Implementasi *Model View Controller* dan *Object Relational Mapping* pada *Content Management System* Sistem Informasi Keuangan

[1)]**Kristoko Dwi Hartomo**, [2)]**Theophilus Wellem**, [3)]**David Adi Sanjaya**

Fakultas Teknologi Informasi
Universitas Kristen Satya Wacana
Jl. Diponegoro 52-60, Salatiga 50711, Indonesia
Email: [1)]kdh73@yahoo.com, [2)]erman_wellem@yahoo.com, [3)]das_84id@yahoo.com

Abstract

This paper discusses the development of a *Content Management System* for Financial *Information System*. The method used is *Component-Based Development*, which is a technique in *software* engineering that is concerned with the assembly of pre-existing *software* components into larger pieces of *software*. The *framework* used for the development is *Kohana*, which is a PHP 5 *web* application *framework* that uses *Model View Controller* (*MVC*) architectural pattern.It can be concluded that development of the *Content Management System* using MVC and ORM resulting in an application where it is easier to modify either the visual appearance of the application or the underlying *business logic* without affecting the other.

Keywords: Object Relational Mapping (ORM), Model View Controller (MVC), Content Management System (CMS)

1. Pendahuluan

Content Management System (CMS) adalah suatu perangkat lunak yang memungkinkan seseorang untuk menambahkan dan atau memanipulasi isi suatu situs web [1]. Suatu CMS aplikasi keuangan pasti memerlukan perancangan yang baik bagian demi bagian, demikian juga dalam proses pembuatannya agar memudahkan dalam pengembangannya. Sekarang ini muncul sebuah konsep yang dapat memudahkan *programmer* dalam mengembangkan aplikasinya dalam jangka ke depan dengan mengimplementasikan manajemen kode yang baik, yaitu dengan cara memisahkan bagian per bagian kode yang disebut *Model View Controller* (MVC), konsep ini memisahkan suatu aplikasi web dimana kode yang semula ditulis pada satu halaman saja, kemudian dipisahkan menjadi tiga bagian, bagian pertama disebut *model* yang menangani aktivitas pada *database*, bagian kedua disebut *view*, yang menangani presentasi data pada sisi *client*, dan bagian terakhir disebut *controller*, yang menjembatani antara *model* dan *view*. Konsep ini akan dapat mempermudah *programmer* ketika mengerjakan aplikasi untuk bisa fokus pada salah satu bagian demi bagian dan dapat lebih mudah mengembangkan aplikasi ini ke depannya.

CMS aplikasi keuangan merupakan CMS yang juga banyak berhubungan dengan pemrosesan *database*, dan *database* yang diperlukanpun cukup banyak, maka harus pula dirancang cara pemrograman yang baik, agar aplikasi ini dapat dikembangkan lagi nanti. Sekarang ini muncul sebuah konsep baru yang disebut *Object Relational Mapping* (ORM), dimana dengan konsep ini *database* dipetakan ke dalam bentuk obyek, sehingga proses penulisan kode program yang berhubungan dengan *database* akan semakin hemat, dan semakin mudah, hal ini merupakan nilai tambah bagi *programmer* dalam mengembangkan aplikasinya.

Bermunculannya konsep tadi seperti MVC dan ORM dapat memudahkan *programmer* dalam mengerjakan dan juga mengembangkan aplikasinya. Dengan digunakannya konsep-konsep tadi maka diharapkan proses pembuatan, dan pengembangan CMS Sistem Informasi Keuangan ini akan menjadi semakin mudah.

2. Kajian Pustaka

Pada artikel mengenai ORM dibandingkan beberapa *framework* ORM. Dari artikel tersebut, selain didapatkan keunggulan dan kekurangan masing-masing *framework*, juga didapatkan kegunaan ORM untuk kepraktisan pengelolaan *database* melalui pemrograman, karena setelah suatu tabel dipetakan ke dalam kelas, maka *programmer* tidak perlu menuliskan *query-query* yang sama secara berulang-ulang dan itu membuat proses pemrograman menjadi lebih praktis dan efisien [2]. Pada artikel mengenai MVC dijelaskan tentang arsitektur MVC dan bagaimana penerapannya serta keuntungannya. Ternyata dengan menggunakan arsitektur MVC, maka dalam pengembangan suatu aplikasi khususnya aplikasi web menjadi lebih mudah, karena pemisahan *bussiness logic*, *presentation logic*, dan *persistence layer*, sehingga apabila *programmer* ingin mengubah suatu fungsi pada aplikasinya, maka tidak perlu dengan susah mencari fungsi tersebut pada satu halaman yang sudah bercampur aduk dengan fungsi lain di luar kelompoknya, dan juga dengan kode HTML [3]. Dari beberapa artikel dan penelitian tersebut, penulis tertarik untuk menerapkan konsep ORM dan MVC pada suatu aplikasi CMS Sistem Informasi Keuangan, karena suatu CMS yang baik dan banyak diminati pasti memerlukan pengembangan aplikasi menjadi lebih baik lagi, sehingga beberapa konsep tadi, yaitu ORM dan MVC sangatlah cocok untuk diterapkan pada aplikasi ini.

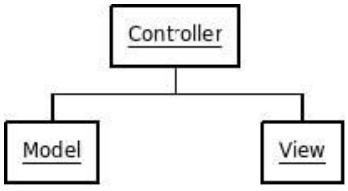
HTTP (*Hyper Text Transfer Protocol*) merupakan protokol yang melakukan dan menentukan *request* dan *response* antar halaman web. HTTP memproses *header* yang dikirimkan tiap kali ada perubahan halaman web, *header* pada HTTP sendiri ada banyak sekali penulisan dan aturan penulisannya. Apabila terdapat kesalahan penulisan *header* HTTP yang dikirimkan, maka halaman berikutnya akan mengalami kesalahan juga, karena *header* HTTP yang menentukan apakah yang akan diproses pada halaman selanjutnya. *Web server* merupakan *server internet* yang mampu melayani koneksi *transfer data* dalam protokol http. *Web server* melayani permintaan *client* terhadap halaman web. *Server side programming* merupakan bahasa pemrograman yang dijalankan pada sisi *server* dan bukan pada sisi *client*, namun hasil outputnya dapat disaksikan pada sisi *client*. Jadi ketika sebuah halaman web yang berisi program *server side* dijalankan, maka yang akan menjalankan program

tersebut adalah dari pihak *server* dengan *interpreter* yang ada pada *server*, kemudian hasil outputnya baru dikirimkan ke *client*. Jadi *client* hanya menerima hasil *output* dari proses yang terjadi pada sisi *server*. *Client side programming* merupakan bahasa pemrograman yang dijalankan pada sisi *client*, kode program tidak dianggap sebagai *server script*, namun sebagai data biasa [4].

Obyek adalah orang, tempat, benda, kejadian, atau konsep-konsep yang ada di dunia nyata yang penting bagi suatu aplikasi (perangkat lunak dan atau sistem informasi). Kelas merupakan kumpulan/himpunan obyek dengan atribut/properti yang mirip, perilaku (operasi) yang mirip, serta hubungan dengan obyek yang lain dengan cara yang mirip. Atribut adalah data yang dimiliki suatu obyek dalam kelas. Misalnya kelas manusia, memiliki atribut nama, usia, tinggi, berat badan, dan lain sebagainya. Operasi adalah fungsi atau transformasi yang mungkin dapat diaplikasikan ke atau oleh suatu obyek dalam kelas. Misalnya suatu obyek dalam kelas manusia mungkin memiliki fungsi-fungsi: tersenyum, marah, makan, minum, dan lain sebagainya. Agregasi adalah hubungan bagian-dari atau bagian-ke-keseluruhan. Suatu kelas/obyek mungkin memiliki/bisa dibagi menjadi kelas/obyek tertentu dimana kelas/obyek yang disebut kemudian merupakan bagian dari kelas/obyek yang terdahulu. Generalisasi dan pewarisan adalah suatu cara yang sangat berdaya-guna untuk berbagi apa yang dimiliki suatu kelas (atau obyek) bagi kelas-kelas (atau obyek-obyek) yang lain. Misalnya kelas kendaraan bermotor, mobil, truk, dan sebagainya bisa saling berbagi atribut yang sama, misalnya atribut model, tahun pembuatan, jumlah gigi transmisi, dan sebagainya [5].

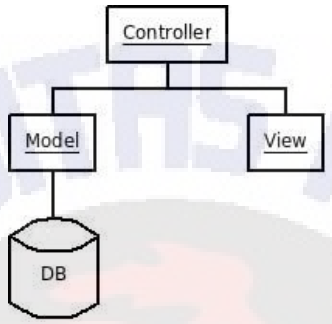
Model View Controller (MVC)

MVC merupakan konsep untuk memudahkan pengelolaan pengembangan aplikasi. Aplikasi yang semula ditulis dalam satu halaman, dipisah-pisahkan menjadi tiga bagian yang terpisah, tapi saling berhubungan. Bagian-bagian itu adalah *Model*, *View*, dan *Controller*. *Model* merepresentasikan struktur data dari aplikasi web. Pada umumnya, model berisi fungsi-fungsi yang berhubungan dengan *database*, seperti pengambilan data, *update data*, *insert data*, *delete data*, dan lain sebagainya. *View* merupakan informasi yang direpresentasikan kepada *user*. *View* biasanya berupa halaman web, dimana *client* dapat melihat informasi yang ditampilkan. *Controller* memberikan pelayanan yang menjembatani *Model* dan *View*. *Controller* berisi fungsi-fungsi yang dapat membantu menjembatani *Model* dan *View*. *Controller* merupakan *level* tertinggi dari keseluruhan *level* MVC. *Request* datang dan di*response* melalui *Controller*, kemudian *Controller* berkomunikasi serta melakukan kontrol terhadap *View* dan *Model*. Arsitektur MVC terlihat pada Gambar 1.



Gambar 1 Arsitektur MVC

Arsitektur MVC memiliki aturan-aturan tertentu dalam proses komunikasi antar *level*-nya yaitu: *Model* ataupun *View* tidak memiliki akses input ke *Controller*, *Model* ataupun *View* tidak secara langsung berkomunikasi satu sama lain, mereka hanya berkomunikasi dengan *Controller*, hanya *Model* yang memiliki akses ke *database*, *Controller* ataupun *View* keduanya tidak perlu mengerti proses ke *database*, baik itu *insert data*, ataupun memproses *query* [7]. Hubungan antar bagian MVC terlihat pada Gambar 2.

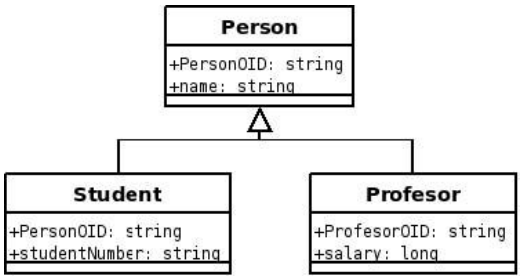


Gambar 2 Hubungan Antar Bagian MVC

Dilihat dari konsep MVC yang memisahkan kode program ke dalam bagiannya masing-masing baik sebagai *Model*, *View*, atau *Controller*, maka suatu kode program dapat tergolong untuk masuk ke bagian *Model* apabila kode program tersebut berhubungan langsung dengan akses ke *database*, seperti menyimpan data, menghapus data, mengambil data, dan lain sebagainya. Suatu kode program dapat tergolong untuk masuk ke bagian *View* apabila kode program tersebut berhubungan dengan proses menampilkan data atau sebagai *presentation layer*, dan suatu kode program dapat tergolong untuk masuk ke bagian *Controller* apabila kode program tersebut berhubungan dengan manipulasi nilai dari hasil *post* atau *get*, seperti validasi nilai, *xss cleaning*, dan lain sebagainya.

Object Relational Mapping (ORM)

Pendekatan berorientasi obyek atau *object oriented* sekarang ini banyak sekali digunakan dalam pengembangan aplikasi bisnis. Banyak sekali kelebihan dengan digunakannya pendekatan ini. Pendekatan berorientasi obyek dan pendekatan relasional memiliki paradigma yang berbeda. Pendekatan berorientasi obyek sering merujuk pada hal-hal dalam pengembangan *software*, seperti pemasangan, pemaduan, enkapsulasi, dan lain sebagainya. Sedangkan pendekatan relasional didasarkan pada prinsip matematika. Ketika kedua paradigma ini digabungkan, muncullah *Object Relational Mapping*, yaitu pemetaan dari model relasional ke model obyek, atau sebaliknya, dimana tabel *database* dipetakan ke dalam kelas-kelas, atau sebaliknya kelas-kelas dipetakan ke dalam bentuk *database* relasional [8]. Pemetaan *database* dapat dilihat pada Gambar 3.

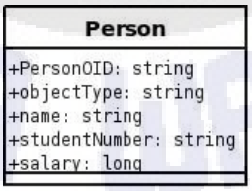


Gambar 3 Pemetaan *Database*

Suatu kelas kadang tidak memiliki atribut sama sekali, kadang memiliki satu atribut, dan dapat juga memiliki lebih dari satu atribut. Ketika mendesain kelas untuk dipetakan ke dalam *database*, atribut-atribut inilah yang menjadi kolom-kolom dari tabel yang merepresentasikan kelas yang dibuat. Setiap atribut pada kelas yang dibuat akan membentuk satu kolom atau sering disebut *field* pada tabel *database*.

Dalam proses perancangan kelas untuk dipetakan dalam *database*, pasti memerlukan ketelitian dan berbagai cara agar *database* lebih mudah dikelola. Berikut ini tiga cara yang digunakan untuk perancangan kelas yang nanti akan dipetakan dalam *database*, ketiganya memiliki ciri-ciri masing-masing dan memiliki kelebihan dan kekurangannya masing-masing.

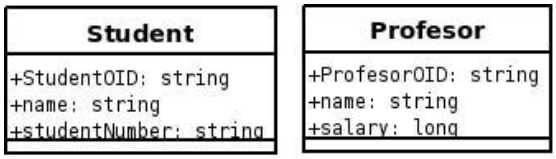
Pertama, penggunaan satu entitas data untuk semua hirarki. Dengan pendekatan ini satu kelas dengan berbagai atributnya dipetakan ke dalam satu entitas data, dalam hal ini tabel, semuanya menjadi satu. Keuntungan penggunaan pendekatan ini adalah karena kesederhanaan dan kemudahannya dalam menemukan data, karena semua data dapat ditemukan dalam satu tabel. Sebaliknya, kelemahannya adalah jika kelas yang bersangkutan mengalami penambahan atribut, maka perlu juga menambah kolom baru ke dalam tabel dan apabila terjadi kesalahan penempatan nilai pada atribut, maka seluruh data juga akan mengalami kesalahan, dan model ini banyak memakan *resource harddisk*, karena data bertumpuk menjadi satu. Untuk lebih jelasnya dapat dilihat pada Gambar 4.



Gambar 4 Satu Entitas Data untuk Semua Hirarki Kelas

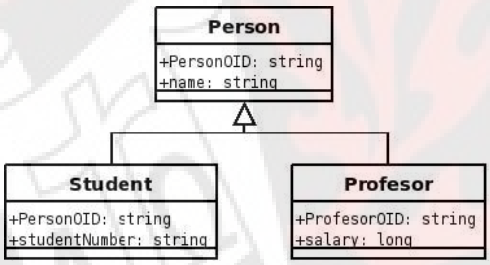
Kedua, penggunaan satu entitas data tiap kelas konkret. Pada Gambar 4, seluruh atribut pada kelas dipetakan dalam satu tabel secara keseluruhan, namun dengan menggunakan pendekatan ini, kelas tadi dibagi lagi menjadi kelas yang lebih spesifik, sehingga pengelolaannya menjadi lebih baik. Keuntungannya sebenarnya sama dengan pedekatan pertama, yaitu kemudahannya dalam penampilan laporan atau pencarian data, karena satu kelas dasar dipetakan dalam satu tabel secara keseluruhan. Kelemahan dengan menggunakan pendekatan ini adalah jika terjadi perubahan atau penambahan atribut pada kelas induk, maka akan terjadi penambahan

atribut pula pada subkelasnya dan ini juga terjadi pada pemetaan di tabel *database*nya. Masalah lain adalah sulitnya menerapkan aturan yang bersifat *multiple*, misalnya harus ditempatkan seseorang dimana dia adalah seorang *student* dan juga seorang *profesor*, seperti pada Gambar 5.



Gambar 5 Satu Entitas Data Tiap Kelas Konkret

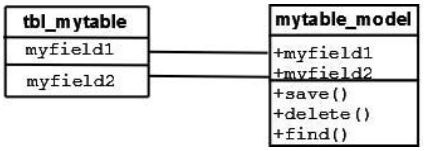
Ketiga, penggunaan satu entitas data tiap kelas. Dengan menggunakan pendekatan ini, maka akan terbentuk satu tabel untuk tiap kelas seperti terlihat pada Gambar 6. Keuntungan dalam menggunakan pendekatan semacam ini adalah pendekatan ini paling sesuai dengan pendekatan berorientasi obyek, mudah mengubah kelas *super*, dan menambahkan subkelas baru. Kelemahan dari pendekatan ini adalah banyaknya tabel pada *database*, karena tabel yang terbentuk didasarkan pada jumlah kelas yang dirancang, proses membaca data akan memakan waktu lebih lama, karena banyaknya tabel di *database*, dan proses pembuatan laporan atau pencarian data menjadi lebih susah karena banyaknya tabel di *database*.



Gambar 6 Satu Entitas Data Tiap Kelas

Implementasi Pemetaan

Prinsip dari ORM yaitu pemetaan entitas data ke dalam kelas atau sebaliknya, Gambar 7 merupakan contoh sederhana dari pemetaan tersebut. Pada Gambar 7, sisi sebelah kiri merupakan gambaran tabel di *database* dengan nama tbl_mytable, sedangkan sisi sebelah kanan merupakan gambaran kelas yang dipetakan dengan nama mytable_model. Setelah proses pemetaan, dapat dilihat bahwa *field-field* dari tabel telah dipetakan menjadi atribut pada kelas, diikuti dengan *method-method* yang menangani operasi pada *database*. Dengan demikian, setiap terjadi operasi *database*, maka data akan dikembalikan ke dalam atribut, sehingga dapat diakses sebagai obyek.

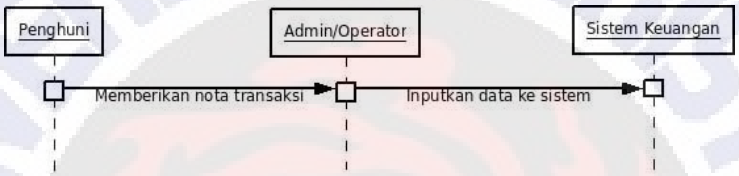


Gambar 7 Contoh Penerapan Pemetaan

6. Perancangan Sistem

Perancangan sistem dibutuhkan untuk membantu proses pengembangan dan untuk dokumentasi perangkat lunak sistem. Pada perancangan sistem ini akan diuraikan mengenai elemen-elemen pengembangan sistem yang digunakan, yaitu *Unified Modelling System* (UML) dan perancangan antarmuka sistem dengan pengguna. UML adalah sebuah "bahasa" yang telah menjadi standar untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML pada sistem yang dibangun ini terdiri dari *use case diagram*, *sequence diagram*, *activity diagram*, *class diagram*, dan *deployment diagram*. *Sequence Diagram*

Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem yang berupa *message* yang digambarkan terhadap waktu. Gambar 8 menunjukkan *sequence diagram* sistem.



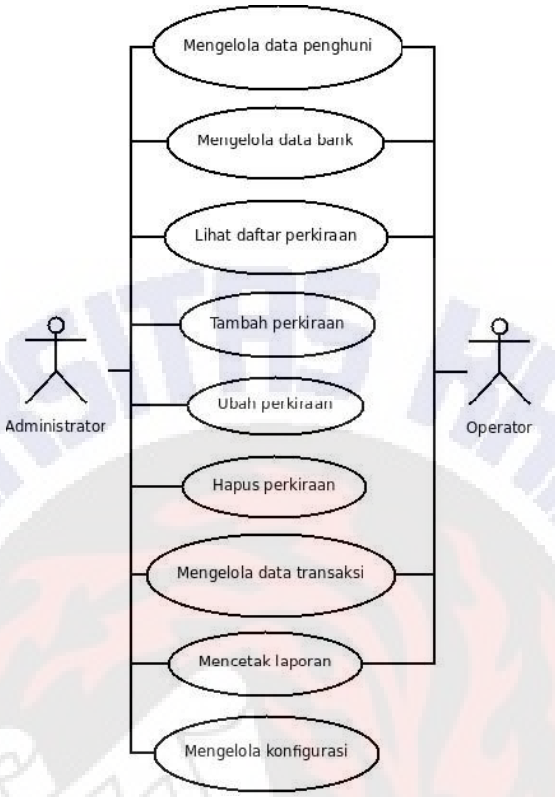
Gambar 8 *Sequence Diagram*

Use case diagram menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dan sistem. Dari Gambar 9 didapati beberapa aturan dalam aplikasi, yang dibagi menjadi dua, yaitu aturan untuk *user* administrator, dan aturan untuk *user* operator. *User* administrator dapat melakukan semua proses dalam aplikasi, seperti mengelola data penghuni, mengelola data bank, melihat daftar perkiraan, menambah perkiraan, mengubah perkiraan, menghapus perkiraan, mengelola data transaksi, mencetak laporan, dan mengelola konfigurasi. Untuk *user* operator hanya beberapa proses yang dapat dilakukan, yaitu mengelola data penghuni, mengelola data bank, melihat daftar perkiraan, mengelola data transaksi, dan mencetak laporan.

Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. Gambar 10 merupakan *activity diagram* proses *Create Read Update Delete* (CRUD) pada proses transaksi, dimana untuk diagram pada proses lain, seperti bank, penghuni, dan lain sebagainya sama dengan diagram pada proses transaksi.

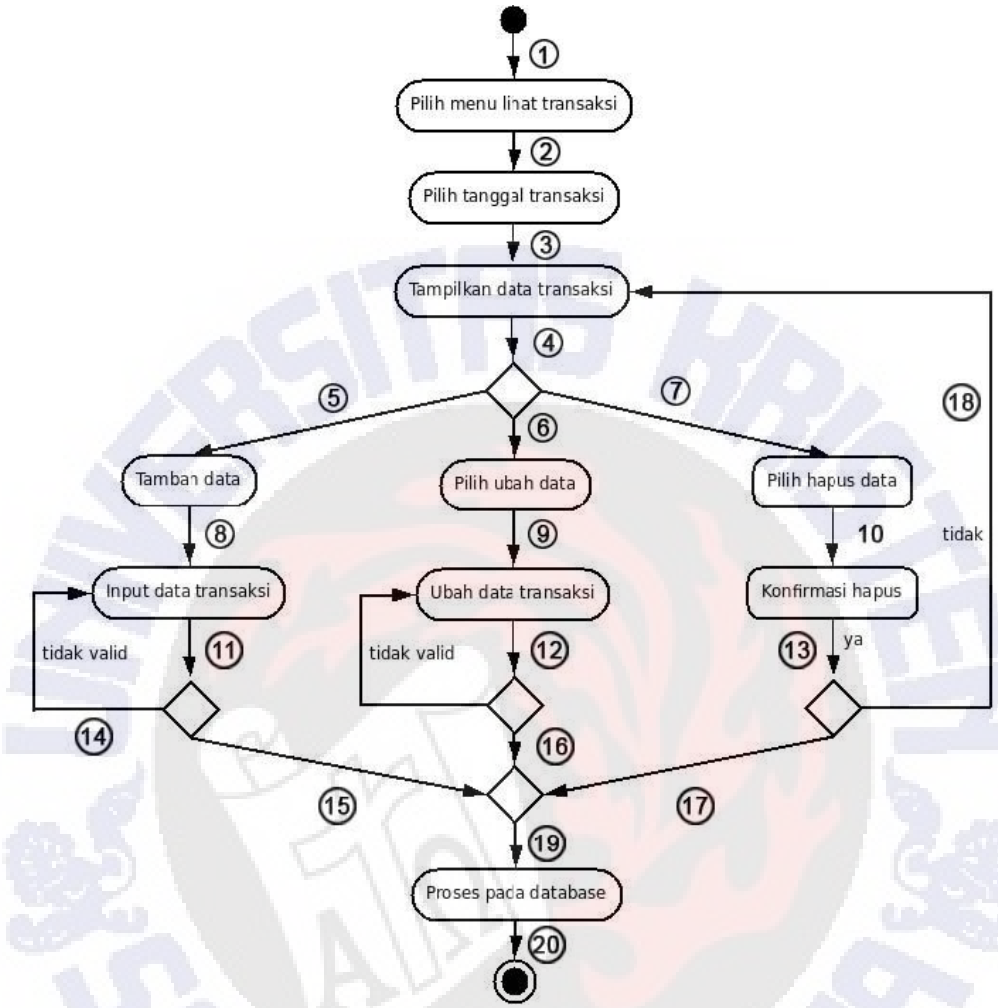
Activity diagram proses CRUD dapat dijelaskan sebagai berikut:

1. Mulai masuk ke sistem aplikasi, dan memilih menu lihat transaksi.
2. Pilih tanggal transaksi.
3. Sistem secara otomatis menampilkan data transaksi berdasarkan tanggal yang telah dipilih tadi.
4. Di sini *user* memilih salah satu dari tambah data, ubah data, atau hapus data.
5. *User* memilih pilihan proses tambah data.
6. *User* memilih pilihan proses ubah data.
7. *User* memilih pilihan proses hapus data.



Gambar 9 *Use Case Diagram Activity Diagram*

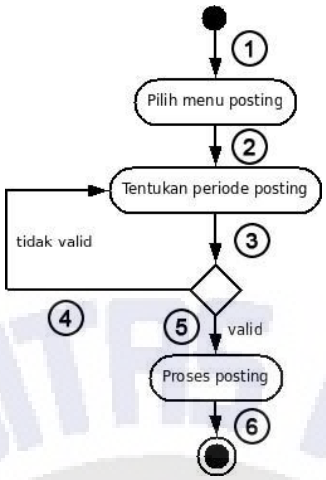
8. *User* diminta untuk menginputkan inputan/data transaksi baru.
9. *User* diminta untuk mengubah data.
10. Setelah dipilih hapus data, maka otomatis sistem akan menampilkan konfirmasi hapus, di sini *user* diminta untuk mengkonfirmasi ulang apakah data akan dihapus atau tidak.
11. Data dikirimkan ke sistem untuk divalidasi.
12. Data yang diubah dikirimkan ke sistem untuk divalidasi.
13. *User* mengkonfirmasi bahwa benar-benar akan menghapus data, maka konfirmasi diteruskan ke sistem untuk diproses.
14. Jika inputan data tidak valid, maka sistem akan mengalihkan *user* ke halaman input data.
15. Jika data inputan valid, maka sistem akan meneruskan data untuk diproses.
16. Jika data yang diubah valid, maka sistem akan meneruskan data untuk diproses.
17. Setelah *user* mengkonfirmasi hapus data, maka sistem meneruskan konfirmasi untuk diproses.
18. Jika *user* mengkonfirmasi untuk tidak menghapus data, maka sistem akan mengalihkan user ke halaman tampil data.
19. Data valid diproses ke *database*.
20. Proses telah selesai.



Gambar 10 *Activity Diagram* Proses CRUD

Gambar 11 merupakan *activity diagram posting* yang dapat dijelaskan sebagai berikut:

1. Mulai masuk ke sistem aplikasi, dan memilih menu *posting*.
2. Memilih periode *posting*, baik tanggal awal *posting*, maupun tanggal akhir *posting*.
3. Data tanggal *posting* diproses untuk divalidasi.
4. Jika data tanggal tidak valid, maka sistem akan mengalihkan *user* ke halaman awal *posting* untuk menginputkan kembali tanggal yang valid.
5. Jika data tanggal valid, maka sistem akan meneruskan untuk memproses *posting* dari periode yang diinputkan tadi.
6. Proses selesai.



Gambar 11 *Activity Diagram Posting*

Gambar 12 merupakan *activity diagram posting* yang dapat dijelaskan sebagai berikut:

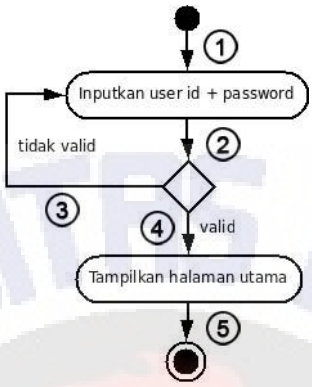
1. Mulai masuk ke sistem aplikasi, dan memilih jenis laporan.
2. Jika ingin menampilkan laporan berdasarkan periode tertentu, inputkan periode laporan yang ingin ditampilkan.
3. Sistem kemudian akan memproses permintaan *user*, dan akan menampilkan laporan berdasarkan periode yang sudah diinputkan tadi.
4. Proses selesai.

Gambar 12 *Activity Diagram* Laporan

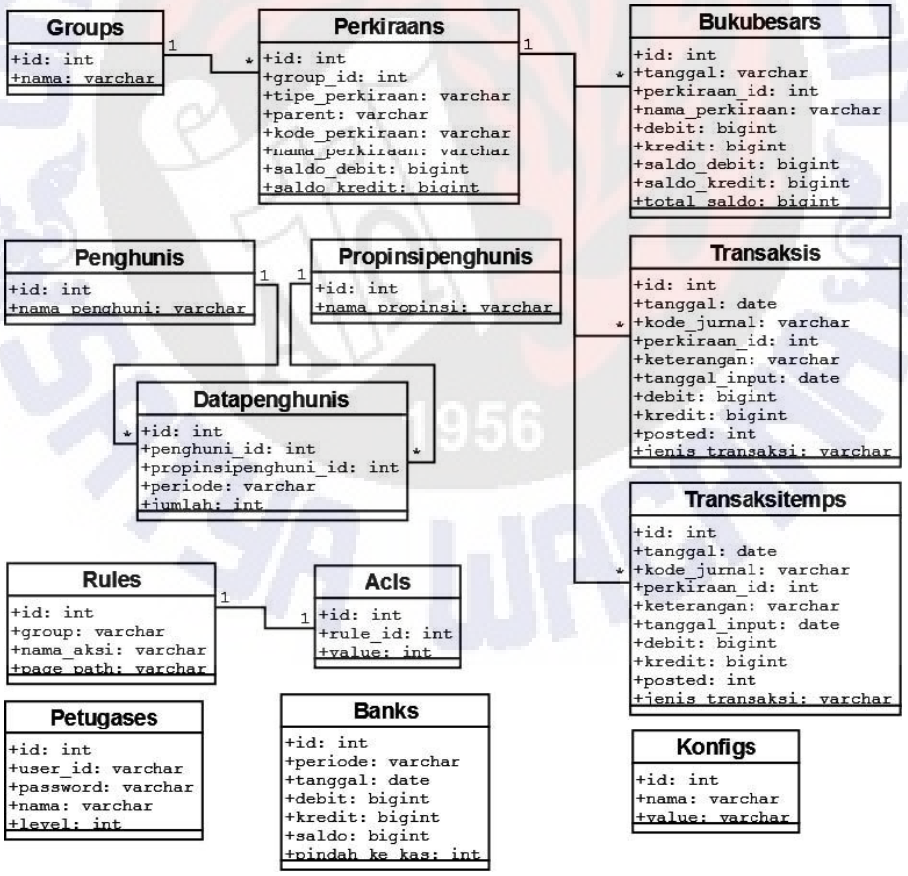
Gambar 13 merupakan *activity diagram posting* yang dapat dijelaskan sebagai berikut:

1. Mulai masuk ke sistem aplikasi, *user* diminta untuk menginputkan *user id* dan *password* untuk dapat mengakses aplikasi utama.
2. Setelah *user* menginputkan *user id* dan *password*, sistem kemudian memproses inputan untuk divalidasi.
3. Jika inputan *user id* dan *password* salah, maka sistem akan mengalihkan *user* ke halaman awal *login*.

4. Jika inputan *user id* dan *password* benar, maka sistem akan meneruskan menampilkan halaman utama aplikasi, di sini *user* sudah dapat mengakses aplikasi utama.
5. Proses selesai.



Gambar 13 *Activity Diagram Login*



Gambar 14 *Class Diagram* Perancangan *Database*

Class diagram menggambarkan alur kelas, dalam hal ini desain masing-masing kelas dan hubungan antara satu kelas dengan yang lain dalam sistem. *Class diagram database* merupakan perancangan kelas yang ditujukan pada pembuatan *database*. Gambar 14 menggambarkan *class diagram* perancangan *database*.

4. Implementasi dan Analisa Hasil

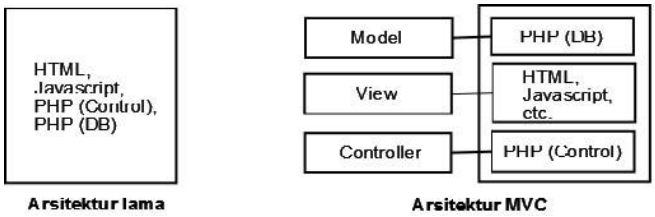
Dari penerapan konsep ORM dan MVC pada aplikasi CMS Sistem Informasi Keuangan dengan studi kasus pada Wisma Retreat Betlehem Salatiga, didapatkan bahwa aplikasi yang telah dibuat sudah dapat memenuhi kebutuhan umum yang diperlukan Wisma Retreat Betlehem Salatiga, yaitu pencatatan penghuni, pencatatan keuangan bank, pencatatan transaksi harian, hingga menghasilkan laporan keuangan, seperti laporan jurnal umum, laporan buku besar, dan rugi laba. Untuk output laporan jurnal umum dapat dilihat pada Gambar 15.

Laporan Jurnal Umum
Sistem Informasi Keuangan
Periode : Agustus 2008

Tgl.	Kd. Jrnl	KP.	Keterangan	Debit	Kredit
01	JN00000001	00.1	KAS	2,000,000	0
		05.1	SUMBANGAN U/ BELI KOMPUTER	0	1,500,000
		05.1	SUMBANGAN U/ BELI KOMPUTER	0	500,000
	JN00000002	7.1	UANG SAKU 5 ROMO	1,000,000	0
		7.1	UANG SAKU 1 TOPER	50,000	0
		13.1	MATERIAL U/ PERBAIKAN SELOKAN	192,500	0
		13.1	REPARASI LISTRIK POMPA AIR	200,000	0
		2.12	BUKU KAS EKONOMI NOVIS	5,600	0
		6.2	GAJI BU YASMI	126,500	0
		6.2	GAJI BU SOFIA	113,000	0
		1.12	BELANJA MINGGU	150,000	0
		00.1	KAS	0	1,837,800
02	JN00000003	00.1	KAS	1,200,000	0
		05.1	SUMBANGAN KEL.CHANDRA B.	0	1,200,000
	JN00000004	2.13	KOMPUTER U/ NOVISIAT	2,800,000	0
		2.13	COVER KOMPUTER	40,000	0
		1.22	SEPATU RM. SUMARGO	79,900	0

Gambar 15 Output Laporan Jurnal Umum

Konsep arsitektur MVC yang telah diterapkan pada aplikasi CMS Sistem Informasi Keuangan ini dirasa sangat memudahkan dalam proses pengembangan aplikasi, karena keteraturan penulisan kode program yang memisahkan kode program ke dalam bagiannya masing-masing. Perbandingan penulisan kode program menggunakan arsitektur MVC dengan penulisan kode program pada satu ruang kerja (arsitektur lama) dapat dilihat pada Gambar 16.



Gambar 16 Perbandingan Arsitektur

Gambar 16 merupakan perbandingan antara penulisan kode program menggunakan arsitektur MVC dengan arsitektur lama, yaitu bahwa dengan MVC, kode program dapat dikelompokkan ke dalam kelompoknya masing-masing sesuai kegunaannya, yang akan memudahkan *programmer* untuk fokus pada bagian per bagian kode program.

Dengan diimplementasikannya ORM pada aplikasi Sistem Informasi Keuangan, akan memudahkan *programmer* melakukan pengembangan aplikasi, karena selain ORM yang berbasiskan obyek sehingga mudah dipahami, ORM juga membuat penulisan kode program menjadi lebih sederhana, karena sangat jarang penulisan *statement* SQL dilakukan oleh *programmer*, kecuali pada kondisi tertentu dimana suatu *statement* SQL tidak bisa ditangani oleh *framework* ORM. Kondisi tersebut terpaksa diatasi dengan penggunaan *statement* SQL. Pada contoh kasus Sistem Informasi Keuangan ini ditemukan beberapa kondisi dimana *statement* SQL tidak dapat ditangani oleh *framework* yang dipakai, yaitu Kohana. Kondisi itu antara lain pada proses pengambilan data dari *database*, dimana data yang diambil memiliki tujuh karakter dari kiri "*field* tanggal" adalah bulan dan tahun sekarang. *Framework* Kohana tidak dapat mengatasi kondisi seperti ini, sedangkan dengan *statement* SQL, hal ini sangat mungkin dilakukan. Kemudahan dan efisiensi penulisan kode program menggunakan ORM dapat dilihat pada perbandingan Gambar 17.

```
Kode untuk mengambil dan mencetak data dengan id 1 dari
tabel user dengan menggunakan Object Relational Mapping

function ambil_data() {
        $users = new User_Model;
        $users->find(1);
        echo $users->userid;
        echo $users->password;
}
```

```
Kode untuk mengambil dan mencetak data dengan id 1 dari
tabel user dengan menggunakan statement SQL biasa

function ambil_data() {
        $sql = "SELECT * FROM users";
        $results = mysql_query($sql);
        foreach($results as $result) {
                echo $result->userid;
                echo $result->password;
        }
}
```

Gambar 17 Perbandingan Kode Program menggunakan ORM dan SQL Biasa

Dapat dilihat pada Gambar 17, data yang diambil melalui *meth*od pada *library* ORM dikembalikan lagi dalam bentuk obyek, sehingga dapat dengan mudah dipanggil dengan perintah $nama_model->nama_obyek. Sedangkan pengambilan data dengan *statement* SQL biasa pengembalian datanya dalam bentuk *record*, sehingga untuk menampilkan data harus menelusuri *record* hasil pengembalian tadi.

Perbandingan ORM dan *Active Record*

Active Record merupakan *abstraction layer* yang menangani operasi *database*, sama halnya dengan penggunaan *statement* SQL biasa dalam penulisan kode program, *Active Record* juga melakukan manipulasi *database* dengan cara mengeksekusi *statement* SQL secara langsung.

Pembuatan aplikasi web, tidak lepas dari performa aplikasi, karena aplikasi web nantinya dapat diakses oleh banyak orang, sehingga kecepatan aplikasi, dan manajemen memori disini penting untuk diperhatikan. Tabel 1 memaparkan perbandingan kinerja antara ORM dan *Active Record* untuk menjalankan beberapa *statement* umum SQL.

Tabel 1 Perbandingan Kinerja ORM dan *Active Record*

		ORM	*Active Record*	**Selisih**
SELECT	Waktu (detik)	0,0006	0,0003	0,0003
	Memori (*byte*)	3.472	816	2.656
WHERE	Waktu (detik)	0,0007	0,0003	0,0004
	Memori (*byte*)	3.272	360	2.912
INSERT	Waktu (detik)	0,0315	0,0331	0,0016
	Memori (*byte*)	2.032	752	1.280
UPDATE	Waktu (detik)	0,0333	0,0444	0,0111
	Memori (*byte*)	3.520	360	3.160
DELETE	Waktu (detik)	0,0004	0,0002	0,0002
	Memori (*byte*)	1.160	1.016	144

Dari hasil percobaan terlihat bahwa dari beberapa *statement* SQL yang dijalankan, *Active Record*-lah yang lebih unggul baik dari hal memori yang digunakan, maupun kecepatannya. Untuk *statement SELECT*, *WHERE*, *UPDATE*, dan *DELETE*, *Active Record* lebih unggul dibandingkan ORM, dalam hal kecepatan dengan rata-rata 0.0003 detik, dan juga dalam hal memori yang dipakai, *Active Record* unggul dengan rata-rata 1904 *byte*. Sedangkan untuk *statement INSERT* dan *UPDATE*, ORM lebih unggul dalam hal kecepatan, dengan rata-rata 0.00635 detik, sedangkan dalam hal memori yang dipakai, *Active Record* lebih unggul dengan rata-rata 2220 *byte*.

Dari percobaan, dapat disimpulkan bahwa dalam pengembangan aplikasi, apabila fokus dari pengembangan adalah untuk mendapatkan kemudahan dalam pengembangan aplikasi, ORM merupakan konsep yang baik digunakan, dengan kemudahan-kemudahan yang disediakan. Sedangkan apabila fokus dari pengembangan aplikasi adalah untuk menonjolkan kecepatan akses data, *Active Record*-lah yang lebih baik digunakan.

5. Simpulan

Simpulan yang dapat diambil dari implementasi MVC dan ORM pada aplikasi CMS Sistem Informasi Keuangan dan hasil percobaan yang telah dilakukan adalah (1) Proses penulisan kode program dengan menggunakan arsitektur MVC menjadi lebih terorganisasi dengan baik, dengan demikian aplikasi dapat lebih mudah dikembangkan; (2) Dengan digunakannya ORM, hampir tidak ada *statement* SQL yang ditulis pada halaman kerja, kecuali untuk kasus tertentu dimana *framework* tidak dapat menangani suatu *statement* SQL, sehingga dengan diterapkannya ORM, penulisan kode program menjadi lebih bersih, efisien dan dengan demikian, aplikasi dapat lebih mudah dikembangkan; (3) Untuk pengembangan aplikasi yang memfokuskan pada kemudahan untuk mengembangkan aplikasi, ORM sangatlah baik untuk menangani hal ini, namun demikian ORM juga memiliki kelemahan, yaitu kecepatan yang lebih lambat, dan pemakaian memori yang lebih banyak dibandingkan dengan menggunakan *Active Record*.

6. Daftar Pustaka

[1] Wiki. 2008. Sistem Manajemen Konten. *id.wikipedia.org/wiki/Sistem manajemen_konten*. Diakses tanggal 20 Agustus 2008.

[2] Orsag, Jaroslav. 2006. Object Relational Mapping. *http://dsrg.mff.cuni.cz/teaching/seminars/2006-03-21-Orsag-ORM.pdf*. Diakses tanggal 15 Juli 2008.

[3] Deacon, John. 2005. Model-View-Controller (MVC) Architecture. *http://www.jdl.co.uk/briefings/mvc.pdf*. Diakses tanggal 16 Juli 2008.

[4] Newman, Frans. 2002. *Membangun Database Web dengan CGI dan Database Server*. Jakarta: PT. Elex Media Komputindo.

[5] Nugroho, Adi. 2005. *Analisis dan Perancangan Sistem Informasi dengan Metodologi Berorientasi Objek*. Bandung: Informatika.

[6] Benarkah. 2004. Pembuatan Content Management System. http://*ejournal.gunadarma.ac.id/files/A16.pdf*. Diakses tanggal 16 Agustus 2008.

[7] Butterfield, Tim. MVC Discussion. *http://forum.kohanaphp.com/comments.php?DiscussionID=23*. Diakses tanggal 15 Juli 2008.

[8] Ambler, Scott W. 2000. Mapping Object To Relational Databases. *http://www.ibm.com/developerworks/library/ws-mapping-to-rdb*. Diakses tanggal 16 Agustus 2008.