP memiliki *lifecycle* yang cepat sehingga selalu mengikuti perkembangan teknologi internet.
c. PHP memiliki tingkat keamanan yang tinggi.
d. PHP mampu berjalan di beberapa server yang ada, misalnya Apache, Microsoft IIS, PWS, AOLserver, phttpd, fhttpd, dan Xitami.
e. PHP juga mendukung akses ke beberapa database yang sudah ada, baik yang bersifat free maupun komersial. Database itu antara lain MySQL, Microsoft Access, Microsoft SQL server, dan lain sebagainya

2.6 Model Proses Perangkat Lunak

Model proses perangkat lunak merupakan representasi abstrak dari proses perangkat lunak. Setiap model memberikan informasi parsial mengenai proses tersebut. Terdapat berbagai macam model proses perangkat lunak, salah satunya adalah model skuensial linier [10].

Model sekuensial linier mengusulkan sebuah pendekatan kepada pengembang perangkat lunak yang sistematik dan sekuensial yang mulai pada tingkat dan kemajuan sistem pada seluruh analisis, perancangan, implementasi, dan pengujian [10].

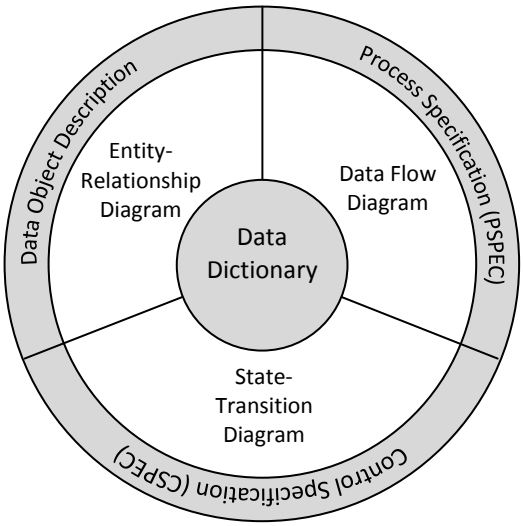


Gambar 2.9 Model Sekuensial Linier

Dalam penyusunan tugas akhir ini, dipilih model sekuensial linier, mulai dari tahap analisis sampai tahap pengujian. Berikut ini penjelasan tiap tahap dari model sekuensial linier :

2.6.1 Analisis

Model analisis harus dapat mencapai tiga sasaran utama : (1) untuk menggambarkan apa yang dibutuhkan oleh pelanggan, (2) untuk membangun dasar bagi pembuatan desain perangkat lunak, (3) untuk membatasi serangkaian persyaratan yang dapat divalidasi begitu perangkat lunak dibangun. Untuk mencapai sasaran tersebut, model analisis yang ditarik selama analisis terstruktur berlangsung ditunjukkan pada gambar 2.10 [10] :



Gambar 2.10 Struktur model analisis

Pada inti model ada kamus data (*data dictionary*) penyimpanan yang berisi deskripsi dari semua objek data yang dikonsumsi atau diproduksi oleh perangkat lunak. Di sini ada tiga diagram yang mengelilingi inti, yaitu : *Entity-relationship diagram*, *Data flow diagram*, dan *State-transition diagram*.

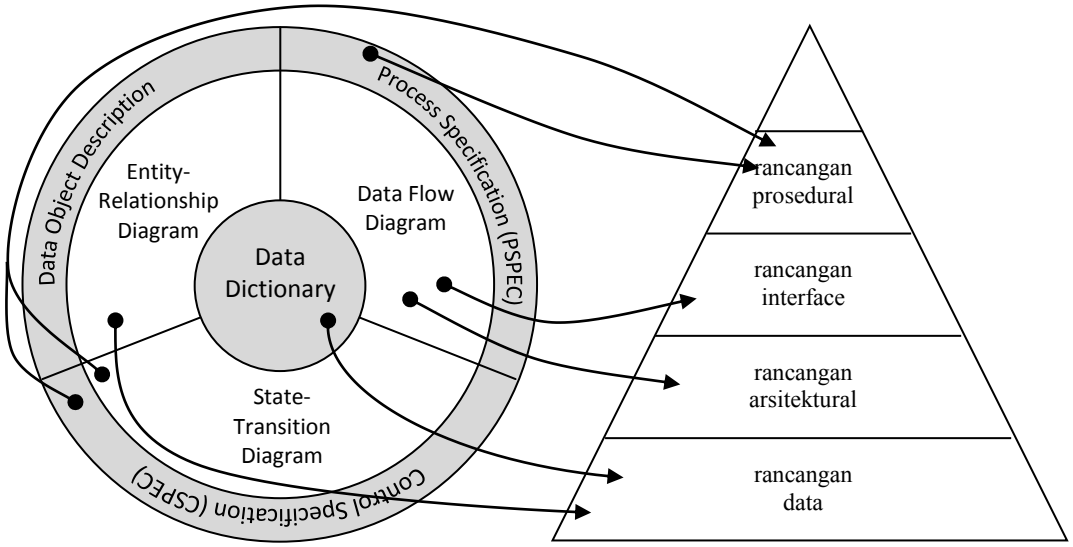
Entity-relationship diagram (ERD) menggambarkan hubungan antara objek data. ERD adalah notasi yang digunakan untuk melakukan aktivitas pemodelan data [10]. Atribut dari masing-masing objek data yang ditulis pada ERD dapat digambarkan dengan menggunakan *deskripsi objek data.*

Data flow diagram (DFD) melayani 2 tujuan : (1) untuk memberikan indikasi mengenai bagaimana data ditransformasikan pada saat data bergerak melalui sistem, dan (2) untuk menggambarkan fungsi-fungsi (dan sub-fungsi) yang mentransformasi aliran data. DFD memberikan informasi tambahan yang digunakan selama analisis domain informasi dan berfungsi sebagai dasar pemodelan fungsi. Deskripsi setiap fungsi yang disajikan pada DFD diisikan dalam sebuah *process specification* (PSPEC).

State-transition diagram (STD) menunjukkan bagaimana sistem bertingkah laku sebagai akibat dari kejadian eksternal. Untuk melakukannya, STD merepresentasikan tingkah laku dari sistem dengan menggambarkan keadaannya dan kejadian yang menyebabkan sistem mengubah keadaan. Informasi tambahan mengenai aspek kontrol dari perangkat lunak diisikan dalam *control specification* (CSPEC).

2.6.2 Perancangan

Masing-masing elemen model analisis memberikan informasi yang diperlukan untuk menciptakan suatu model perancangan. Aliran informasi selama perancangan perangkat lunak ditunjukkan pada Gambar 2.11 [10].



Gambar 2.11 Menerjemahkan model analisis ke dalam suatu desain perangkat lunak

Dalam kegiatan perancangan ini, ditentukan arsitektur sistem secara keseluruhan. Dalam langkah perancangan ini, dihasilkan rancangan data, rancangan arsitektur, rancangan interface, serta rancangan prosedural dari hasil analisis sebelumnya [10].

Desain data mentransformasikan model domain informasi yang dibuat selama analisis ke dalam struktur data yang akan diperlukan untuk mengimplementasi perangkat lunak. Objek dan hubungan data yang ditetapkan dalam ERD dan isi data detail yang digambarkan di dalam kamus data, menjadi aktivitas desain data.

Desain arsitektur menentukan hubungan di antara elemen-elemen struktural utama dari program. Desain arsitektur ini dirancang berdasarkan DFD yang telah dibuat selama proses analisis.

Desain prosedural (perancangan fungsional) mentransformasikan elemen-elemen struktural dari arsitektur program ke dalam suatu deskripsi prosedural dari komponen-komponen perangkat lunak.

Desain interface menggambarkan bagaimana perangkat lunak berkomunikasi dalam dirinya sendiri, dengan sistem yang berinteroperasi dengannya, dan dengan manusia yang menggunakannya.

2.6.3 Implementasi dan pengujian

Pada tahap implementasi, rancangan yang telah dibuat pada tahap sebelumnya diterapkan. Setelah tahap implementasi, aplikasi siap untuk diuji. Terdapat dua cara dalam pengujian sistem [10]:

1. Pengujian *black box*, dengan mengetahui fungsi yang ditentukan di mana produk dirancang untuk melakukannya, pengujian dapat dilakukan untuk memperlihatkan bahwa masing-masing fungsi beroperasi sepenuhnya, pada waktu yang sama mencari kesalahan pada setiap fungsi.
2. Pengujian *white box*, dengan mengetahui kerja internal suatu produk, maka pengujian dapat dilakukan untuk memastikan bahwa operasi internal bekerja sesuai dengan spesifikasi dan semua komponen internal telah diamati dengan baik..

2.7 *Entity-Relationship Diagram*

Entity-relationalship diagram (ERD) merupakan diagram yang menunjukkan keterhubungan antar objek data. Model data *Entity-Relationship* terdiri dari sekumpulan obyek-obyek, yang disebut dengan entitas dan hubungan yang terjadi diantara obyek-obyek tersebut. Model data ERD terbagi menjadi tiga konsep dasar yaitu himpunan entitas, himpunan relasi, dan atribut [16].

1) Entitas (*entity*) dan himpunan entitas (*entity set*)

   Suatu entitas merupakan suatu obyek dasar atau individu yang mewakili sesuatu yang nyata eksistensinya dan dapat dibedakan dari obyek-obyek yang lain. Suatu entitas mempunyai sekumpulan sifat, dan nilai dari beberapa sifat tersebut adalah unik yang dapat mengidentifikasi entitas tersebut [16].

   Sekumpulan entitas yang mempunyai tipe yang sama (sejenis) dan berada dalam lingkup yang sama membentuk suatu himpunan entitas. Sehingga dapat dikatakan bahwa entitas menunjuk pada individu suatu objek, sedangkan himpunan entitas menunjuk pada rumpun dari individu tersebut [16].

2) Atribut

   Suatu entitas memiliki atribut. Atribut merupakan sifat-sifat atau properti yang dimiliki oleh entitas. Atribut inilah yang membedakan antara entitas yang satu dengan entitas yang lain [16].

3) Relasi (*relationship*) dan himpunan relasi (*relationship set*)

   Relasi menunjukkan adanya hubungan di antara sejumlah entitas yang berasal dari sejumlah himpunan entitas yang berbeda sedangkan himpunan relasi yaitu kumpulan semua relasi di antara entitas-entitas yang terdapat pada himpunan entitas [16].

4) Kardinalitas

Kardinalitas merupakan jumlah maksimum entitas dimana entitas tersebut dapat berelasi dengan entitas pada himpunan entitas yang lain. Terdapat empat macam kardinalitas relasi yang terjadi antara himpunan entitas A dan himpunan entitas B, yaitu [16] :

a. Satu ke satu (*One to one*)

Suatu entitas di dalam himpunan entitas A dihubungkan dengan paling banyak satu entitas di dalam himpunan entitas B, dan entitas di dalam himpunan entitas B dihubungkan dengan paling banyak satu entitas dalam himpunan entitas A.

b. Satu ke banyak (*One to many*)

Suatu entitas di dalam himpunan entitas A dihubungkan dengan lebih dari satu entitas di dalam himpunan entitas B, dan entitas di dalam himpunan entitas B hanya dapat dihubungkan dengan paling banyak satu entitas dalam himpunan entitas A.

c. Banyak ke satu (*Many to one*)

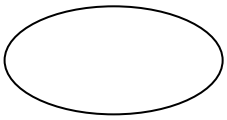
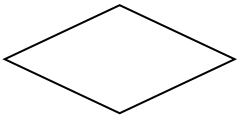
Suatu entitas di dalam himpunan entitas A dihubungkan dengan paling banyak satu entitas di dalam himpunan entitas B, dan entitas di dalam himpunan entitas B dapat dihubungkan dengan lebih dari satu entitas dalam himpunan entitas A.

d. Banyak ke banyak (*Many to many*)

Suatu entitas di dalam himpunan entitas A dapat dihubungkan dengan lebih dari satu entitas di dalam himpunan entitas B, dan entitas di dalam himpunan entitas B dapat dihubungkan dengan lebih dari satu entitas dalam himpunan entitas A.

Simbol-simbol yang digunakan dalam ERD dapat dilihat pada tabel 2.5 berikut [16] :

Tabel 2.5 Notasi Diagram E-R

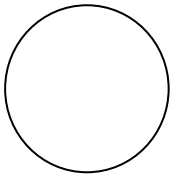
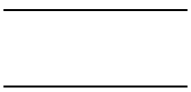
Simbol	Keterangan
	**Himpunan entitas** Menggambarkan himpunan entitas yang diidentifikasikan dalam lingkungan pemakai.
	**Atribut** Menggambarkan himpunan atribut yang merupakan suatu elemen-elemen dari entitas.
	**Relasi** Menggambarkan himpunan *relationship* antara entitas yang satu dengan yang lain.
	Garis Menggambarkan hubungan atribut ke entitas dan himpunan entitas ke himpunan *relationship*.

2.8 *Data Flow Diagram*

Data Flow Diagram (DFD) adalah alat pembuatan model yang memungkinkan profesional sistem untuk menggambarkan sistem sebagai suatu jaringan proses fungsional yang dihubungkan satu sama lain dengan alur data, baik secara manual maupun komputerisasi [9].

Simbol-simbol yang digunakan dalam DFD dapat dilihat pada tabel 2.6 berikut [9] :

Tabel 2.6 Notasi DFD Yordan dan DeMarco

Notasi	Keterangan
	**Terminator/Entitas** Mewakili entitas eksternal yang berkomunikasi dengan sistem yang dikembangkan. Biasanya terminator dikenal dengan nama entitas luar (*external entity).*
	**Proses** Menggambarkan bagian dari sistem yang mentransformasikan dari *input* menjadi *output.*
	***Data Store*** Komponen ini digunakan untuk membuat model sekumpulan paket data dan diberi nama dengan kata benda jamak. *Data store* juga berkaitan dengan penyimpanan seperti *file* atau *database.*
	***Data Flow*/Alur Data** Menunjukkan arah menuju ke dan keluar dari suatu proses. Alur data ini digunakan untuk menerangkan perpindahan data atau paket data/informasi dari satu bagian sistem ke bagian lainnya.

2.9 Kamus Data

Kamus data merupakan sebuah daftar yang teorganisasi dari elemen data yang berhubungan dengan sistem, dengan definisi yang tegar dan teliti sehingga

pemakai dan analis sistem akan memiliki pemahaman yang umum mengenai *input*, *output*, komponen penyimpanan, dan bahkan kalkulasi inter-mediate [10].

Adapun notasi dalam penulisan kamus data dapat dilihat pada tabel 2.7 [10].

Tabel 2.7 Notasi Kamus data

Konstruk data	Notasi	Arti
	=	Disusun atas
Berurutan	+	dan
Pilihan	[|]	Baik ini - atau
Pengulangan	$\{\ \}^n$	Pengulangan ke-*n* dari
	( )	Data opsional
	* *	Komentar tidak dibatasi