**DESAIN MODEL PERANGKAT LUNAK BERBASIS DATABASE MENGGUNAKAN METODE MODEL VIEW CONTROLLER (MVC)**

D Tri Octafian 1,Izman Herdiansyah2 ,Ahmad Luthfi3

 Software Engineering, Ilmu Komputer, Universitas Bina Darma

email: d.trioctafian@gmail.com 1, izman@yahoo.com 2 , luthfi.ubd@gmail.com3

ABSTRAK

Perangkat lunak memiliki karakteristik yang mengikuti si pembangun perangkat lunak, dilihat dari pola yang menyusun arsitektur perangkat lunak tersebut. Hal ini menyebabkan perangkat lunak akan sulit dikembangkan jika si pengembang mengalami pergantian. Perangkat lunak yang dibentuk menggunakan arsitektur MVC akan membentuk pola yang memiliki aturan atau standarisasi, sehingga mempermudah dalam pengembangan perangkat lunak. Kerangka utama arsitektur MVC dibagi menjadi tiga bagian yaitu Model, View, dan Controller. Setiap bagian memiliki hubungan dan saling berkomunikasi antar bagian, sehingga membentuk perangkat lunak yang secara utuh.

Bagian model dari sebuah arsitektur MVC bersifat indenpenden, sejauh mana sifat indenpenden tersebut dibuktikan dalam ruang lingkup pemodelan perangkat lunak yang menjalankan proses-proses DDL dan DML. Sifat indenpenden model dari arsitektur MVC menjadikan perangkat lunak bersifat reuseability.

**Kata Kunci:** *Perangkat lunak, arsitektur, pola, MVC*

1. **PENDAHULUAN**

Dalam pembangunan perangkat lunak untuk pengolahan data pada sebuah *database*, sering kali para *programmer* menghadapi rutinitas yang tinggi dalam menuliskan perintah-perintah dengan menggunakan bahasa pemrograman tertentu, seperti perintah-perintah untuk pendefenisian data (*Data Defenition Language* atau DDL), dan perintah-perintah untuk memanipulasi data (*Data Manipulation Language* atau DML).

Sekumpulan perintah untuk koneksi ke sebuah *DataBase Management System* (DBMS) termasuk *codes* yang sering digunakan dalam pembangunan perangkat lunak, setiap perangkat lunak akan melakukan pengambilan data untuk diolah atau pengiriman data untuk disimpan, koneksi pasti dilakukan terlebih dahulu. Teknik koneksi ke sebuah DBMS pun banyak pilihan, tergantung teknik koneksi yang mana, yang terbaik untuk digunakan pada perangkat lunak yang sedang dibangun.

Sebuah *model* perangkat lunak yang bersifat *reusability,* dalam pembangunan perangkat lunak sangat diperlukan, untuk meminimalkan penulisan ulang perintah-perintah yang sering digunakan, sehingga menghasilkan perangkat lunak yang optimal dalam LOC (*Line Of Code*). Selain itu juga, memudahkan para *programmer* dalam membangun perangkat lunak, karena dengan *model* yang bersifat *reusability* terdapat *document-document* yang bisa dipakai ulang.

Pada penelitian ini, penulis mencoba membangun sebuah *model* perangkat lunak yang dikhususkan untuk kebutuhan konektifitas ke beberapa DBMS, kebutuhan pendefenisian data (DDL), dan kebutuhan manipulasi data (DML) yang menggunakan arsitektur *Model View Controller* (MVC). Arsitektur MVC diharapkan mampu menghasilkan sebuah perangkat lunak yang memiliki standarisasi. Sehingga dengan adanya standarisasi tersebut, perangkat lunak bisa dibangun atau dikembang oleh siapapun.

MVC adalah salah satu arsitektur dalam pembentukan perangkat lunak, dimana metode ini memisahkan arsitektur perangkat lunak menjadi 3 bagian, yaitu: Data (*Model*), Tampilan (*View*), dan Pemroses (*Controller*). Dengan tujuan untuk mempermudah dan mempercepat dalam pengembangan perangkat lunak. Karena setiap bagian bisa dikerjakan secara terpisah dan oleh orang yang berbeda, dengan begitu pengembangan perangkat lunak bisa saling berjalan secara paralel.

1. **ANALISIS DAN DESAIN MODEL**
	1. **ANALIS MODEL**
		1. **Domain Masalah**

Berdasarkan latar belakang, maka domain masalah yang merupakan variable penelitian adalah sebagai berikut:

1. Konektifitas beberapa DBMS (MySQL, Oracle, dan Postgree SQL).
2. Generalisasi struktur DDL dan DML.
3. Arsitektur MVC
	* 1. **Deskripsi Model Perangkat Lunak**

Model perangkat lunak yang akan dibangun adalah sebuah model yang bertugas untuk memanajemen data dari sebuah *DBMS* yang digunakan. Mulai dari proses pemilihan *DBMS* yang akan digunakan, koneksi ke *DBMS* yang dipilih, membuat objek yang dibutuhkan dalam *DBMS* (misal: *Database*, *Table*, *Trigger*, *Procedure*, dan *Function*), menyimpan data ke dalam *DBMS*, mengambil data dari *DBMS*, dan memutuskan koneksi dari *DBMS*. Sehingga bisa disimpulkan, terdapat dua jenis aliran antara *DBMS* yang digunakan dengan model perangkat lunak, yaitu aliran komunikasi antara model dengan *DBMS*, dan aliran data dari model ke *DBMS* atau sebaliknya.



Gambar 1. Aliran data dan komunikasi dalam pemodelan

Adapun permasalahan yang harus diperhatikan dalam membuat model ini adalah sebagai berikut:

1. Bagaimana model mampu melakukan konektifitas ke sebuah DBMS yang akan digunakan, dari beberapa pilihan DBMS yang ada.
2. Bagaimana model mampu melakukan komunikasi dengan DBMS, menggunakan bahasa SQL untuk mengalirkan data dari atau ke DBMS yang digunakan.
3. Bagaimana model mampu dikontrol atau berkomunikasi dengan bagian Controller dan View dalam perangkat lunak, menggunakan konsep message dari asosiasi antar objek.

Permasalahan-permasalahan diatas bisa diselesaikan dengan cara sebagai berikut:

1. Setiap DBMS yang ada memiliki class tersendiri. Dikarenakan metode konektifitas DBMS yang ada, memiliki perbedaan satu sama lain, baik dilihat dari struktur perintah, maupun parameter konektifitas yang dibutuhkan.
2. Sama halnya dengan permasalahan konektifitas diatas, komunikasi antar model dengan DBMS yang digunakan, menggunakan bahasa universal SQL, yang menjadikan permasalahan yaitu: statemen sql dieksekusi atau dijalankan oleh PHP menggunakan library pendukung yang berbeda-beda (misal: MySQL menggunakan library php\_mysql.dll, Oracle menggunakan library php\_oci8.dll, dan Postgree SQL menggunakan library php\_pgsqll.dll). Sehingga struktur perintah yang menjalankan perintah SQL untuk masing-masing DBMS, pasti berbeda-beda. Sehingga dibutuhkan masing-masing class dalam melakukan komunikasi untuk masing-masing DBMS yang ada.
3. Class-class yang melayani kebutuhan masing-masing DBMS, memiliki hubungan (asosiasi) dengan sebuah class utama. Class utama inilah yang nantinya yang melayani class-class dibagian Controller dan View untuk berkomunikasi dengan DBMS (Jembatan – bridge).



Gambar 2. Aliran data dan komunikasi dilihat dari objek dalam pemodelan

Adapun yang menjadi kunci dari model perangkat lunak dan sekaligus untuk menerangkan kandidat-kandidat class dalam analisis perangkat lunak, yaitu:

* 1. MySQL : Objek yang melakukan komunikasi dengan DBMS MySQL.
	2. Oracle : Objek yang melakukan komunikasi dengan DBMS Oracle.
	3. Postgre : Objek yang melakukan komunikasi dengan DBMS Postgree SQL.
	4. DataAccess: Objek yang melakukan komunikasi dengan Objek MySQL, Oracle, dan Postgre.
	5. Model: Objek yang melakukan komunikasi dengan Objek DataAccess dan Objek-objek yang ada di bagian Controller dan View.



Gambar 3. Aliran data dan komunikasi dilihat dari objek yang telah didefenisikan dalam pemodelan.

* + 1. **Model Objek**

Berdasarkan deskripsi model perangkat lunak, didapatkan beberapa kelompok objek, yaitu:

1. Objek-objek yang melakukan komunikasi dengan DBMS, yaitu: objek yang berkomunikasi dengan MySQL (diberi nama objek “MySQL”), objek yang berkomunikasi dengan Oracle (diberi nama objek “Oracle”), Objek yang berkomunikasi dengan Postgree SQL (diberi nama objek “Postgre”).
2. Objek yang melakukan komunikasi dengan objek-objek eksternal (diluar model, yaitu dengan pada Controller, dan View. Diberi nama objek “Model”) dan internal(didalam model, yaitu dengan objek-objek yang berkomunikasi dengan DBMS. Diberi nama objek “DataAccess”).

Tabel 1. Objek pada tahap analisis

|  |  |  |
| --- | --- | --- |
| **No** | **Objek** | **Attribut-Operasi** |
| 1 | MySQL: Objek yang melakukan komunikasi dengan *DBMS MySQL* (mulai dari melakukan koneksi, menjalankan statemen-statemen sql, mengambil nilai, dan memutuskan koneksi) | **con**:boolean //atribut yang berfungsi menyimpan status koneksi ke *MySQL*.**host**:String //atribut yang berfungsi untuk menyimpan lokasi atau nama server database *MySQL*.**user**:String //atribut yang berfungsi untuk menyimpan informasi nama pengakases database *MySQL*.**password**:String //atribut yang berfungsi untuk menyimpan informasi kata sandi si pengakses database *MySQL.***database**:String //atribut yang berfungsi untuk menyimpan informasi nama *database* yang digunakan pada *MySQL.***query**:String //atribut yang berfungsi untuk menyimpan perintah *SQL* yang dieksekusi.**result**:Boolean //atribut yang berfungsi untuk memberikan status hasil eksekusi perintah SQL.**row**:Array //atribut yang berfungsi untuk menampung baris data yang berasal dari table yang diakses.**num\_rows**:Integer //atribut yang berfungsi untuk menampung jumlah baris data dari table yang diakses.**MySQL()** //constructor, melakukan inisialisasi atribut.**connect()** //melakukan koneksi ke MySQL.**execution()** //menjalankan statemen SQL.**getRecord()** //mengambil baris data/ record data dari table.**getTotalRecord()** //mengambil jumlah baris data dari table.**close()** //menutup koneksi ke DBMS. |
| 2 | Oracle: Objek yang melakukan komunikasi dengan DBMS Oracle (mulai dari melakukan koneksi, menjalankan statemen-statemen sql, mengambil nilai, dan memutuskan koneksi) | **con**:boolean //atribut yang berfungsi menyimpan status koneksi ke MySQL.**host**:String //atribut yang berfungsi untuk menyimpan lokasi atau nama server database MySQL.**user**:String //atribut yang berfungsi untuk menyimpan informasi nama pengakases database MySQL.**password**:String //atribut yang berfungsi untuk menyimpan informasi kata sandi si pengakses database MySQL**query**:String //atribut yang berfungsi untuk menyimpan perintah SQL yang dieksekusi.**result**:Boolean //atribut yang berfungsi untuk memberikan status hasil eksekusi perintah SQL.**row**:Array //atribut yang berfungsi untuk menampung baris data yang berasal dari table yang diakses.**num\_rows**:Integer //atribut yang berfungsi untuk menampung jumlah baris data dari table yang diakses.**parsing**:Boolean //atribut yang memberikan status parse dari statemen SQL.**Oracle()** //constructor, melakukan inisialisasi atribut.**connect()** //melakukan koneksi ke Oracle.**parse()** //melakukan penguraian (pemeriksaan) statemen SQL.**execution()** //menjalankan statemen SQL.**getRecord()** //mengambil baris data/ record data dari table.**getTotalRecord()** //mengambil jumlah baris data dari table.**close()** //menutup koneksi ke DBMS. |
| 3 | Postgre: Objek yang melakukan komunikasi dengan DBMS Postgree SQL (mulai dari melakukan koneksi, menjalankan statemen-statemen sql, mengambil nilai, dan memutuskan koneksi) | **con**:boolean //atribut yang berfungsi menyimpan status koneksi ke Postgree SQL.**host**:String //atribut yang berfungsi untuk menyimpan lokasi atau nama server database Postgree SQL.**user**:String //atribut yang berfungsi untuk menyimpan informasi nama pengakases database Postgree SQL.**password**:String //atribut yang berfungsi untuk menyimpan informasi kata sandi si pengakses database Postgree SQL.**database**:String //atribut yang berfungsi untuk menyimpan informasi nama database yang digunakan pada Postgree SQL.**query**:String //atribut yang berfungsi untuk menyimpan perintah SQL yang dieksekusi.**result**:Boolean //atribut yang berfungsi untuk memberikan status hasil eksekusi perintah SQL.**row**:Array //atribut yang berfungsi untuk menampung baris data yang berasal dari table yang diakses.**num\_rows**:Integer //atribut yang berfungsi untuk menampung jumlah baris data dari table yang diakses.**Postgre()** //constructor, melakukan inisialisasi atribut.**connect()** //melakukan koneksi ke Postgree SQL**execution()** //menjalankan statemen SQL.**getRecord()** //mengambil baris data/ record data dari table.**getTotalRecord()** //mengambil jumlah baris data dari table.**close()** //menutup koneksi ke DBMS. |
| 4 | DataAccess: Objek yang melakukan komunikasi internal (komunikasi dengan objek MySQL, Oracle, dan Postgre) | **obj**:Boolean //atribut yang berfungsi menyimpan status komunikasi ke objek MySQL, Oracle, dan Postgre.**host**:String //atribut yang berfungsi untuk menyimpan lokasi atau nama server database dari DBMS yang digunakan.**user**:String //atribut yang berfungsi untuk menyimpan informasi nama pengakases database dari DBMS yang digunakan.**password**:String //atribut yang berfungsi untuk menyimpan informasi kata sandi si pengakses database dari DBMS yang digunakan.**database**:String //atribut yang berfungsi untuk menyimpan informasi nama database dari DBMS yang digunakan.**query**:String //atribut yang berfungsi untuk menyimpan perintah SQL yang dieksekusi.**row**:Array //atribut yang berfungsi untuk menampung baris data yang berasal dari table yang diakses.**num\_rows**:Integer //atribut yang berfungsi untuk menampung jumlah baris data dari table yang diakses.**choice**:Integer //atribut yang berfungsi menampung pilihan objek (MySQL, Oracle, dan Postgre) yang akan digunakan. **DataAccess()** //Melakukan inisialisasi atribut.**access()** //Melakukan pilihan objek mana yang akan digunakan, melakukan koneksi, dan memerintah objek yang dipilih untuk melakukan eksekusi statemen SQL.**execute()** //mengirim message ke metode access().**getRecord()** //mengambil nilai dalam bentuk baris data dari objek (MySQL, Oracle, dan Postgre) yang dipilih.**getTotalRecord()** //mengambil nilai dalam bentuk jumlah baris data dari objek (MySQL, Oracle, dan Postgre) yang dipilih.**close()** //melakukan instruksi ke objek (MySQL, Oracle, dan Postgre) yang dipilih, untuk menutup koneksi ke DBMS. |
| 5 | Model: objek yang menghubungi komunikasi antara objek-objek yang berada pada bagian controller dan view dengan objek-objek yang berada di dalam bagian model. | **DA**:Boolean //atribut yang menampung status komunikasi antara objek model dengan objek-objek dibagian controller (external) dan status komunikasi antara objek model dengan DataAccess.**host**:String //atribut yang berfungsi untuk menyimpan lokasi atau nama server database dari DBMS yang digunakan.**user**:String //atribut yang berfungsi untuk menyimpan informasi nama pengakases database dari DBMS yang digunakan.**password**:String //atribut yang berfungsi untuk menyimpan informasi kata sandi si pengakses database dari DBMS yang digunakan.**database**:String //atribut yang berfungsi untuk menyimpan informasi nama database dari DBMS yang digunakan.**row**:Array //atribut yang berfungsi untuk menampung baris data yang berasal dari table yang diakses.**num\_rows**:Integer //atribut yang berfungsi untuk menampung jumlah baris data dari table yang diakses.**choice**:Integer //atribut yang berfungsi menampung pilihan objek (MySQL, Oracle, dan Postgre) yang akan digunakan. **model()** //Constructor, melakukan inisialisasi atribut**execute()** //melakukan eksekusi statemen SQL**getRecord()** //mengambil baris data (record data) dari table.**getTotalRecord()** //mengambil jumlah baris data (record data) dari table**close()** //menutup koneksi |

* + 1. **Model Dinamis**
			1. **Event Flow Pada Model**
1. Pada objek model
	1. Pertama kali akan menciptakan objek instance dari objek DataAccess (yang dinyatakan dengan atribut instance “DA”) , dengan menyertakan nilai parameter yang dibutuhkan, seperti: nilai host, user, password, database, dan choice (untuk memilih DBMS mana yang akan digunakan).
	2. Lalu objek DataAccess akan melakukan inisialisasi nilai untuk atribut privatenya (seperti: host, user, password, database, dan choice).
2. Pemanggilan metode execute() pada objek model
	1. Jika metode execute() pada objek model dipanggil, dengan disertakan statemen SQL yang akan dijalankan sebagai nilai parameter “query”.
	2. Maka metode execute() pada objek DataAccess juga akan dipanggil oleh metode execute() pada objek model, dengan disertakan statemen SQL yang sama sebagai nilai parameter “query” untuk metode execute() pada objek DataAccess.
	3. Metode execute() pada model DataAccess akan memanggil metode access() yang berada pada objek DataAccess.
	4. Pada metode access() akan dilakukan pemilihan DBMS mana yang akan digunakan, berdasarkan nilai pada atribut “choice” pada objek DataAccess.
	5. Jika pilihan = 1,
	6. Diciptakan objek instance MySQL dari objek MySQL. Dinyatakan dengan atribut “obj” pada objek DataAccess, dengan disertakan nilai-nilai untuk atribut host, user, password, dan database.
	7. Metode access() akan memanggil metode connect() pada objek MySQL. Metode connect() pada objek MySQL akan menjalankan perintah koneksi ke DBMS MySQL berdasarkan nilai atribut host, user, dan password yang diberikan. Lalu melakukan koneksi ke database yang akan digunakan berdasarkan nilai atribut database yang diberikan.
	8. Metode access() akan memanggil metode execute() pada objek MySQL dengan disertakan statemen SQL sebagai nilai parameter metode execute() pada objek MySQL.
	9. Metode execute() pada objek MySQL akan menjalankan statemen SQL tersebut.
	10. Jika pilihan = 2,
	11. Diciptakan objek instance Oracle dari objek Oracle. Dinyatakan dengan atribut “obj” pada objek DataAccess, dengan disertakan nilai-nilai untuk atribut host, user, dan password.
	12. Metode access() akan memanggil metode connect() pada objek Oracle. Metode connect() pada objek Oracle akan menjalankan perintah koneksi ke DBMS Oracle berdasarkan nilai atribut host, user, dan password yang diberikan.
	13. Metode access() akan memanggil metode parse() pada objek Oracle dengan disertakan statemen SQL sebagai nilai parameter metode parse.
	14. Metode parse() pada objek Oracle akan melakukan penguraian statemen SQL, untuk memeriksa kebenarannya statemen SQL tersebut.
	15. Metode access() akan memanggil metode execute() pada objek oracle, dengan disertakan statemen SQL sebagai nilai parameter metode execute() pada objek Oracle.
	16. Metode execute() pada objek oracle akan menjalankan statemen SQL tersebut.
	17. Jika pilihan = 3,
		1. Diciptakan objek instance Postgre dari objek Postgre. Dinyatakan dengan atribut “obj” pada objek DataAccess, dengan disertakan nilai-nilai untuk atribut host, user, password, dan database.
		2. Metode access() akan memanggil metode connect() pada objek Postgre. Metode connect() pada objek Postgre akan menjalankan perintah koneksi ke DBMS Postgre berdasarkan nilai atribut host, user, dan password yang diberikan. Lalu melakukan koneksi ke database yang akan digunakan berdasarkan nilai atribut database yang diberikan.
		3. Metode access() akan memanggil metode execute() pada objek Postgre dengan disertakan statemen SQL sebagai nilai parameter metode execute() pada objek Postgre.
		4. Metode execute() pada objek Postgre akan menjalankan statemen SQL tersebut.
3. Pemanggilan metode getRecord() pada objek model.
4. Memanggil metode getRecord() pada objek model, maka metode getRecord() pada objek DataAccess juga akan dipanggil oleh metode getRecord() pada objek model.
5. Jika pilihan = 1
	* 1. Maka metode getRecord() pada objek DataAccess akan memanggil metode getRecord() pada objek MySQL.
		2. Metode getRecord() pada objek MySQL akan menjalankan hasil eksekusi yang dilakukan metode execute() pada objek yang sama.
		3. Metode getRecord() pada objek MySQL akan memberikan nilai kembalian berupa baris data (record data) dari table.
		4. Metode getRecord() pada objek DataAccess akan meneruskan nilai kembalian tersebut ke metode getRecord() pada objek model.
		5. Metode getRecord() pada objek model akan meneruskan nilai kembalian yang diperoleh dari metode getRecord() pada objek DataAccess.

c. Jika pilihan = 2

1. Maka metode getRecord() pada objek DataAccess akan memanggil metode getRecord() pada objek Oracle.
2. Metode getRecord() pada objek Oracle akan menjalankan hasil eksekusi yang dilakukan metode parse() pada objek yang sama.
3. Metode getRecord() pada objek Oracle akan memberikan nilai kembalian berupa baris data (record data) dari table.
4. Metode getRecord() pada objek DataAccess akan meneruskan nilai kembalian tersebut ke metode getRecord() pada objek model.
5. Metode getRecord() pada objek model akan meneruskan nilai kembalian yang diperoleh dari metode getRecord() pada objek DataAccess.

d. Jika pilihan = 3

1. Maka metode getRecord() pada objek DataAccess akan memanggil metode getRecord() pada objek Postgre.
2. Metode getRecord() pada objek Postgre akan menjalankan hasil eksekusi yang dilakukan metode execute() pada objek yang sama.
3. Metode getRecord() pada objek Postgre akan memberikan nilai kembalian berupa baris data (record data) dari table.
4. Metode getRecord() pada objek DataAccess akan meneruskan nilai kembalian tersebut ke metode getRecord() pada objek model.
5. Metode getRecord() pada objek model akan meneruskan nilai kembalian yang diperoleh dari metode getRecord() pada objek DataAccess.
6. Pemanggilan metode getTotalRecord() pada objek model.
7. Memanggil getTotalRecord() pada objek model akan diteruskan dengan pemanggilan getTotalRecord() pada objek DataAccess oleh metode getTotalRecord() pada objek model.
8. Jika pilihan = 1
9. Maka metode getTotalRecord() pada objek DataAccess akan memanggil metode getTotalRecord() pada objek MySQL
10. Metode getTotalRecord() pada objek MySQL akan menjalankan hasil eksekusi yang dilakukan metode execute() pada objek yang sama.
11. Metode getTotalRecord() pada objek MySQL akan menghasilkan nilai kembalian berupa jumlah baris data (record data) dari table.
12. Nilai ini akan diteruskan ke metode getTotalRecord() pada objek DataAccess.
13. Metode getTotalRecord() pada objek DataAccess akan meneruskan nilai kembalian ke metode getTotalRecord() pada objek model.
14. Metode getTotalRecord() pada objek model akan meneruskan nilai kembalian yang diperoleh dari metode getTotalRecord() pada objek DataAccess.
15. Jika pilihan = 2
16. Maka metode getTotalRecord() pada objek DataAccess akan memanggil metode getTotalRecord() pada objek Oracle.
17. Metode getTotalRecord() pada objek Oracle akan menjalankan hasil eksekusi yang dilakukan metode parse() pada objek yang sama.
18. Metode getTotalRecord() pada objek Oracle akan menghasilkan nilai kembalian berupa jumlah baris data (record data) dari table.
19. Nilai ini akan diteruskan ke metode getTotalRecord() pada objek DataAccess.
20. Metode getTotalRecord() pada objek DataAccess akan meneruskan nilai kembalian ke metode getTotalRecord() pada objek model.
21. Metode getTotalRecord() pada objek model akan meneruskan nilai kembalian yang diperoleh dari metode getTotalRecord() pada objek DataAccess.
22. Jika pilihan = 3
23. Maka metode getTotalRecord() pada objek DataAccess akan memanggil metode getTotalRecord() pada objek Postgre.
24. Metode getTotalRecord() pada objek Postgre akan menjalankan hasil eksekusi yang dilakukan metode execute() pada objek yang sama.
25. Metode getTotalRecord() pada objek Postgre akan menghasilkan nilai kembalian berupa jumlah baris data (record data) dari table.
26. Nilai ini akan diteruskan ke metode getTotalRecord() pada objek DataAccess.
27. Metode getTotalRecord() pada objek DataAccess akan meneruskan nilai kembalian ke metode getTotalRecord() pada objek model.
28. Metode getTotalRecord() pada objek model akan meneruskan nilai kembalian yang diperoleh dari metode getTotalRecord() pada objek DataAccess.
29. Pemanggilan metode close() pada objek model.
30. Memanggil metode close() pada objek model.
31. Metode close() pada objek model akan melakukan pemanggilan metode close() pada objek DataAccess.
32. Jika pilihan = 1
33. Metode close() pada objek DataAccess akan memanggil metode close() pada objek MySQL.
34. Metode close() pada objek MySQL akan melakukan penutupan koneksi ke DBMS MySQL.
35. Jika pilihan = 2
36. Metode close() pada objek DataAccess akan memanggil metode close() pada objek Oracle.
37. Metode close() pada objek Oracle akan melakukan penutupan koneksi ke DBMS Oracle.
38. Jika pilihan = 3
39. Metode close() pada objek DataAccess akan memanggil metode close() pada objek Postgre.
40. Metode close() pada objek Postgre akan melakukan penutupan koneksi ke DBMS Postgre.
	* + 1. **Status Pada Model**

Dalam model objek perangkat lunak untuk pengolahan data pada sebuah DBMS terdiri dari 2 buah status, yaitu status pendefenisian data, dan status manipulasi data. Keterkaitan model perangkat lunak dengan status-status yang ada, dapat dilihat pada tabel 2 berikut:

Tabel 2: Keterkaitan model objek dengan status

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Nama Model Objek** | **Nama Status** | **Status objek yang Terlibat** |
| 1. | CREATEMisal: membuat table. | Pendefenisian data | 1. Membuka koneksi.
2. Membuka database.
3. Menjalankan/ eksekusi statemen SQL.
4. Menutup koneksi
 |
| 2. | ALTERMisal: mengubah struktur table | Pendefenisian data | 1. Membuka koneksi
2. Membuka database.
3. Menjalankan/ eksekusi statemen SQL.
4. Menutup koneksi.
 |
| 3. | DROPMisal: menghapus table | Pendepenisian data | 1. Membuka koneksi
2. Membuka database
3. Menjalankan/ eksekusi statemen SQL
4. Menutup koneksi.
 |
| 4. | SELECTMisal: menampilkan/ mengambil data pada table. | Manipulasi data | 1. Membuka koneksi
2. Membuka database
3. Menjalankan statemen SQL
4. Menampilkan/ mengambil data
5. Menutup koneksi
 |
| 5. | INSERTMisal: menyisipkan/ menambah data pada table | Manipulasi data | 1. Membuka koneksi
2. Membuka database
3. Menjalankan statemen SQL
4. Menutup koneksi
 |
| 6. | DELETEMisal:Menghapus data pada table | Manipulasi data | 1. Membuka koneksi
2. Membuka database
3. Menjalankan statemen SQL
4. Menutup koneksi
 |
| 7. | UPDATEMisal: mengubah baris data pada table | Manipulasi data | 1. Membuka koneksi
2. Membuka database
3. Menjalankan statemen SQL
4. Menutup koneksi
 |

* + 1. **Model Fungsional**

Model perangkat lunak berhubungan dengan entitas luar yaitu: controller dan view. Controller dan view akan mengirimkan data untuk melakukan pemilihan DBMS yang akan digunakan, melakukan koneksi, membuka database, dan menjalankan statemen SQL untuk pendefenisian data atau manipulasi data (choice/ pilihan, host, user, password, database, dan statemen SQL). Respon model akan mengirimkan status koneksi ke DBMS, status koneksi database, status hasil menjalankan statemen SQL, hasil baris data yang diperoleh dari table (record data), dan jumlah baris data dari table.



Gambar 4. Diagram konteks untuk aliran data objek-objek luar dan bagian model

Gambar 5. Diagram level 0 untuk aliran data objek-objek luar dan pada bagian model



Gambar 6. Diagram level 1 untuk proses koneksi pada bagian model



Gambar 7. Diagram level 1 untuk proses “eksekusi query SQL” pada bagian model

Berdasarkan class diagram diatas, bisa kita lihat hubungan antar subsistem yang ada berserta tugas-tugas yang diberikan pada setiap subsistem, berikut table yang menjelaskan tugas-tugas dari subsistem controller dan view:

Tabel 3: Alokasi subsistem dan tugas-tugas

|  |  |  |
| --- | --- | --- |
| **Subsistem** | **Metode** | **Tugas** |
| Controller | Construct() | 1. Inisialisasi objek model
2. Inisialisasi objek view
3. Pemilihan metode yang akan dijalankan
 |
|  | View\_list() | Memanggil metode view\_list pada subsistem view |
|  | View\_input() | Memanggil metode view\_input pada subsistem view |
|  | View\_edit() | Memanggil metode view\_edit() pada subsistem view |
|  | View\_hapus() | 1. Inisialisasi sql statemen hapus yang akan dijalankan.
2. Memanggil metode execute pada subsistem model.
 |
|  | View\_default() | Memanggil metode view\_default pada subsistem view |
|  | Proses\_simpan() | 1. Inisialisasi variabel-variabel yang akan disimpan dari subsistem view
2. Inisialisasi sql statemen insert yang akan dijalankan
3. Memanggil metode execute pada subsistem model
 |
|  | Proses\_edit() | 1. Inisialisasi variabel-variabel yang akan disimpan dari subsistem view
2. Inisialisasi sql statemen update yang akan dijalankan
3. Memanggil metode execute pada subsistem model
 |
| View | Construct() | Inisialisasi objek model  |
|  | Header() | Menampilkan bagian header perangkat lunak. |
|  | Menu() | Menampilkan bagian pilihan menu perangkat lunak. |
|  | Footer() | Menampilkan bagian footer perangkat lunak. |
|  | View\_default() | Menampilkan tampilan default/ awal perangkat lunak |
|  | View\_list() | Menampilkan data dari table |
|  | View\_input() | Menampilkan form inputan data. |
|  | View\_edit() | Menampilkan form edit data |

Berdasarkan class diagram diatas, dapat kita gambarkan kolaborasi yang terjadi antara klien dan server dilihat dari mekanisme control untuk mengakases modul-modul yang ada dalam pemodelan perangkat lunak.



Gambar 8. Komunikasi client/ server untuk menampilkan halaman awal/ default

Manajemen DDL memiliki mekanisme komunikasi yang sama pada setiap struktur perintahnya (misal: CREATE TABLE, ALTER TABLE, DROP TABLE, dll). Secara sederhana, komunikasi dapat berjalan dalam satu arah, yaitu client hanya merequest statemen SQL yang dikirimkan ke server, dan server menjalankan statemen SQL tersebut.



Gambar 9. Komunikasi client/ server untuk memanajemen DDL

Proses menampilkan data ke sisi client atau proses pencarian data yang dilakukan oleh client merupakan contoh manajemen DML, yang menggunakan statemen SQL yang sama dan memiliki perilaku yang sama.



Gambar 10. Komunikasi client/ server untuk manejemen DML (menampilkan data)

Manajemen DML dalam menyimpan data, mengubah data, dan menghapus data memiliki perilaku yang sama dalam mempengaruhi komunikasi antar subsistem-subsitem yang ada. Komunikasi ini secara sederhana mampu berjalan satu arah antara client dan server. Client melakukan request DML untuk memerintahkan server menyimpan atau meghapus data yang dikirimkan.



Gambar 11. Komunikasi client/ server untuk manejemen DML (input, edit, dan hapus data)

Berikut operasi-operasi yang dilakukan model yang dijelaskan dalam algoritma:

**Operasi execute**, adalah operasi yang digunakan untuk menjalankan statemen SQL.

PROCEDURE execute(query:String)

BEGIN

 IF pilihan = 1 THEN

 [Connect to MySQL];

 [Execute query];

 ELSEIF pilihan = 2 THEN

 [Connect to Oracle];

 [Execute query];

 ELSEIF pilihan = 3 THEN

 [Connect to Postgre];

 [Execute query];

 END IF

END;

**Operasi getRecord**, adalah operasi yang digunakan untuk mengambil record data dari table.

PROCEDURE getRecord()

BEGIN

 CALL Execute();

 GET\_DATA();

 IF NOT EOF THEN

 RETURN [RECORD DATA];

 ELSE

 RETURN [FALSE];

 END IF

END;

**Operasi getTotalRecord**, adalah operasi yang digunakan untuk mengambil jumlah record data dari table.

PROCEDURE getTotalRecord()

BEGIN

 CALL Execute();

 COUNT\_DATA();

 RETURN [RECORD DATA];

 RETURN [NUM];

END;

**Operasi close**, adalah operasi yang digunakan untuk menutup koneksi DBMS.

PROCEDURE close()

BEGIN

 CLOSE\_CONNECT();

END;

* 1. **HASIL**
		1. **Konektifitas DBMS**

Pemodelan yang dibuat mampu melayani konektifitas DBMS (MySQL, Oracle, dan Postgre SQL). Dimana pemodelan mampu melakukan migrasi DBMS yang digunakan tanpa mengubah struktur model yang dibangun.

**Koneksi ke DBMS MySQL**

Pada *class* controller, untuk melakukan koneksi ke *DBMS* *MySQL* memerlukan inisialisasi beberapa atribut *class* controller, seperti: hostname, user, password, database, dan choice. Inisialisasi dilakukan dalam metode \_\_construct() pada *class* controller, dengan begitu koneksi akan dilakukan secara automatis ketika *class* controller diciptakan objek instancenya. Berikut function \_\_construct() di dalam *class* controller untuk koneksi ke *DBMS MySQL*:

function \_\_Construct(){

$this->host = "localhost";

 $this->user = "root";

 $this->password = "";

 $this->database = "perpus";

 $this->choice = 1;

 $this->model = new Model($this->host,$this->user,$this->password,$this->database,$this->choice);

 }

Untuk koneksi ke DBMS MySQL, atribut choice diberi nilai 1,

**Koneksi ke DBMS Oracle**

Pada *class* controller, untuk melakukan koneksi ke *DBMS* *Oracle* memerlukan inisialisasi beberapa atribut *class* controller, sama halnya seperti koneksi ke *DBMS MySQL*. Pada tahap implementasi, penulis menggunakan *DBMS Oracle 10G*  versi *Express Edition*. Pada *oracle* versi ini, *schema user* telah meliputi fungsi sebagai database, sehingga nilai atribut database dikosongkan, dan choice diberi nilai 2. Berikut inisialisasi di function \_\_construct() pada class controller, untuk koneksi ke *DBMS Oracle*:

function \_\_Construct(){

$this->host = "localhost/XE";

 $this->user = "system";

 $this->password = "123456";

 $this->database=””;

 $this->choice = 2;

 $this->model = new Model($this->host,$this->user,$this->password,$this->database,$this->choice);

 }

**Koneksi ke DBMS Postgre SQL**

Tahap koneksi ke DBMS Postgre SQL, sama perlakuannya seperti koneksi ke DBMS MySQL, untuk koneksi ke DBMS Postgre SQL atribut choice diberi nilai 3 dan menggunakan port komunikasi 5432. Berikut inisialisasi di function \_\_construct() pada class controller, untuk koneksi ke *DBMS Postgre SQL*:

function \_\_Construct(){

$this->host = "localhost";

 $this->user = "postgres";

 $this->pass = "123456";

 $this->database = "perpus";

 $this->choice = 3;

 $this->model = new Model($this->host, $this->user, $this->pass, $this->database, $this->choice);

}



Gambar 12. Alur skema koneksi ke DBMS pada model

* + 1. **Manajemen DDL dan DML**
			1. **Manajemen DDL**

Prosedure manajemen DDL dalam pemodelan perangkat lunak sebagai berikut:

* + - 1. Melakukan inisialisasi objek controller sebagai objek instance.
			2. Melakukan inisialisasi query DDL.
			3. Memanggil metode execute() pada objek controller instance.
			4. Pada objek controller dilakukan inisialisasi objek model sebagai objek instance.

Komunikasi yang terjadi antar objek controller dan model, dengan melakukan pengiriman message berupa query DDL yang akan dijalankan. Berikut penjelasan dalam bentuk penggalan scripting:

$a = new controller(); //Inisialisasi objek controller

$sql = "CREATE TABLE books( //Inisialisasi query DDL

 kode\_buku CHAR(21) PRIMARY KEY,

 judul\_buku VARCHAR(60) NOT NULL,

 pengarang VARCHAR(30) NOT NULL,

 penerbit VARCHAR(30) NOT NULL,

 thn\_terbit CHAR(4) NOT NULL,

 stok TINYINT NOT NULL)";

$a->execute($sql); //memanggil metode pada objek controller

Pada objek controller, misal menggunakan DBMS MySQL:

function \_\_Construct(){

$this->host = "localhost";

 $this->user = "root";

 $this->password = "";

 $this->database = "perpus";

 $this->choice = 1;

 $this->model = new Model($this->host,$this->user,$this->password,$this->database,$this->choice); //Inisialisasi objek Model untuk melakukan koneksi ke DBMS.

}

function execute($sql){

 $this->model->execute($sql);

}



Gambar 13. Ilustrasi pengiriman message oleh objek Controller ke objek Model

* + - 1. **Manajemen DML**

Dalam sebuah perangkat lunak, implementasi manajemen DML biasanya membutuhkan interface yang digunakan si pemakai untuk melihat hasil proses manajemen DML atau sarana bantu interaktif si pemakai yang dibutuhkan perangkat lunak untuk melakukan manajemen DML, misal: menampilkan data, menghapus data, menambah data, mengubah data, dan pencarian data pada sebuah table atau beberapa table.

**Menampilkan data**

Berikut prosedur menampilkan data pada pemodelan perangkat lunak yang dibangun:

* 1. Inisialisasi objek Model dan View menjadi objek-objek instance.

$this->model = new Model($host, $user, $pass, $database, $choice);

$this->view = new View($this->model);

* 1. Memanggil metode view\_list() pada objek view untuk menampilkan data.

$this->view->view\_list();

* 1. Pada metode view\_list() di dalam objek View, dilakukan inisialisasi query DML untuk menampilkan data, memanggil metode execute pada objek Model untuk menjalankan query DML tersebut, memanggil metode getRecord() pada objek Model. Berikut penggalan script pada metode view\_list():

$sql = "SELECT \* FROM buku";

$this->model->execute($sql);

$no = 0;

while($baris = $this->model->getRecord()){

$no++;

 $this->output .= "<tr>

 <td>".$no."</td>

 <td>".$baris['kode\_buku']."</td>

 <td>".$baris['judul\_buku']."</td>

 <td>".$baris['pengarang']."</td>

 <td>".$baris['penerbit']."</td>

 <tdalign='center'>".$baris['thn\_terbit']."</td>

 <td align='center'>".$baris['stok']."</td>

<td align='center'><a href='".$\_SERVER['PHP\_SELF']."?view=hapus&kode=$baris[kode\_buku]'>Hapus</a> | <a href='".$\_SERVER['PHP\_SELF']."?view=edit&kode=$baris[kode\_buku]'>Edit</td>

 </tr>";

}

Gambar 14 memperlihatkan skema alur manajemen DML, untuk menampilkan data dalam bentuk table.



Gambar 14. Skema alur manajemen DML (menampilkan data)



Gambar 15. Tampilan objek view untuk menampilkan data

**Menghapus data**

Berikut prosedur menghapus data pada pemodelan perangkat lunak yang dibangun:

* 1. Inisialisasi objek Model dan View menjadi objek-objek instance.

$this->model = new Model($host, $user, $pass, $database, $choice);

$this->view = new View($this->model);

* 1. Memanggil metode view\_list() pada objek view untuk menampilkan data.

$this->view->view\_list();

* 1. Mengirimkan data yang akan dihapus dari objek view ke objek controller.

<a href='".$\_SERVER['PHP\_SELF']."?view=hapus&kode=$baris[kode\_buku]'>Hapus</a>

* 1. Pada objek controller, memanggil metode execute() dari objek model, disertakan message berupa query DML untuk menghapus data.

$this->model->execute($sql);

Gambar 16 memperlihatkan skema alur manajemen DML, untuk menghapus data dalam table.



Gambar 16. alur manajemen DML (menghapus data)

**Menambah data**

Berikut prosedur menambah data pada pemodelan perangkat lunak yang dibangun:

* + - 1. Inisialisasi objek Model dan View menjadi objek-objek instance.

$this->model = new Model($host, $user, $pass, $database, $choice);

$this->view = new View($this->model);

* + - 1. Mengirimkan message ke objek controller dari class view, untuk meminta objek controller memanggil metode view\_input() dari class view.

Pengiriman message:

<a href='".$\_SERVER['PHP\_SELF']."?view=input'>Input</a>

Pemanggilan metode view\_input() oleh objek controller:

$this->view->view\_input();

Metode view\_input() pada objek view:

function view\_input(){

$this->Header();

$this->menu();

$this->output .= "<h3>INPUT DATA BUKU</h3>";

$this->output .= "<form name='frmbuku' method='POST' action='".$\_SERVER['PHP\_SELF']."'><table border=0 align='left'>

<tr>

<td>Kode Buku</td>

<td><input type='text' name='txtkode' /></td>

</tr><tr>

<td>Judul Buku</td>

<td><input type='text' name='txtjudul' /></td>

</tr><tr>

<td>Pengarang</td>

<td><input type='text' name='txtpengarang /></td>

</tr><tr>

<td>Penerbit</td>

<td><input type='text' name='txtpenerbit' /></td>

</tr><tr>

<td>Tahun Terbit</td>

<td><input type='text' name='txttahun' /></td>

</tr><tr>

<td>Stok</td>

<td><input type='text' name='txtstok' /></td>

</tr><tr>

<td><input type='hidden' name='txtproses' value='input\_buku'/></td><td>

<button type='submit'><strong>Simpan</strong></button>

<button type='reset'><strong>Reset</strong></button>

</td></tr></table></form>";

$this->footer();

echo $this->output;

}

* + - 1. Mengirimkan message dalam bentuk data-data yang diinputkan dalam komponen form beserta query DML tambah data, dari objek view ke objek controller.
			2. Objek controller mengirimkan message yang diterima ke objek Model, dengan memanggil metode execute() dari objek model.

$this->model->execute($sql);

Gambar 17 memperlihatkan skema alur manajemen DML, untuk menambah data dalam table.

Gambar 17. Skema alur manajemen DML (menambah data)



Gambar 18. Tampilan objek view untuk menampilkan input data

**Mengubah data**

Berikut prosedur mengubah data pada pemodelan perangkat lunak yang dibangun:

1. Inisialisasi objek Model dan View menjadi objek-objek instance.

$this->model = new Model($host, $user, $pass, $database, $choice);

$this->view = new View($this->model);

1. Memanggil metode view\_list() pada objek view untuk menampilkan data.

$this->view->view\_list();

1. Mengirimkan data yang akan diubah dari objek view ke objek controller.

<a href='".$\_SERVER['PHP\_SELF']."?view=edit&kode=$baris[kode\_buku]'>Edit</a>

1. Menampilkan informasi data yang akan diubah ke dalam bentuk form pengubahan data dari objek controller ke objek view.

**Objek controller:**

function view\_edit(){

$this->view->view\_edit();

}

**Objek view:**

function view\_edit(){

$this->Header();

$this->menu();

$this->output .= "<h3>EDIT DATA BUKU</h3>";

$kode = $\_GET['kode'];

$sql = "SELECT \* FROM buku WHERE kode\_buku='$kode'";

$this->model->execute($sql);

$hasil = $this->model->getRecord();

$this->output .= "<form name='frmbuku' method='POST' action='".$\_SERVER['PHP\_SELF']."'><table border=0 align='left'>

<tr><td>Kode Buku</td>

<td><input type='text' name='txtkode' value='".$hasil['kode\_buku']."' size='25' /></td>

</tr><tr>

<td>Judul Buku</td>

<td><input type='text' name='txtjudul' value='".$hasil['judul\_buku']."' size='60' /></td>

</tr><tr>

<td>Pengarang</td>

<td><input type='text' name='txtpengarang' value='".$hasil['pengarang']."' size='30' /></td>

</tr><tr>

<td>Penerbit</td>

<td><input type='text' name='txtpenerbit' value='".$hasil['penerbit']."' size='30' /></td>

</tr><tr>

<td>Tahun Terbit</td>

<td><input type='text' name='txttahun' value='".$hasil['thn\_terbit']."' size='4' /></td>

</tr><tr><td>Stok</td>

<td><input type='text' name='txtstok' value='".$hasil['stok']."' size='5' /></td>

</tr><tr>

<td><input type='hidden' name='txtproses' value='edit\_buku'/></td>

<td><button type='submit'><strong>Simpan</strong></button><button type='reset'><strong>Reset</strong></button>

</td></tr></table></form>";

$this->footer();

echo $this->output;

}

1. Mengirimkan data-data yang diubah dari form pengubahan data di objek view ke objek controller.
2. Mengirimkan data-data yang akan diubah beserta query DML dari objek controller ke objek model.

$kode = $\_POST['txtkode'];

$judul = $\_POST['txtjudul'];

$pengarang = $\_POST['txtpengarang'];

$penerbit = $\_POST['txtpenerbit'];

$tahun = $\_POST['txttahun'];

$stok = $\_POST['txtstok'];

$sql = "UPDATE buku SET judul\_buku='$judul', pengarang='$pengarang', penerbit='$penerbit', thn\_terbit='$tahun', stok='$stok' WHERE kode\_buku='$kode'";

$this->model->execute($sql);

Gambar 19. memperlihatkan skema alur manajemen DML, untuk mengubah data pada table.



Gambar 19. Skema alur manajemen DML (mengubah data)



Gambar 20. Tampilan objek view untuk perubahan data

**Pencarian data**

Berikut prosedur mengubah data pada pemodelan perangkat lunak yang dibangun:

1. Inisialisasi objek Model dan View menjadi objek-objek instance.

$this->model = new Model($host, $user, $pass, $database, $choice);

$this->view = new View($this->model);

1. Memanggil metode view\_search() pada objek view untuk menampilkan data.

$this->view->view\_search();

1. Mengirimkan data yang akan dicari dari objek view ke objek controller
2. Memanggil metode view\_search() pada objek view dari objek controller untuk menampilkan informasi yang dicari.

Gambar 21. memperlihatkan skema alur manajemen DML, untuk mencari data pada table.



Gambar 21. Skema alur manajemen DML (mencari data)



Gambar 22. Tampilan objek view untuk pencarian data

* + 1. **Standarisasi Pemodelan Perangkat Lunak**

Pemodelan perangkat lunak yang dihasilkan dapat dilihat pada gambar 23 berikut:

Gambar 23. Standar pemodelan perangkat lunak

**Objek View:**

{ Metode yang digunakan untuk membentuk/ Inisialisasi tampilan }

**[Procedure/ Function] [Nama([Parameter])]**

[Inisialisasi Tampilan]

**[End] [Procedure/ Function]**

{ Metode yang digunakan untuk memanggil metode-metode penginisialisasi tampilan }

**[Procedure/ Function] [Nama([Parameter])]**

 [Nama Metode Penginisialisasi 1]

 . . .

 [Nama Metode Penginisialisasi n]

**[End] [Procedure/ Function]**

**Objek Controller:**

{ Pemilihan untuk menentukan metode inisialisasi tampilan mana yang akan digunakan }

**[Condition]**

 [Memanggil metode inisialisasi tampilan]

 . . .

**[End Condition]**

{ Pemilihan untuk menentukan metode proses bisnis mana yang akan digunakan }

**[Condition]**

 [Memanggil metode proses bisnis]

 . . .

**[End Condition]**

{ Metode untuk menjalankan proses bisnis perangkat lunak }

**[Procedure/ Function] [Nama([Parameter])]**

 [Proses Bisnis]

**[End] [Procedure/ Function]**

{ Metode untuk memanggil metode pemanggil penginisialisasi tampilan pada objek view }

**[Procedure/ Function] [Nama([Parameter])]**

 [Nama Metode Pada Objek View]

**[End] [Procedure/ Function]**

Pembangunan perangkat lunak dengan menambahkan metode-metode yang dibutuhkan, metode-metode dapat berupa function-function dan procedure-procedure.

Pemodelan yang dihasilkan membentuk pola standar bagi pengembangan perangkat lunak. Dimana objek model selama uji coba tidak mengalami perubahan untuk melakukan manajemen database yang meliputi konektifitas, manajemen DDL dan DML. Begitu juga bagian Controller dan View dihubungkan dengan bagian Model dengan mengikuti aturan dan pola tertentu.

* 1. **KESIMPULAN**

Berdasarkan hasil pembahasan dan analisis yang telah dilakukan serta sesuai dengan maksud dan tujuan penelitian, maka dapat diambil kesimpulan sebagai berikut:

1. Arsitektur MVC mampu memisahkan tiga bagian utama sebuah perangkat lunak, yaitu Model, View, dan Controller. Dimana pembangunan perangkat lunak akan mempengaruhi bagian controller, dan view, yang disesuaikan dengan proses bisnis dan interface perangkat lunak yang akan dibangun. Bagian model relatif tetap dalam melakukan konektifitas, manajemen DDL dan DML untuk berbagai proses bisnis.
2. Arsitektur MVC memiliki pola/ *pattern* dalam membentuk perangkat lunak, sehingga memudahkan dalam pembangunan perangkat lunak. Penambahan, pengurangan dan perubahan modul-modul perangkat lunak dalam proses pembangunan, dilakukan pada bagian controller, dan view. Pembangunan perangkat lunak dapat dilakukan terpisah secara paralel untuk masing-masing bagian.
3. Pola/ *pattern* perangkat lunak yang dibentuk oleh arsitektur MVC, membentuk standarisasi perangkat lunak. Sehingga membantu para pengembangan perangkat lunak dalam penambahan, pengurangan, dan perubahan modul-modul perangkat lunak. Selain itu juga.

**REFERENSI**

1. Afriani, I. (2009), *Metode Penelitian Kualitatif*, Diakses 4 Juni 2011, dari http://www.penalaran-unm.org/index.php/artikel-nalar/penelitian/116-metode-penelitian-kualitatif.html
2. Althammer, E & Pree, W, ‘Design And Implementation Of A MVC-Based Architecture For E-Commerce Applications’, Jurnal e-commerce, h. 1-22.
3. Armansyah, Indra (2005), *Tip dan Trik Oracle*, Elex Media Komputindo, Jakarta.
4. Azis, M Farid (2005), *Object Oriented Programming dengan PHP5*, Elex Media Komputindo, Jakarta
5. Badriyah, T. (2007), *UML Class Diagram*. Diakses 2 Juni 2011, dari http://lecturer.eepis-its.edu/~tessy.
6. Dharwiyanti, S & Wahono, R.S. (2003), *Pengantar Unified Modeling Languae (UML)*, ilmukomputer.
7. Gupta, P & Govil, M.C. (2010), ‘MVC Design Pattern for the multi framework distributed applications using XML, spring and struts framework’, Jurnal international computer science and engineering, Vol. 02, No. 04, h. 1047-1051.
8. Jogiyanto. (2008), *Metodologi Penelitian Sistem Informasi*, Penerbit Andi, Yogyakarta.
9. Kadir, Abdul (2009), *Dasar Perancangan & Implementasi Database Relasional*, Penerbit Andi, Yogyakarta.
10. Kadir, Abdul (2010), *Mudah Mempelajari Database MySQL,* Penerbit Andi, Yogyakarta.
11. Kristanto, Andri (2010), *Kupas Tuntas PHP & MySQL 9 Jam Menguasai PHP dan MySQL Dengan Mudah & Cepat*, Penerbit Cable Book, Klaten
12. Narbuko, C & Achmadi, A. (1997), *Metodologi Penelitian*, Penerbit Bumi Aksara, Jakarta.
13. Nugraha, Bunafit (2005), *Database Relasional Dengan MySQL*, Penerbit Andi, Yogyakarta.
14. Rahman, A. (2007), *Pengantar Rekayasa Perangkat Lunak (RPL)*, Universitas Brawijaya, Malang.
15. Syafi’i, Muhammad (2005), *Panduan Membuat Aplikasi Database dengan PHP5, MySQL, PostgreSQL, Oracle*, Penerbit Andi, Yogyakarta.
16. Thamura, F. (2004), *Pengenalan Arsitektur MVC*, Meruvian Foundation.
17. Yanto. (2009), ‘Penerapan Java Server Faces Untuk Design Pattern Web’, Jurnal Informatika, Vol. 5, No. 1, h. 38-49.
18. Pressman, Roger S. (2002), *Rekayasa Perangkat Lunak Pendekatan Praktisi*, Penerbit Andi, Yogyakarta.