MEMAHAMI ALGORITMA PEMOGRAMAN
(Analisis Pembelajaran dalam Implementasi Software)

Oleh: Yuni Sugiarti *

Abstrak

Kenyataan menunjukan bahwa pembelajaran pemograman komputer masih banyak yang mengalami kesulitan baik secara teori ataupun praktek. Akibatnya kurang mengahayati atau memahami konsep-konsep pemograman dan kesulitan mengaplikasikan pemograman komputer untuk aplikasi bisnis, pendidikan, game, dll. Salah satu cara pembelajaran pemograman komputer adalah dituntut untuk membiasakan menggunakan algoritma. Algoritma yaitu urutan langkah atau tahapan-tahapan berhingga untuk memecahkan masalah logika/matematika, dimulai dengan mendefinisikan masalah, menentukan solusi, memilih algoritma, menulis program, menguji program, menulis dokumentasi sampai merawat program atau maintenance. Tulisan ini menganalisis secara teoritis pembelajaran pemograman komputer menggunakan algoritma, serta implementasinya dalam kasus konversi software engeenering dikonversi ke sistem data base.

Kata kunci:
pembelajaran, pemograman, algoritma pemograman

*) *Yuni Sugiarti, ST. adalah dosen Teknik Informatika beberapa PT di Bandung.*

A. PENDAHULUAN

Dewasa ini, komputer digunakan di hampir semua bidang kehidupan manusia, mulai dari pendidikan, bisnis, kesehatan, permainan dan lain-lain. Khususnya dalam bidang pendidikan, komputer dipakai untuk pembelajaran sekolah mulai TK, SD, SMP, SMU, hingga PT. Komputer (*software*) juga sebagai alat bantu dalam pengelolaan pendidikan misalnya sistem akademik baik untuk administrasi, keuangan, kepegawaian, perwalian, nilai, dan banyak produk-produk perangkat lunak lainnya.

Berbicara tentang komputer terutama *software* tidak lepas dari pemograman komputer. Hal ini karena komputer pada dasarnya adalah mesin yang tidak bisa apa-apa. Kita harus memberikan serangkaian instruksi kepada komputer agar mesin 'pintar' ini dapat memecahkan suatu masalah. Langkah-langkah yang kita lakukan dalam memberikan instruksi kepada komputer untuk memecahkan masalah inilah yang dinamakan algoritma pemograman komputer.

Dalam kehidupan sehari-hari, untuk berkomunikasi dengan orang lain, kita harus menggunakan bahasa yang sama dengan orang tersebut. Apabila kita menggunakan Bahasa Inggris, lawan bicara kita juga harus mengerti Bahasa Inggris. Kalau lawan bicara kita tidak mengerti Bahasa Inggris, kita masih bisa berkomunikasi dengannya melalui seorang penerjemah.

Dalam pemograman komputer, berlaku juga hal seperti di atas, menggunakan bahasa yang dimengerti oleh komputer untuk memberikan suatu instruksi. Pada dasarnya, komputer adalah mesin digital, artinya komputer hanya mengenal kondisi ada arus listrik (biasanya dilambangkan dengan 1) dan tidak ada arus listrik (biasanya dilambangkan dengan 0). Dengan kata lain harus menggunakan kata sandi 0 dan 1 untuk melakukan pemograman komputer. Bahasa pemograman yang menggunakan sandi 0 dan 1 ini disebut bahasa mesin. Mungkin kita sudah bisa membayangkan bagaimana sulitnya memprogram dengan bahasa mesin. Namun tidak perlu kuatir karena dewasa ini jarang sekali orang yang

memprogram dengan bahasa mesin.

Karena bahasa mesin sangat susah, maka muncul ide untuk melambangkan untaian sandi 0 dan 1 dengan singkatan kata yang lebih mudah dipahami manusia yang disebut *mnemonic code*. Bahasa pemograman yang menggunakan singkatan kata ini disebut bahasa *assembly*.

Sebagai contoh, dalam prosesor intel, terdapat perintah 0011 1010 0000 1011. Perintah dalam bahasa mesin ini sama artinya dengan perintah assembly CMP AL, 0D, yang artinya bandingkan nilai register AL dengan 0D. CMP di sini sebenarnya adalah singkatan dari CoMPare. Dapat kita lihat disini bahwa perintah CMP AL, 0D jauh lebih mudah dipahami daripada 0011 1010 0000 1011. Jika dilihat dari sudut pikiran manusia, bagi komputer, kombinasi 0 dan 1 tentu lebih mudah dipahami. Perangkat lunak yang mengkonversikan perintah-perintah *assembly* ke dalam bahasa mesin sering disebut juga *assembler*.

Berikutnya *assembly* digantikan bahasa pemograman generasi ketiga atau 3GL (*Third Generation Language*) atau disebut bahasa tingkat tinggi (HLL = *High Level language*). Contohnya : Basic, Pascal, C, C++, COBOL dan sebagainya.

Bahasa pemograman generasi berikutnya adalah generasi ke 4 atau 4GL (*fourth-generation language*). Bahasa ini banyak digunakan untuk mengembangkan aplikasi basis data (*data base*). Salah satu contohnya adalah SQL (*Structure Query Language*). Pada bahasa ini perintah-perintah yang digunakan lebih manusiawi, misalnya "SELECT Nama, Alamat FROM Karyawan", untuk mengambil Nama dan Alamat dari basis data karyawan.

Tulisan ini sesuai dengan uraian di atas, mengkaji secara teoritis pembelajaran Algoritma Pemograman (AP) dan pengimplementasian pembelajaran AP dalam kasus konversi matakuliah *software*

engeenering dikonversi ke mata kuliah *data base*.

B. Hakikat Algoritma Pemograman

Algoritma Pemograman adalah urutan langkah-langkah berhingga untuk memecahkan masalah secara logika atau matematika. Dalam pemograman komputer beberapa langkah yang perlu dilakukan, sebagai berikut:

1. Mendefinisikan masalah

Langkah pertama ini sering dilupakan oleh banyak pemogram. Begitu mereka mendapat perintah untuk membuat suatu program, mereka langsung menulis programnya tanpa mendefinisikan masalahnya terlebih dulu. Dalam buku hukum Murphy untuk pemograman karangan Henry Ledgard dikatakan "semakin cepat kita menulis program akan semakin lama kita untuk menyelesaikannya", setelah dibuktikan bersama ternyata kata-kata tersebut memang benar. Mendefinisikan masalah sangat penting tetapi sering dilupakan. Tentukan masalahnya seperti apa, kemudian apa yang harus dipecahkan dengan komputer, yang terakhir apa masukannya dan apa keluarannya.

2. Menentukan soal

Setelah masalah sudah didefinisikan dengan jelas, masukan apa yang diberikan dengan jelas, keluaran apa yang diinginkan sudah jelas, langkah selanjutnya adalah mencari jalan bagaimana masalah tersebut diselesaikan. Apabila permasalahan terlalu kompleks, biasanya kita harus membaginya ke dalam beberapa modul kecil agar lebih mudah diselesaikan.

3. Memilih algoritma

Langkah ini merupakan salah satu langkah penting dalam pemograman komputer, karena pemilihan algoritma yang salah akan menyebabkan program memiliki unjuk kerja yang kurang

baik.

4. Menulis Program

Ada beberapa hal yang harus dipertimbangkan saat memilih bahasa pemograman, antara lain masalah yang dihadapi, bahasa pemograman yang kita kuasai dan sebagainya.

5. Menguji Program

Setelah program selesai ditulis, kita harus mengujinya. Pengujian pertama adalah apakah program berhasil di kompilasi dengan baik. Pengujian berikutnya apakah program dapat menampilkan keluaran yang diinginkan.

6. Menulis Dokumentasi

Hal ini biasanya dilakukan, bersamaan menulis program, artinya pada setiap baris program atau setiap beberapa baris program, kita menambahkan komentar yang menjelaskan kegunaan dari suatu pernyataan. Dokumentasi ini kelihatannya sepele dan banyak dilupakan orang, padahal fungsinya penting sekali. Dimasa mendatang, apabila kita perlu melakukan perubahan atau perbaikan terhadap suatu program, kita akan merasakan pentingnya dokumentasi yang baik. Dokumentasi yang dijadikan satu dalam program, berupa komentar-komentar pendek, biasanya sudah cukup.

7. Merawat Program

Langkah ini dilakukan setelah program selesai dibuat dan sudah digunakan oleh pengguna kita. Hal yang sering terjadi disini munculnya *bug* yang sebelumya tidak terdeteksi. Atau juga pengguna ingin tambahan suatu fasilitas baru. Apabila hal-hal seperti ini terjadi berarti program harus direvisi ulang.

Dalam kehidupan sehari-hari, sebenarnya kita sering menggunakan algoritma untuk melakukan sesuatu. Sebagai contoh ketika menulis surat kita perlu melakukan beberapa langkah sebagai berikut:

mempersiapkan kertas dan amplop, mempersiapkan alat tulis seperti pena atau pensil, mulai menulis, memasukkan kertas ke dalam amplop, pergi ke kantor pos untuk mengeposkan surat tersebut. Langkah-langkah itulah yang di sebut dengan algoritma. Jadi sebenarnya kita sendiri juga sudah menggunakan algoritma baik sadar maupun tidak sadar.

Dalam banyak kasus, algoritma yang dilakukan tidak selalu berurutan seperti di atas, kadang-kadang harus memilih dua atau beberapa pilihan. Misalnya kita ingin makan, kita harus menentukan akan makan dirumah makan atau masak sendiri. Jika memilih untuk makan di rumah makan, kita akan menjalankan algoritma yang berbeda dengan yang memilih masak sendiri. Dalam dunia algoritma, hal semacam ini sering disebut percabangan.

Dalam kasus lain lagi, kita mungkin harus melakukan langkah-langkah tertentu beberapa kali. Misalnya saat menulis surat, sebelum memasukkan kertas ke dalam amplop, mungkin kita harus mengecek apakah surat itu sudah benar atau belum. Jika belum benar, berarti harus mempersiapkan kertas baru dan menulis lagi. Demikian seterusnya sampai surat kita sesuai dengan yang diharapkan. Dalam dunia pemograman, hal semacam ini sering disebut pengulangan.

C. IMPLEMENTASI ALGORITMA PEMOGRAMAN

Untuk memberikan gambaran tentang implementasi algoritma pemograman, berikut ini diberikan contoh kasus algoritma pemograman.

Kasus 01

Tulislah program untuk menampilkan jumlah hari, jam, menit, dan detik dari masukan yang berupa lamanya waktu dalam detik. Sebagai contoh, masukan 100.000 detik akan menghasilkan keluaran : 1 hari, 3 jam 46 menit dan 40 detik. Masukan dari program ini adalah bilangan bulat bertipe longint yang melambangkan waktu

dalam detik. Keluaran dari program ini adalah banyaknya hari, jam, menit, dan detik dari waktu tersebut.

PEMBAHASAN Kasus 01

Untuk menentukan algoritma perhitungan hari, jam, menit, dan detik, kita mulai dengan contoh sederhana. Misalkan masukan yang diberikan pengguna adalah 100.000 detik. Langkah-langkah penghitungan tersebut dapat dituliskan sebagai berikut :

1. Banyaknya hari = 100.000 / (60 * 60 * 24) = 100.000 / 86.400
   = 1 hari.
2. Banyaknya jam = (100.000 – 1 * 86.400) / (60 * 60)
   = 13.600 / 3.600 = 3 jam.
3. Banyaknya menit = (13.600 – 3 * 3.600) / 60 = 2.800 / 60
   = 46 menit.
4. Banyaknya detik = (2.800 – 46 * 60) = 2.800 – 2.760 = 40 detik.

Berdasarkan contoh di atas, dapat dituliskan algoritma pencarian hari, jam menit dan detik adalah sebagai berikut :

1. Masukkan detik.
2. hr ← detik / 86400.
3. detik ← detik - hr * 86400.
4. jm ← detik / 3600.
5. detik ← detik – jm * 3600.
6. mn ← detik / 60.
7. detik ← detik – mn * 60.
8. dt ← detik.
9. Tulis hr, jm, mn, dt.

Implementasi dalam bahasa pascal

```
Const
    SatuHari = 60 * 60 * 24;
Var
    detik: longint;
    Hr, jm, mn, dt : integer;
Begin
```

```
    Write(‘ Masukkan waktu (detik) : ‘);
    Readln (detik);
    Writeln (detik,   ‘detik terdiri dari : ‘)’

    Hr  := detik div SatuHari;
    detik := detik -  hr  * SatuHari;

    Jm := detik  div 3600;
    detik:= detik – jm * 3600;

    Mn := detik div 60;
    dt := detik  - mn * 60;

    Writeln(hr,   ‘hari’)’;
    Writeln(jm,   ‘jam’)’;
    Writeln(mn,   menit’)’;
    Writeln(dt,   ‘detik’)’;
End.
```

Hasil Keluaran program diatas adalah sebagai berikut :

```
Masukkan waktu (detik) : 100000
100000 detik terdiri dari :
1    hari
3    jam
46  menit
40   detik

Masukkan waktu (detik) : 3600
3600 detik terdiri dari :
0    hari
1     jam
0    menit
0    detik
```

Kasus 02

Tulislah program untuk konversi metodologi Objek OMT (Object Modelling Technique) ke SQL

PEMBAHASAN Kasus 02

Model Objek

Metodologi berorientasi objek merupakan suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek-objek yang berisi data dan aksi yang dilakukan terhadapnya. Salah satu metodologi ini adalah metodologi objek OMT (Object Modelling Technique). Tujuannya adalah untuk mengidentifikasi secara jelas aspek-aspek metodologi dan penerapannya terhadap metodologi yang akan dibangun. Komponen objek adalah : Objek, Kelas, Diagram-objek, atribut, Operasi dan metoda, Link dan Asosiasi, Qualifikasi, Agregasi, Generalisasi dan Pewarisan dan Metadata.

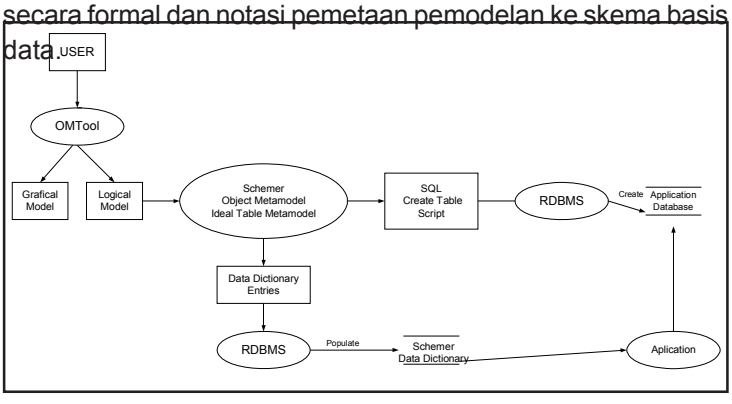
Model Relasional

Model ini bertujuan untuk menghasilkan sekumpulan skema relasi dalam bentuk yang tepat (normal), menghindari redudansi dan mengakses informasi dengan mudah.

Elemen dasar dari model relasional adalah relasi yaitu instans dari skema relasi yang berbentuk table dan terdiri atas baris dan kolom. Baris dari relasi disebut tuple, adalah kumpulan dari nilai yang dimiliki tiap atribut. Degree dari suatu relasi adalah jumlah kolom dari relasi, sedangkan kardinalitas adalah jumlah baris dari relasi. Skema model relasional adalah kumpulan skema relasi, skema relasi adalah gabungan atribut-atribut. Himpunan dari semua nilai yang mungkin dari atribut tertentu disebut domain atribut tertentu. Komponen model relasional adalah ketergantungan fungsional, relationship, key, normalisasi.

1. Aturan konversi OMT ke Model Relasional

Suatu percobaan arsitektur yang dilakukan *James Rumbaugh* pada *GE Corporate R & D,* dan *Schemer*, yaitu *batch compiler* yang dikembangkan untuk membangkitkan kode SQL dengan mengotomatisasikan aturan-aturan pemetaan melalui suatu model kompilator. Dalam gambar 1 menunjukkan arsitektur untuk mengoptimasikan konversi model objek ke model relasional. Arsitektur memiliki tiga bagian utama : *OMTool, Schemer dan RDBMS* itu sendiri. *Schemer* adalah kumpulan aturan-aturan pemetaan objek dan metamodel table-tabel secara formal dan notasi pemetaan pemodelan ke skema basis data.



Gambar 1. *Konversi model OMT ke model relasional*

a. *OMTool*

OMTool menyimpan dua bilangan terhadap model objek dalam notasi pemodelan *OMT*. Model grafik, menggambarkan bentuk kotak, garis, dan teks ke layar. Model Grafik digunakan untuk menambah antar muka dengan pemakai dan menyajikan model objek secara visual untuk dokumentasi. Model lojik merupakan ringkasan tentang pengertian yang mendasar pada gambar dan menjelaskan kelas-kelas serta keterhubungannya antara

yang satu dengan lainnya.

OMTool memiliki beberapa keistimewaan untuk membuat pembangkit model yang termudah. *OMTool* memungkinkan pemakai untuk membuat, memuat, mengubah, menyimpan dan mencetak diagram secara otomatis. *OMTool* mengelola keterhubungan lojik seperti pemakai memindahkan entitas yang terhubung.

Antarmuka disajikan dalam bentuk jendela-jendela melayang. Untuk melakukan pertanyaan secara detail tidak ditunjukkan pada diagram seperti lingkup atribut, pengijinan nilai kosong untuk setiap atribut dan kunci-kunci primer. Juga jendela untuk menentukan aturan pemetaan yang akan diaplikasikan pada semua kelas dan relasionship dalam model, yang mana secara mendasar menyediakan strategi global untuk implementasi basis data. Pemakai dapat mengesampingkan aturan-aturan pemakai global untuk kelas-kelas khusus atau relasionship pada hasil efisiensi, jangkauan perluasan dan integritas dalam kode relasional.

Ketika pemakai membuat keputusan implementasi, *OMTool* menulisnya dan model lojik pada suatu file ASCII sebagai masukan untuk kompilator skema basisdata (dalam kasus ini shemer).

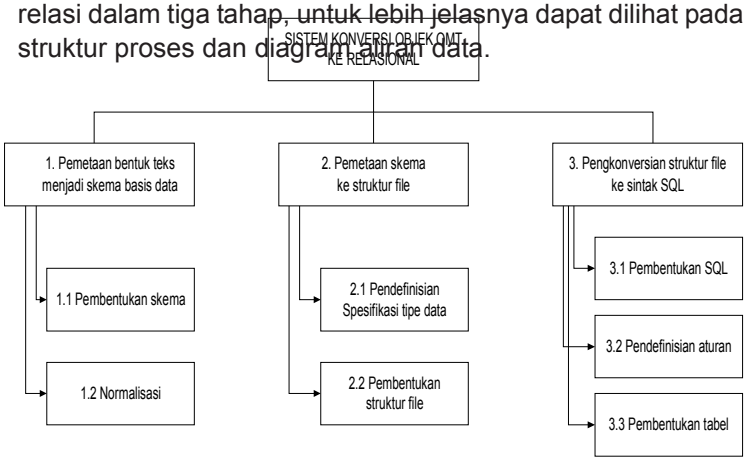
b. *Schemer*

Schemer mengkonversikan bentuk teks menjadi perintah-perintah SQL yang diperlukan untuk menciptakan table-tabel relasi, index, domain, hak akses dan bagian-bagian lain pada skema. Selama proses ini, Schemer mengeluarkan suatu pesan kesalahan seperti nama terlalu panjang atau informasi yang hilang. Schemer juga memeriksa batasan-batasan yang digunakan untuk pewarisan gkita, identitas yang diturunkan dan tipe data

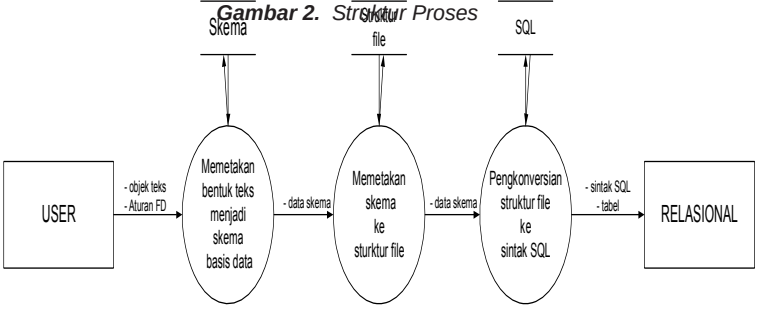
yang dihasilkan serta memperingatkan jika dipilih aturan pemetaan yang tidak konsisten.

2. Proses Konversi

Schemer mengkonversikan model objek menjadi table-tabel relasi dalam tiga tahap, untuk lebih jelasnya dapat dilihat pada struktur proses dan diagram aliran data.



Gambar 2. *Struktur Proses*



Gambar 3. *Diagram Aliran Data*

3. Metamodel

Metamodel dalam *schemer* menjelaskan struktur diagram secara jelas dan ringkas, termasuk semua keistimewaan pada notasi OMT. Metamodel juga mempermudah implementasi pada aturan-aturan untuk membangkitkan skema basisdata. Akhirnya, metamodel menyediakan kamus data umum untu banyak relasi, menyediakan informasi tentang model, dimana skema akan dipetakan. Informasi ini dibuat sebagai suatu perluasan pada kamus data yang secara umum disediakan oleh suatu relasi. Dengan informasi tambahan, pemakai memilih kamus data umum yang dapat digunakan oleh relasi.

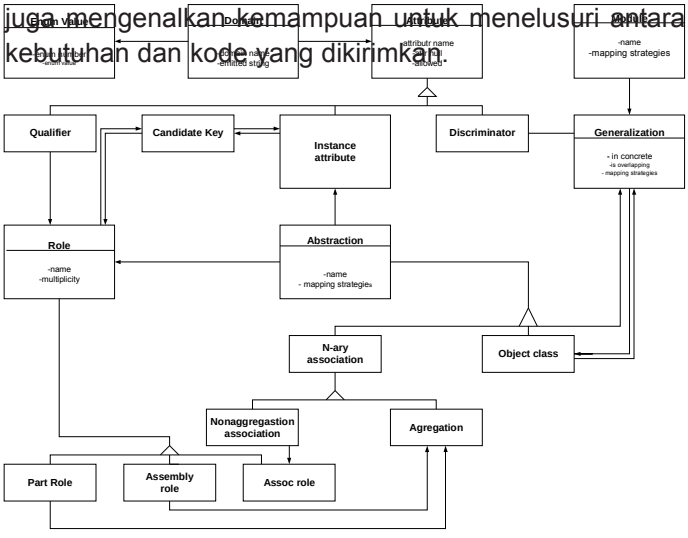
Metamodel table ideal menjelaskan semua kemungkinan table-tabel basisdata dan memungkinkan pemakai untuk merubah/ mangatur table-tabel untuk relasi tertentu.

a. Bekerja dengan Metamodel

Objek dan metamodel table ideal adalah dinamai karena itu adalah model objek dari model. Metamodel objek digunakan notasi OMT untuk menjelaskan aspek lojik dari diagram objek yang digunakan shemer. Metamodel table ideal menggunakan notasi OMT untuk menjelaskan aspek lojik dari table-tabel ideal, yang mana schemer menggunakannya untuk menjelaskan table-tabel relasional.

Metamodel memiliki keterkaitan silang yang memungkinkan untuk mengendalikan antara pembuatan berorientasi objek dan berorientasi table. Ini dapat membantu untuk pengembangan aplikasi yang harus menggali secara semantic berorientasi objek dari aplikasi bukan menyimpan data dalam table-tabel relasional. Keterhubungan silang

juga mengenalkan kemampuan untuk menelusuri antara kebutuhan dan kode yang dikirimkan.



Gambar 4. *Metamodel*

Gambar 4 merupakan rangkuman dari metamodel, yang belum lengkap. Metamodel mengenali pemodelan utama dari kelas, asosiasi, dan generalisasi, yang mendasari konsep selanjutnya tentang notasi OMT itu sendiri.

Enum Value adalah suatu nilai enumerasi termasuk domain enumerasi. Contohnya: warna, mungkin mempunyai nilai merah, biru dan hijau. Tidak semua domain adalah enumerasi, seperti dollar, yang mana tidak memiliki nilai tersendiri. OMTool menyediakan fasilitas untuk mendefinisikan enumerasi dalam suatu model.

Domain adalah tipe data basisdata. Masing-masing atribut harus diberikan domain baik secara eksplisit ataupun secara implisit. Domain memiliki nama, contohnya jalan,

mungkin didefinisikan bernilai char 30). *OMTool* menyediakan fasilitas untuk mendefinisikan domain dalam model. Sedangkan *Atribut* adalah nilai relasi yang terletak dibagian kepala atau sebagai kolom-kolom relasi. *Qualifier* berfungsi membedakan antara sekumpulan objek pada akhir sebuah asosiasi.

Instance attribute, dinamakan karakteristik dari suatu abstraksi yang menjelaskan nilai data yang dihasilkan oleh masing-masing contoh abstraksi. Suatu atribut contoh pada asosiasi adalah atribut yang terkait. *Descriminator*, suatu atribut dari jenis enumerasi menyatakan karakteristik dari kelas yang dialamatkan oleh generalisasi tertentu.

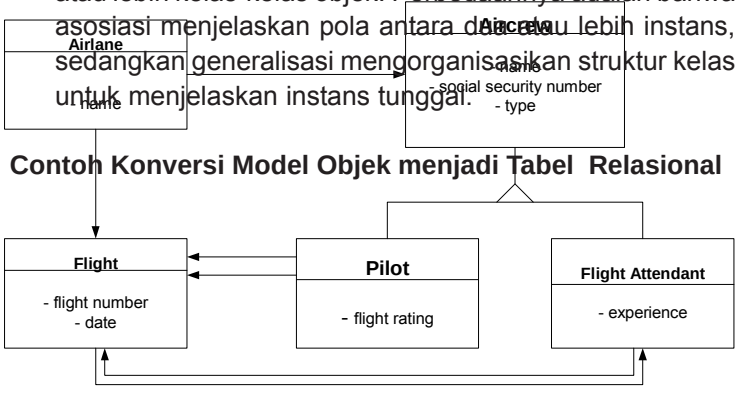
Module adalah bagian dari sistem yang berisi sejumlah kelas dan relasionshipnya. Masing-masing modul ini mempunyai nama dan strategi pemetaan, yang menentukan tetapan aturan pemetaan yang akan digunakan oleh modul. Secara umum kelas-kelas yang sama mempunyai keterhubungan yang kuat dengan asosiasi dan generalisasi daripada kelas-kelas dalam modul yang berbeda.

Generalization, relastonship antara suatu kelas atau lebih. Masing-masing generalisasi memiliki satu kelas superkelas dan satu atau lebih kelas-kelas sebagai subkelas. *Candidate Key* adalah sekumpulan minimal atribut-atribut dan aturan yang unik untuk suatu abstraksi. Sedangkan *Role*, aturan-aturan kelas digunakan dalam asosiasi. *Role* dapat sebagai *part role, assembly role* dan *assocition role*. *Part role* dalam suatu agregasi adalah aturan yang mengijinkan penggabungan. *Assembly role* dalam agregasi adalah apapun yang dibuat dari yang lain. Notasi dilambangkan dengan bentuk diamond. *Association role* menjelaskan keterlibatan dari suatu kelas dalam asosiasi yang khusus.

Abstraction, salah satu dari asosiasi *n-ary* atau kelas. Merupakan mekanisme yang memungkinkan untuk merepresentasikan dunia nyata yang kompleks menjadi satu bentuk model yang sederhana.

N-ary Assocition, keterhubungan antara dua kelas atau lebih. *Schemer* mendukung asosiasi binary & ternary. *Asosiasi N-ary* dapat memiliki atribut-atribut melalui superkelas abstraksi. *Object-class,* deskripsi dari kelompok yang memiliki property sama, tingkah laku umum, relationship umum dan semantic yang umum.

Nonagregation association, suatu asosiasi yang khusus. Kelas-kelas asosiasi mempunyai nama disebut dengan instance, mungkin mempunyai peran dalam asosiasi, dan memiliki kunci-kunci kandidat. Perbedaannya dengan objek dalam kelas memiliki identitas yang sebenarnya, identitas dari keterkaitannya dalam suatu asosiasi diturunkan dari abstraksi yang dihubungkan. Generalisasi dan asosiasi adalah relasi dasar dalam metamodel, yang melibatkan dua atau lebih kelas-kelas objek. Perbedaannya adalah bahwa asosiasi menjelaskan pola antara dua atau lebih instans, sedangkan generalisasi mengorganisasikan struktur kelas untuk menjelaskan instans tunggal.



4. Contoh Konversi Model Objek menjadi Tabel Relasional

Gambar 5. *Contoh kasus konversi*

a. Batasan, Deskripsi dengan Algoritma

1) Class dan object description

```
Class Airline
    Attribute name:string
    Attribute AssFlight : Flight
    Attribute AssAircrew :Aircrew
    Operation Set_name(Pname:string)
    Operation Set_AssFlight(Pflight number,PDate)
    Operation Set_AssAircrew(PName,PSocial security
    number)
End;
Class Aircrew
    Attribute name:string
    Attribute Social Security Number : integer
    Attribute AssAirline : Airline
    Operation Set-Name-Social Security Number
    (Pname,Psocial Security Number)
    Operation Set-assAirline(Pname)
End;

Class Pilot is a Aircrew
    Attribute Flight rating : number
    Attribute Ass_Copilot : Copilot
    Attribute Ass_Flight : Flight
    Operation Set_Flight rating(Pflight rating:number)
    Operation Set_AssCopilot(Flight rating)
    Operation Set_AssFlight(Pflight number,PDate)
End;
```

Class Flight Attendant is a Aircrew
Attribute Experience :integer
Attribute Ass_Flight : Flight
Operation Set_Experience(PExperience:integer)
Operation Set_AssFlight(Pflight number,PDate)
End;

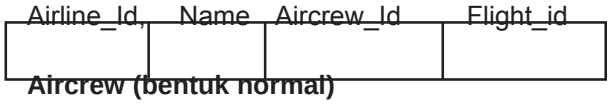
Class Flight
Attribute flight Number:integer
Attribute Date:Date
Attribute AssAirline : Airline
Attribute AssPilot:Pilot
Attribute AssCopilot:Pilot
Attribute AssFlight Attendant:Flight Attendant
Operation Set- flight Number,Date (Pflight Number,PDate)
Operation Set-AssAirline(Pname)
Operation Set-AssPilot(PFlight rating)
Operation Set-AssCopilot(Pflight rating)
Operation Set-AssFlight Attendatt(PExperience)
End;

2) Pembentukan skema berdasarkan gambar D.1.6 diperoleh deskripsi relasi sebagai berikut :

1. Airline(Airline_Id,Name,Aircrew_Id,Flight_Id)
2. Aircrew(Aircrew_Id, Name,SSN,Airline_Id)
3. Pilot(Pilot_Id, Flight rating, Aircrew_Id)
4. Flight Attendan (FlightAtt_Id,Experience,Aircrew_Id,Flight-Id)
5. Flight(Flight_Id, Flight number,Date,Airline_Id,Pilot_Id,Copilot_Id,Flight Att-Id)

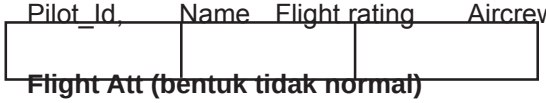
3) Normalisasi

Airline (bentuk normal)

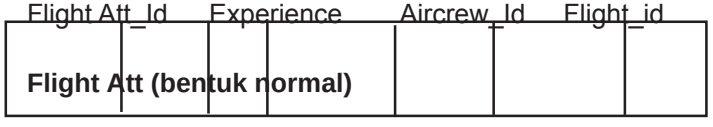


Aircrew (bentuk normal)

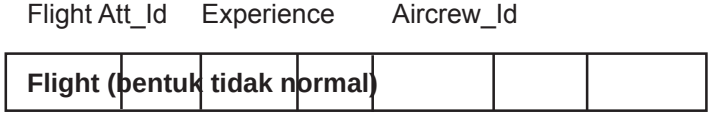
Pilot (bentuk normal)



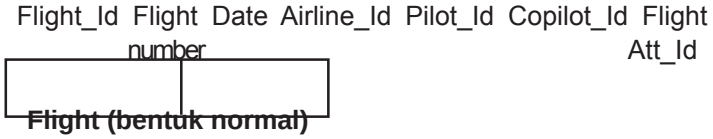
Flight Att (bentuk tidak normal)



Flight Att (bentuk normal)



Flight (bentuk tidak normal)



Flight (bentuk normal)

Flight_Id Flight number Date Airline_Id Pilot_Id
Copilot_Id

FF (bentuk normal)

Flight Att_Id Flight_Id

4) Pembentukan SQL

a) Create Table Airline
((Airline_Id Varchar[5] Notnull Default,
Name Varchar[30] Notnull Default 'noname',
Aircrew_Id Varchar[11] Notnull Default,
Flight_Id integer Notnull Default 0
Primary key (Airline_Id)
Foreign key (Aircrew_Id)
Foreign key (Flight_Id)
Reference Aircrew_Id to Aircrew (Aircrew_Id)
Reference Flight_Id to Flight (Flight_Id)
)

b) Create Table Aircrew
((Aircrew_Id Varchar[11] Notnull Default,
Name Varchar[30] Notnull Default 'noname',
SSN Varchar[7] Notnull Default,
Airline_Id Varchar[5] Notnull,
Primary key (Aircrew_Id)
Foreign key (Airline_Id)
Reference Airline_Id to Airline (Airline_Id)
)

c) Create Table Pilot
((Pilot_Id Varchar[4] Notnull Default,
Flight rating integer Notnull Default 0,
Aircrew_Id Varchar[11] Notnull,
Primary key (Pilot_Id)

Foreign key (Aircrew_Id)
Reference Aircrew_Id to Aircrew (Aircrew_Id)
)

d) Create Table Flight Att
((Flight Att_Id integer Notnull Default 0,
Experience integer Notnull Default 0,
Date Date Notnull,
Pilot_Id Varchar[4] Notnull,
Copilot_Id Varchar[4] Notnull,
Primary key (Flight Att_Id),
Foreign key (Pilot_Id),
Foreign key (Copilot_Id),
Reference Pilot_Id to Pilot (Pilot_Id),
Reference Copilot_Id to Pilot (Pilot_Id),
)

e) Create Table Flight
((Flightt_Id integer Notnull Default 0,
Flight number integer Notnull Default 0,
Date date notnull
Pilot_Id Varchar[4] Notnull,
Copilot_Id Varchar[4] Notnull,
Airline_Id Varchar[5] Notnull,
Primary key (Flight_Id),
Foreign key (Airline_Id),
Foreign key (Pilot_Id),
Foreign key (Copilt_Id),
Reference Airline_Id to Airline(Airline_Id),
Reference Pilot_Id to Pilot(Pilot_Id),
Reference Copilot_Id to Pilot(Copilot_Id)
)

D. KESIMPULAN DAN SARAN

Algoritma Pemograman merupakan urutan langkah berhingga untuk memecahkan masalah logika atau matematika, Pembelajaran Algoritma Pemograman (AP) merupakan suatu hal yang penting dalam pemograman komputer. Karena pembuatan algoritma yang salah akan menyebabkan program memiliki unjuk kerja yang kurang baik.

Melalui Pembelajaran Algoritma Pemograman diharapkan dapat menemukan dan merekonstruksi konsep-konsep logika pemograman atau pengetahuan pemograman formal. Selanjutnya, diberi kesempatan menerapkan konsep-konsep logika pemograman untuk memecahkan masalah-masalah sehari-hari atau masalah dalam bidang lain. Dengan kata lain, pembelajaran algoritma pemograman berorientasi pada logika pengalaman sehari-hari, dan menerapkan algoritma pemograman dalam kehidupan sehari-hari, sehingga mahasiswa belajar dengan bermakna.

Pembelajaran algoritma pemograman berpusat pada mahasiswa, sedangkan dosen hanya sebagai fasilitator dan motivator, sehingga memerlukan paradigma yang berbeda tentang bagaimana mahasiswa belajar, bagaimana dosen mengajar, dan apa yang dipelajari oleh mahasiswa dengan paradigma pembelajaran pemograman selama ini. Karena itu, perubahan persepsi dosen tentang mengajar perlu dilakukan bila ingin mengimplementasikan pembelajaran algoritma pemograman.

Sebagai saran dari tulisan ini diharapkan kepada praktisi, pecinta, pengajar/dosen, pakar pemograman diharapkan untuk melakukan penelitian-penelitian yang berorientasi pada pembelajaran algoritma pemograman, serta mencoba mengimplementasikan pembelajaran algoritma pemograman secara bertahap, sehingga diperoleh hasil yang optimal.

DAFTAR PUSTAKA

Rumbaugh.(1991). Object Oriented Modelling and Design. Prentice Hall New Jersey, J.et al.

Blaha, Michael. (1994). Converting OO Model into RDBMS Schema. IEEE Software.

Roger S Pressman, Ph D (1992). Software Engineering. A Practitioner's Approach, Mc Graw-Hill Singapore.

Derek Coleman. (1994). Object Oriented Development The Fusion Method. Prentice Hall International Editions.

http://www.eg3.com/object.html

http://cuiwww.unige.ch/OSG/OOinfo/index.html

http:/www.soft-design.com/softinfo/objects.html

Adison Wesley C.J.Date. (1996). Relational Database Selected Writing. Mc Graw Hill.

Henry F Kort & Abraham S. (1996). Database Sistem Consepts. Mc Graw Hill.

Gio Wiederhold (1993). Database Design.Mc Graw Hill.
