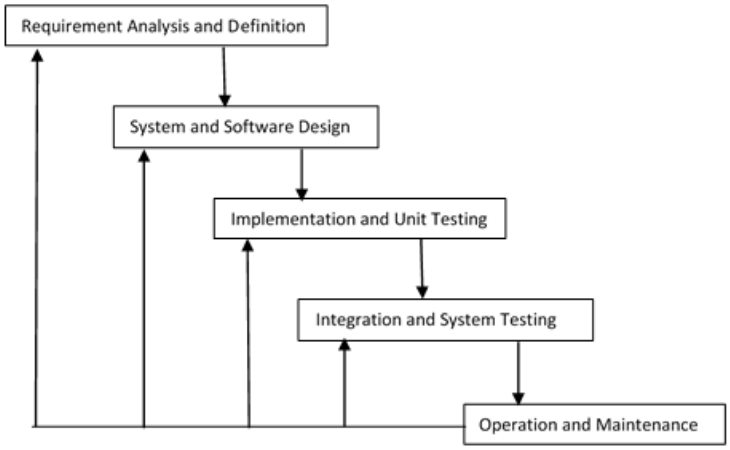
BAB 2
TINJAUAN PUSTAKA

2.1 Teori Umum

2.1.1 *Waterfall Software Development*

Tahapan utama dari *waterfall model* (Sommerville, 2011, pp. 30-31) langsung mencerminkan aktivitas pengembangan dasar. Terdapat 5 tahapan pada *waterfall model,* yaitu *requirement, analysis and definition, system and software design, implementation and unit testing, integration and system testing,* dan *operation and maintenance.*



Gambar 2. 1 Diagram Tahapan *Waterfall Model*

Berikut ini penjelasan tahapan-tahapan dari *waterfall model*:

1. ***Requirement Analysis and Definition***

   Merupakan tahapan penetapan fitur, analisa kebutuhan, kendala pembuatan dan tujuan sistem melalui konsultasi dengan pengguna sistem.

2. ***System and Software Design***

   Merupakan tahapan pembentukan arsitektur sistem berdasarkan persyaratan yang telah ditetapkan pada tahapan sebelumnya.

3. ***Implementation and Unit Testing***

Merupakan tahapan hasil dari desain perangkat lunak untuk direalisasikan sebagai satu set program atau unit program.

4. ***Integration and System Testing***

   Merupakan tahapan mengintegrasikan setiap unit program satu sama lain dan diuji sebagai satu sistem yang utuh untuk memastikan sistem sudah memenuhi persyaratan yang sudah ditetapkan.

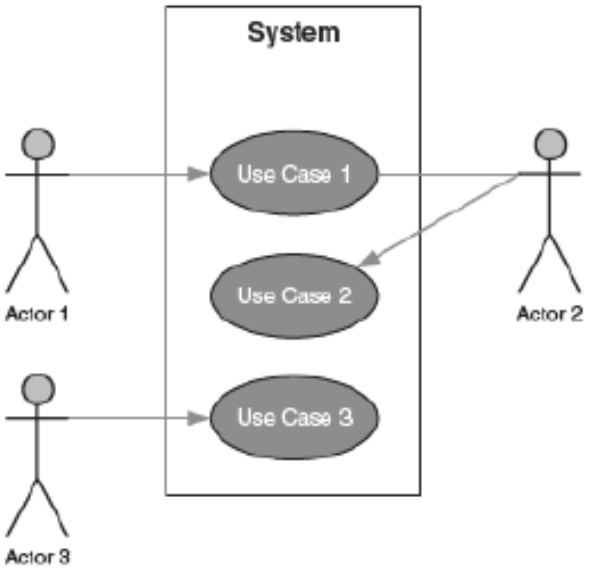
5. ***Operation and Maintenance***

   Merupakan tahapan instalasi dan penerapan sistem. Pada tahap ini juga dilakukan pengujian pada saat sistem dijalankan untuk menemukan dan memperbaiki *error* yang tidak ditemukan pada tahap pembuatan.

2.1.2 *Unified Modelling Language*

UML (*Unified Modeling Language*) adalah sebuah bahasa modeling tertentu yang digunakan untuk menentukan atau mendeskripsikan sebuah sistem *software* yang dipakai dalam suatu objek (Whitten & Bentley, 2007, p. 371).

2.1.2.1 Use Case Diagram



Gambar 2. 2 Contoh *Use Case Diagram*

(Sumber : Whitten & Bentley, 2007)

Use case diagram adalah diagram yang menggambarkan interaksi antara sistem, eksternal sistem dan *user*. *Use case* mendeskripsikan siapa yang akan menggunakan sistem dan dengan cara apa *user* dapat berinteraksi dengan sistem (Whitten & Bentley, 2007, hal. 246).

Use case modeling dapat digunakan untuk menentukan kebutuhan sistem dari pengguna dan perspektif *stakeholde*r, dengan menggunakan *use case modeling* mendorong keterlibatan pengguna yang sangat penting dalam mengembangkan atau membangun *software*. *Use case modeling* dapat dilakukan atau direpresentasikan dengan menggunakan *use case narrative/description* (Whitten & Bentley, 2007, hal. 246).

Berikut ini adalah komponen-komponen yang ada di dalam sebuah *use case diagram*:

a. Use Case

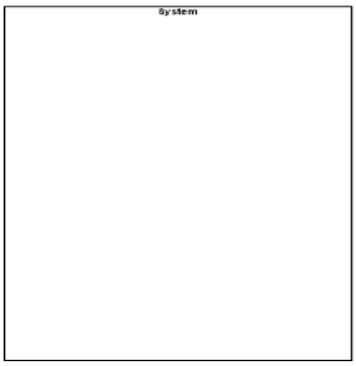
Use case merupakan deskripsi dari sistem berdasarkan prespektif *user* dengan menggunakan istilah yang mereka pahami. Sebuah *use case* mempresentasikan tujuan akhir dari sistem dan mendeskripsikan urutan aktivitas dan interaksi *user* untuk mencapai tujuan tersebut (Whitten & Bentley, 2007, hal. 246).

Gambar 2. 3 Simbol *Use Case* pada *Use Case Diagram*

(Sumber : Whitten & Bentley, 2007)

b. System Boundary

System Boundary menggambarkan interaksi-interaksi yang dilakukan oleh pengguna dalam suatu sistem.

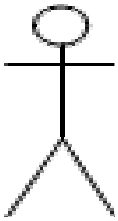


Gambar 2. 4 Simbol *System Boundary*

(Sumber : Whitten & Bentley, 2007)

c. Actor

Use case diinisialisasi atau dikendalikan oleh *user* yang disebut dengan *actor.* Sebuah *actor* mewakili sebuah atau beberapa interaksi yang dilakukan *user* terhadap sistem. *Actor* tidak selalu berupa manusia, tetapi dapat berupa suatu organisasi, sistem informasi, atau perangkat (Whitten & Bentley, 2007, hal. 247).



Gambar 2. 5 Simbol *Actor*

(Sumber : Whitten & Bentley, 2007)

d. Relationship

Relationship adalah suatu hubungan yang digambarkan sebagai garis antara dua simbol pada diagram *use case* (Whitten & Bentley, 2007, hal. 248). Bagaimana garis ditarik dan jenis simbol mereka terhubung akan menentukan makna yang berbeda. Ada 5 jenis hubungan yang memiliki arti yang berbeda, yaitu :

1. *Associations*

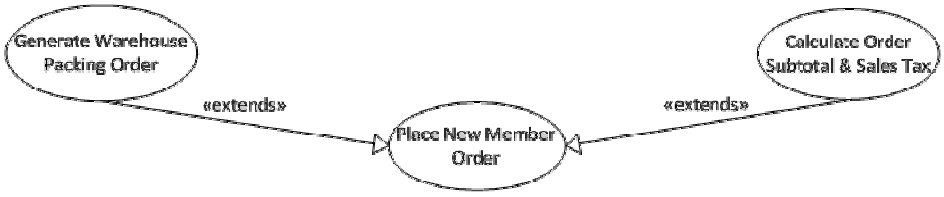
Hubungan antara aktor dan *use case* yang terjadi interaksi diantara keduanya. Pada gambar dibawah dapat dilihat bahwa *club member* melakukan proses ke dalam sistem dan kemudian direspon oleh *distribution center*.

Gambar 2. 6 Contoh Hubungan *Associations*

(Sumber : Whitten & Bentley, 2007)

2. *Extends*

Sebuah *use case* mungkin saja memiliki beberapa kegiatan lain yang membuat maksud dari *use case* tersebut sulit dimengerti. Oleh karena itu, untuk mempermudah pemahaman terhadap *use case* yang kompleks dibuatlah *extend association. Extend association* memberi penjelasan kepada *use case* tentang kegiatan lain yang ada di dalamnya atau hubungan yang terdiri dari langkah-langkah yang diekstraksi dari u*se case* yang lebih kompleks untuk menyederhanakan kasus awal dan memperluas fungsinya. Sebuah *extend association* ditandai dengan label <<extend>> pada garis hubungannya.

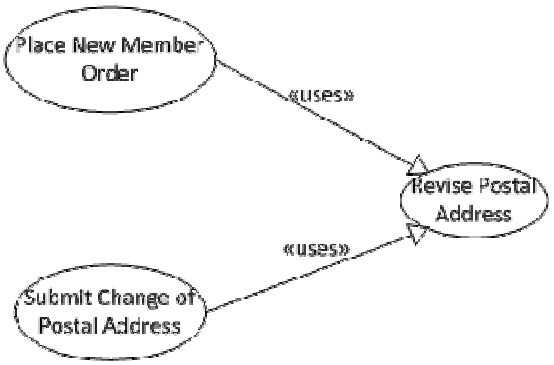


Gambar 2. 7 Contoh Hubungan *Extends*

(Sumber : Whitten & Bentley, 2007)

3. Uses (include)

Digunakan ketika terdapat dua atau lebih *use case* yang melakukan suatu langkah yang sama. Langkah tersebut lebih baik dipisahkan menjadi *use case* berbeda yang disebut *abstract use case* untuk mengurangi *redudancy* pada *use case.* Hubungan antara *use cass* dengan *abstract use case* ini disebut *include.* Sebuah *include relationship* ditandai dengan label <<uses>> pada garis hubungannya.

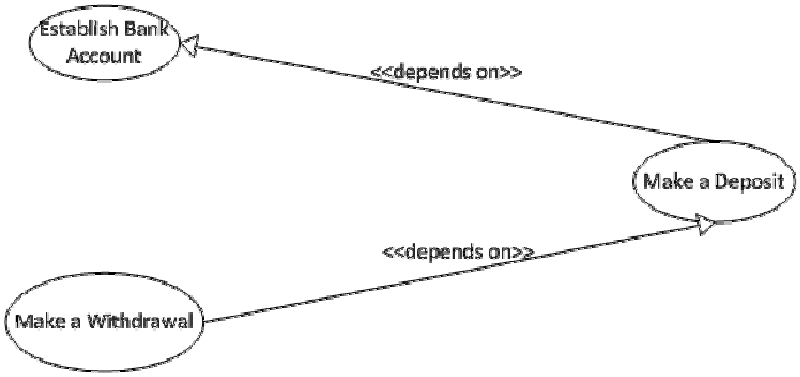


Gambar 2. 8 Contoh Hubungan *Uses*

(Sumber : Whitten & Bentley, 2007)

4. Depends on

Hubungan antara u*se case* yang menunjukkan bahwa satu *use case* tidak bisa dijalankan sampai u*se case* lain telah dijalankan. Hubungan *depends on* ditandai dengan label <<depends on>> pada garis hubungan.

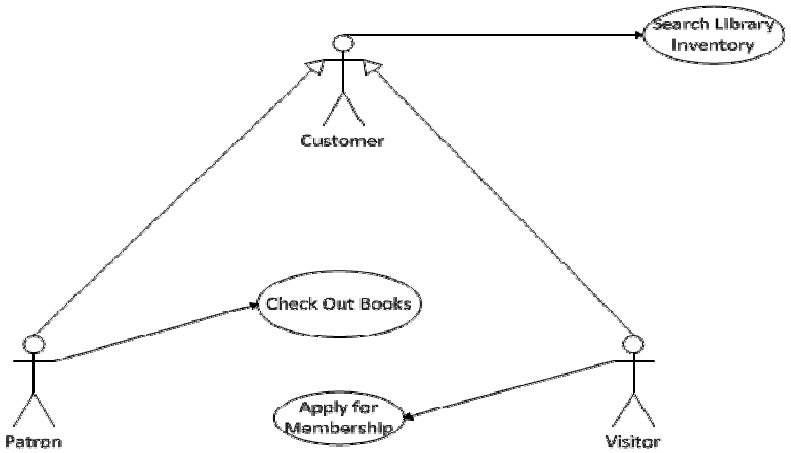


Gambar 2. 9 Contoh Hubungan *Depends on*

(Sumber : Whitten & Bentley, 2007)

5. Inheritence

Hubungan antara aktor-aktor yang dibuat untuk menyederhanakan gambar ketika seorang aktor abstrak mewarisi beberapa aktor nyata.



Gambar 2. 10 Contoh Hubungan *Inheritence*

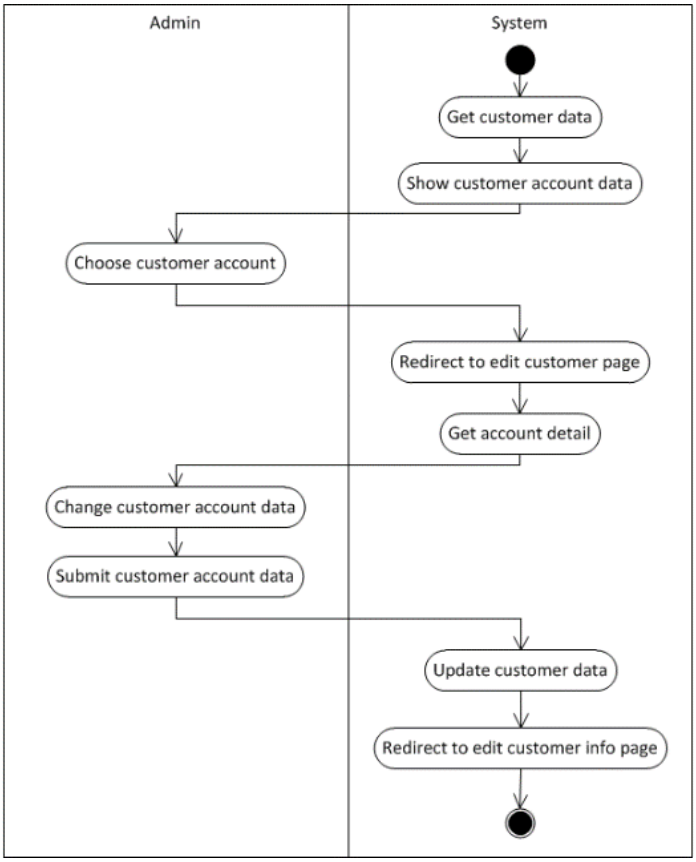
(Sumber : Whitten & Bentley, 2007)

2.1.2.2 Use Case Narrative

Use case narrative merupakan deskripsi tekstual dari setiap simbol *use case* yang menjelaskan kejadian dan bagaimana proses *user* ketika berinteraksi dengan sistem untuk menyelesaikan suatu tugas. Untuk mendapatkan pengertian dari kejadian suatu sistem, sangatlah baik untuk mendokumentasikan kejadian tersebut kedalam sebuah *use case* secara *narrative* (Whitten & Bentley, 2007, hal. 257).

2.1.2.3 Activity Diagram

Menurut (Whitten & Bentley, 2007, hal. 390), *activity* diagram adalah diagram yang digunakan untuk grafis menggambarkan aliran sekuensial kegiatan dari proses bisnis atau kasus penggunaan. Diagram ini sangat berguna untuk model tindakan yang akan dilakukan ketika operasi mengeksekusi dan menunjukkan hasil tindakan tersebut. Satu *use case* dapat membangun satu atau lebih *activity* diagram, tergantung pada seberapa kompleksnya use case tersebut.



Gambar 2. 11 Contoh *Activity Diagram*

(Sumber : Whitten & Bentley, 2007)

Ada 8 notasi dasar *activity diagram*, yaitu :

1. ***Initial node,*** di wakili oleh lingkaran yang solid. Simbol bahwa ini berarti awal proses.

Gambar 2. 12 Simbol *Initial Node*

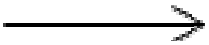
(Sumber : Whitten & Bentley, 2007)

2. ***Action*****,** diwakili oleh persegi panjang bulat. Tindakan ini berarti langkah-langkah individu.

Gambar 2. 13 Simbol *Action*

(Sumber : Whitten & Bentley, 2007)

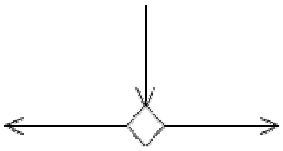
3. *Flow,* diwakili oleh panah pada diagram, menunjukkan perkembangan melalui tindakan. *Flow* perlu kata-kata untuk mengidentifikasi mereka, tetapi kebanyakan mereka tidak perlu kata-kata untuk mengidentifikasi mereka.



Gambar 2. 14 Simbol *Flow*

(Sumber : Whitten & Bentley, 2007)

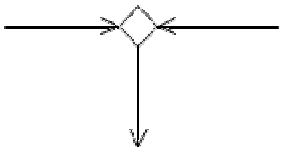
4. *Decision*, diwakili oleh bentuk *diamond* dengan satu aliran masuk dan dua atau lebih arus keluar. Datang keluar arus ditandai untuk menunjukkan kondisi.



Gambar 2. 15 Simbol *Decision*

(Sumber : Whitten & Bentley, 2007)

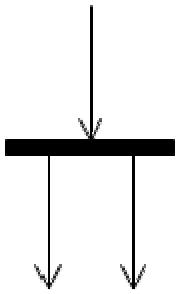
5. *Merge,* diwakili oleh bentuk *diamond* dengan dua atau lebih arus masuk dan satu aliran keluar. *Merge* ini menggabungkan aliran yang dipisahkan oleh keputusan.



Gambar 2. 16 Simbol *Merge*

(Sumber : Whitten & Bentley, 2007)

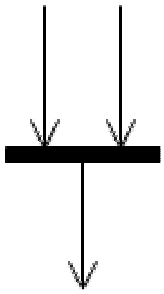
6. *Fork,* diwakili oleh bar hitam dengan satu aliran masuk dan dua atau lebih arus keluar. *Fork* ini menunjukkan tindakan paralel yang terjadi secara bersamaan atau berjalan bersama-sama.



Gambar 2. 17 Simbol *Fork*

(Sumber : Whitten & Bentley, 2007)

7. ***Join,*** diwakili oleh bar hitam dengan dua atau lebih arus masuk dan satu aliran keluar. *Join* ini menunjukkan akhir yang dipisahkan oleh *fork*. Semua *action* yang masuk ke *join* harus diselesaikan sebelum pengolahan berlanjut.



Gambar 2. 18 Simbol *Join*

(Sumber : Whitten & Bentley, 2007)

8. ***Activity final,*** diwakili oleh lingkaran yang solid dalam lingkaran berlubang, menunjukkan akhir dari proses.

Gambar 2. 19 Simbol *Activity Final*

(Sumber : Whitten & Bentley, 2007)

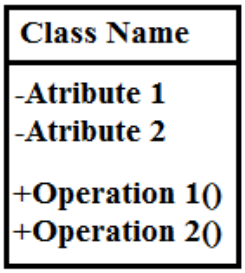
2.1.2.4 Class Diagram

Menurut (Whitten & Bentley, 2007, hal. 373) *Class Diagram* merupakan gambaran mengenai struktur objek yang membentuk sistem berupa *class* dan juga menunjukan relasi antar *class.*

A. Class

Class merupakan kumpulan *attribute* dan *behavior* yang mempresentasikan suatu objek. *Class* sebuah kotak yang terbagi menjadi tiga bagian, yaitu:

1. *Name* adalah nama dari suatu *class*.
2. *Attributes* adalah karakteristik dari suatu entitas.
3. *Method* adalah logika *software*yang dieksekusi untuk memberikan respon pada suatu pesan.



Gambar 2. 20 Struktur *Class Diagram*

B. Visibility

Tingkatan akses *attributes* dan method dalam suatu *class* (Whitten & Bentley, 2007, hal. 650). Tipe-tipe *visibility* terbagi menjadi tiga, yaitu :

1. ***Public***

   Attributes dan *methods* dapat diakses dan dipanggil oleh semua *class*. Dinotasikan dengan simbol “+”.

2. ***Protected***

   Attributes dan *methods* dapat diakses dan dipanggil oleh *subclass* dari *class* itu sendiri. Dinotasikan dengan simbol “#”.

3. Private

Attributes dan *methods* dapat diakses dan dipanggil hanya *class* itu sendiri. Dinotasikan dengan simbol "-".

C. Multiplicity

Dalam *class diagram,* antar *class* memiliki hubungan yang dilengkapi dengan *multiplicity*, yaitu jumlah minimal dan maksimal suatu objek *class* yang berhubungan dengan objek *class* lainnya, terdiri dari (Whitten & Bentley, 2007, hal. 377) :

Tabel 2. 1 Lima Macam Hubungan *Multiplicity*

Multiplicity	**Deskripsi**
1	Hanya satu
0..1	Nol atau satu
0..*	Nol atau lebih
1..*	Satu atau lebih
m..n	Rentang spesifik

D. Relationship

Hubungan antar class terbagi menjadi beberapa jenis, antara lain :

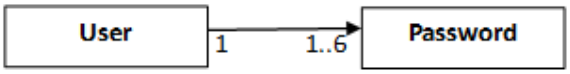
1. *Association*

Berupa sebuah garis yang menghubungkan hubungan antar *class* yang saling berinteraksi. Garis ini melambangkan tipe dari hubungan antar *class.*

Ada dua jenis hubungan asosiasi, yaitu:

a. *Uni-directional*

Hubungan antara dua *class* yang saling terhubung, tetapi hanya satu yang memiliki peran untuk dapat mengirim pesan ke *class* lainnya yang ditunjuk. Hubungan *uni-directional* direpresentasikan dengan menggunakan anak panah.



Gambar 2. 21 Contoh *Uni-Directional*

b. *Bi-Directional*

Hubungan antara dua class yang saling terhubung dan keduanya memiliki peran untuk dapat mengirim pesan ke class lainnya yang terhubung. Hubungan *bi-directorinal* dipresentasikan dengan garis tanpa anak panah.

Gambar 2. 22 Contoh *Bi-directional*

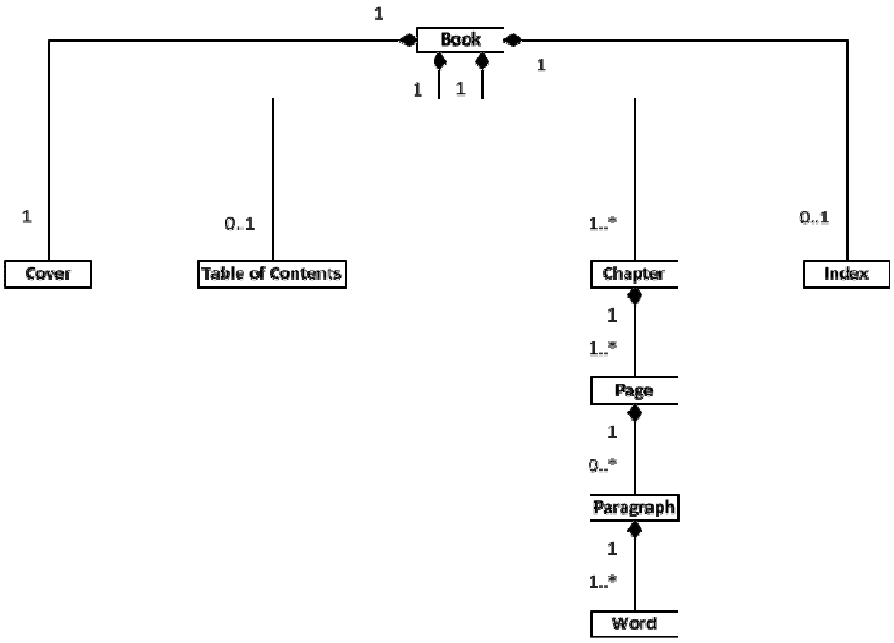
2. *Aggregation*

Berupa garis dengan ujung berbentuk wajik tidak berisi. Digunakan untuk mengindikasikan keseluruhan bagian hubungan antar *class,* biasa sering disebut relasi.

Gambar 2. 23 Contoh *Aggregation*

3. *Composition*

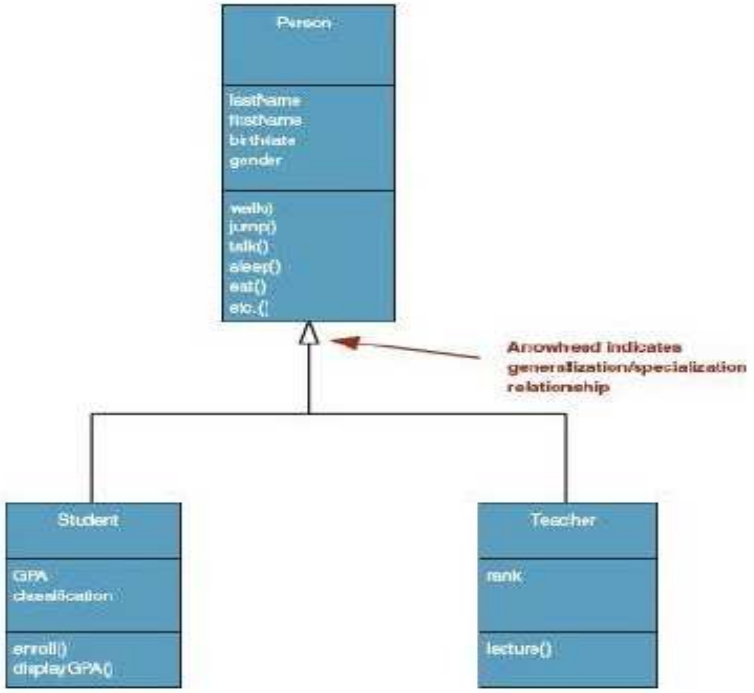
Berupa garis dengan ujung berbentuk wajik berisi. *Composition* digunakan apabila ada sebuah *class* yang tidak dapat berdiri sendiri dan merupakan bagian dari *class* lainnya.



Gambar 2. 24 Contoh *Composition*

4. *Inheritance*

Konsep dimana *method* dan *attribute* yang berada di dalam sebuah *class* bisa diturunkan dan digunakan di *class* lain.

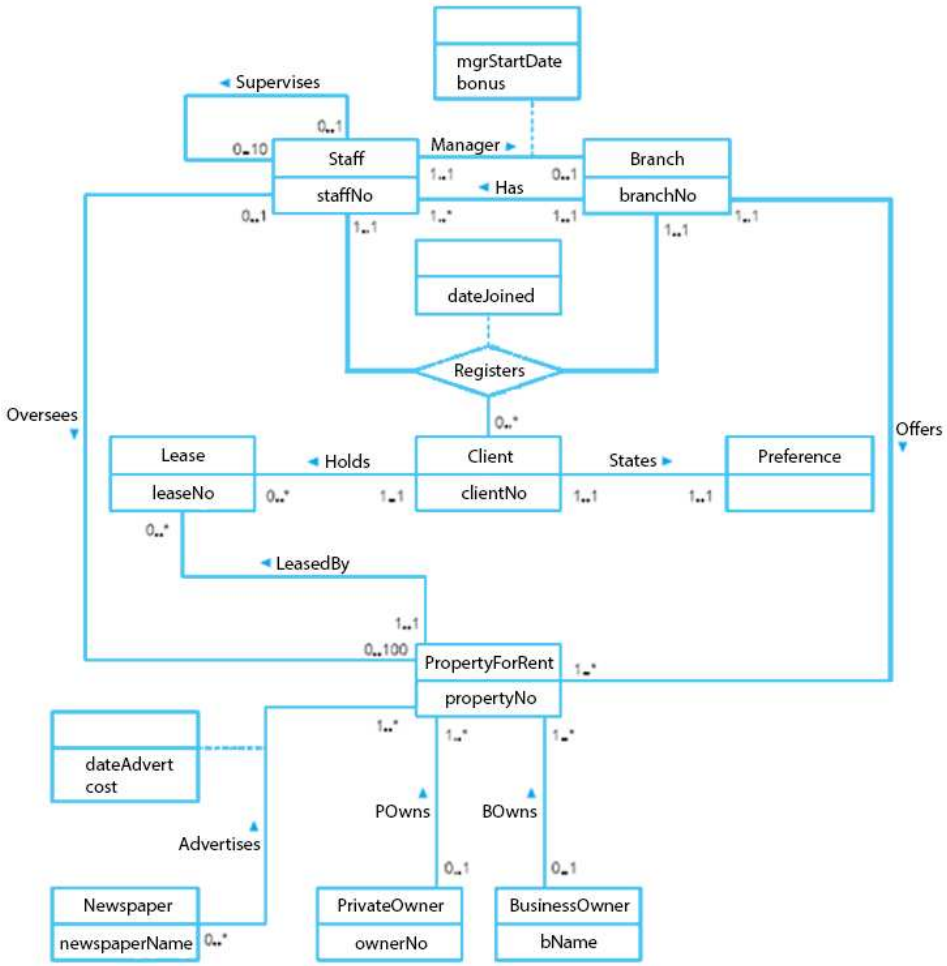


Gambar 2. 25 Contoh *Inheritance*

(Sumber : Whitten & Bentley, 2007)

2.1.3 *Entity Relationship Modeling (ER Model)*

Konsep dasar dari *ER model* adalah jenis *entity* yang mewakili kelompok dari "objek" di "dunia nyata" dengan sifat yang sama. Suatu entitas memiliki keberadaan yang independen dan keberadaan objek dengan fisik (atau "nyata") atau keberadaan objek dengan konseptual (atau "abstrak"). Tipe *entity* sendiri merupakan sekelompok benda dengan sifat yang sama, yang diidentifikasi oleh perusahaan sebagai keberadaan yang bebas (Connolly & Begg, 2015, pp. 406-407).



Gambar 2. 26 Contoh *Entity Relationship* (ER) Driagram

(Sumber : Connolly & Begg, 2010)

2.1.4 Interaksi Manusia dan Komputer

Interaksi manusia dan komputer merupakan penerapan metode eksperimen psikologi ke dalam perangkat yang dikembangkan dalam ilmu komputer (Shneiderman & Plaisant, 2010, hal. 22).

2.1.4.1 *Eight Golden Rule*

Menurut (Shneiderman & Plaisant, 2010, p. 88), terdapat *eight golden rule* atau delapan aturan emas di dalam interaksi manusia dan komputer. Aturan-aturan tersebut digunakan sebagai panduan dalam merancang aturan antar muka *user*.

Delapan aturan emas tersebut antara lain:

1. **Jaga konsistensi**

   Konsisten dengan urutan tindakan dan *user interface* sehingga pengguna dapat menggunakan aplikasi ini lebih mudah dan mahir dengan cepat.

2. **Memenuhi kegunaan secara *universal***

   Tahu kebutuhan dari pengguna yang berbeda dan desain konten dinamis.

3. **Memberikan umpan balik**

   Berikan tanggapan ketika pengguna melakukan sesuatu. Contoh: memberikan kotak dialog dan suara peringatan saat pengguna membuat kesalahan.

4. **Desain dialog yang menghasilkan penutupan**

   Urutan tindakan harus diatur untuk beberapa bagian seperti memulai, konten, dan finishing bagian. Dari mereka atau aplikasi, memberikan umpan balik dari tindakan sehingga pengguna dapat melanjutkan.

5. **Cegah kesalahan**

   Sistem harus mampu mendeteksi kesalahan sehingga pengguna tidak menciptakan kesalahan fatal dan memberikan solusi untuk kesalahan.

6. **Mengizinkan tindakan pembalikan yang mudah**

   Sebuah fitur yang memungkinkan pengguna untuk kembali ke tindakan sebelumnya untuk menghilangkan kecemasan ketika pengguna membuat kesalahan.

7. **Mendukung *internallokus of control***

   Pengguna ahli biasanya ingin mengontrol sistem dan bukan sebaliknya sehingga sistem ini lebih baik dirancang dengan pengguna sebagai inisiator dalam pikiran.

8. **Mengurangi beban ingatan jangka pendek**

   Pengguna terbatas kemampuan dalam memproses informasi dan mengingatnya dalam jangka pendek sehingga aplikasi tidak harus membuat pengguna mengingat informasi dalam layar sebelumnya.

2.1.4.2 Lima Faktor Manusia Terukur

Menurut (Shneiderman & Plaisant, 2010, hal. 23), terdapat lima faktor yang menjadi pusat evaluasi yang berguna untuk langkah-langkah dan tujuan *usability* yaitu :

1. ***Time to Learn***

   Lama waktu yang dipakai oleh pengguna untuk belajar penggunaan sistem.

2. ***Speed of Perfomance***

   Faktor yang digunakan untuk mengukur waktu yang diperlukan pengguna untuk menyelesaikan suatu tugas.

3. ***Rate of Error by Users***

   Tingkat kesalahan pengguna dalam penggunaan aplikasi, waktu yang terpakai untuk memperbaiki kesalahan akan berdampak pada kinerja dan kecepatan sistem.

4. ***Retention Over Time***

   Tingkat daya ingat pengguna seiring penggunaan aplikasi dalam jangka waktu tertentu. Daya ingat berpengaruh pada lama waktu belajar pengguna.

5. ***Subjective Satisfaction***

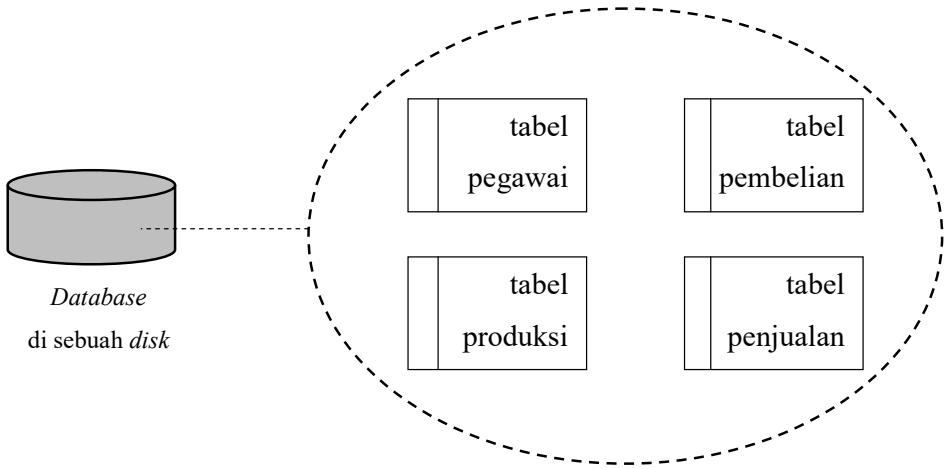
   Tingkat kepuasan pengguna atas penggunaan aplikasi dalam berbagai aspek sistem.

2.2 Teori Khusus

2.2.1 Database

Database dapat dibayangkan sebagai sebuah lemari arsip. Himpunan kelompok data (arsip) yang saling berhubungan sedemikian rupa agar dapat dimanfaatkan kembali dengan cepat dan mudah (Fathansyah, 2015, hal. 2).

Database dan lemari arsip sesungguhnya memiliki prinsip kerja dan tujuan yang sama. Prinsip utamanya adalah pengaturan data/arsip. Dan tujuan utamanya adalah kemudahan dan kecepatan dalam pengambilan kembali data/arsip. Perbedaannya hanya terletak pada media penyimpanan yang digunakan. Jika lemari arsip menggunakan lemari dari besi atau kayu sebagai media penyimpanan, maka basis data menggunakan media penyimpanan elektronik seperti cakram magnetis (*magnetic disk* atau disingkat sebagai *disk* saja). Perbedaan media ini yang selanjutnya melahirkan perbedaan – perbedaan lain yang menyangkut jumlah dan jenis metode yang dapat digunakan dalam upaya penyimpanan (Fathansyah, 2015, hal. 3).



Gambar 2. 27 Contoh *Database*

Database hanyalah sebuah objek yang pasif. *Database* ada karena ada pembuatnya. *Database* tidak akan pernah beguna jika tidak ada pengelola dan penggeraknya. Pengelola atau penggeraknya secara langsung adalah

program/aplikasi dan gabungan keduanya menghasilkan sebuah sistem. Secara umum sebuah sistem *database* merupakan sistem yang terdiri atas kumpulan tabel data yang saling berhubungan dan sekumpulan program yang memungkinkan beberapa pemakai atau program lain untuk mengakses dan memanipulasi tabel – tabel tersebut (Fathansyah, 2015, hal. 12).

Komponen–komponen utama pada sistem *database* menurut (Fathansyah, 2015, hal. 5), sebagai berikut :

1. *Hardware*
2. *Operating system*
3. *Database*
4. Sistem pengelola basis data (DBMS)
5. *User*
6. Aplikasi lain

2.2.1.1 Operasi Dasar *Database*

Menurut (Fathansyah, 2015, hal. 5), operasi–operasi dasar yang dapat dilakukan pada *database* meliputi :

1. ***Create database,*** pembuatan *database* yang baru.
2. ***Drop database,*** penghapusan *database.*
3. ***Create tabel,*** pembuatan tabel baru ke *database.*
4. ***Drop tabel,*** penghapusan tabel dari *database.*
5. ***Query,*** pengambilan data dari sebuah tabel.
6. ***Update,*** pengubahan data dari sebuah tabel.
7. ***Delete,*** penghapusan data dari sebuah tabel.

2.2.1.2 Tujuan Database

Menurut (Fathansyah, 2015, hal. 6-7), pemanfaatan *database* dilakukan untuk memenuhi sejumlah tujuan seperti berikut ini :

1. ***Speed***

   Pemanfaatan *database* memungkinkan untuk dapat menyimpan data atau melakukan perubahan/manipulasi terhadap data atau menampilkan kembali data tersebut dengan lebih cepat dan mudah.

2. ***Space***

   Karena keterkaitan yang erat antar kelompok data dalam sebuah *database,* maka redudansi (pengulangan) data pasti akan selalu ada. Banyaknya redudansi ini tentu akan memperbesar ruang penyimpanan yang harus disediakan. Dengan *database,* efisiensi/optimalisasi pengguaan ruang penyimpanan dapat dilakukan, karena dapat melakukan penekanan jumlah redudansi data, baik dengan menerapkan sejumlah pengodean atau dengan membuat relasi–relasi antar kelompok data yang saling berhubungan.

3. ***Accuracy***

   Pemanfaatan pengodean atau pembentukan relasi antar data bersama dengan penerapan aturan/batasan tipe data, *domain* data, keunikan data, dan sebagainya, yang secara ketat dapat diterapkan dalam sebuah *database,* sangat berguna untuk menekan ketidak akuratan penyimpanan data.

4. ***Availability***

   Pertumbuhan data sejalan dengan waktu akan semakin membutuhkan ruang penyimpanan yang besar, padahal tidak semua data itu selalu di butuhkan. Karena itu, dapat memilah adanya data utama/master/referensi, data transaksi, data histori hingga data yang kadarluasa. Data yang sudah jarang atau bahkan tidak pernah lagi digunakan, dapat di atur untuk dilepaskan dari sistem *database* yang sedang aktif baik dengan cara penghapusan atau dengan memindahkannya ke media penyimpanan *off-line.*

5. ***Completeness***

   Lengkap atau tidaknya data yang dikelola dalam sebuah *database* bersifat relatif. Seorang pemakai mungkin sudah menganggap bahwa data yang dikelola sudah lengkap, tetapi pemakai yang lain belum tentu berpendapat sama. Untuk mengakomodasi kebutuhan kelengkapan data yang semakin berkembang, maka tidak hanya dapat menambah *record-record* data, tetapi juga dapat melakukan perubahan struktur dalam

database, baik dalam bentuk penambahan objek/tabel baru atau dengan penambahan *field-field* baru pada suatu tabel.

6. ***Security***

   Ada sejumlah sistem aplikasi pengelola *database* yang tidak menerapkan aspek keamaan dalam penggunaan *database.* Akan tetapi untuk sistem yang besar dan serius, aspek keamanan juga dapat diterapkan dengan tepat. Dapat menentukan siapa–siapa yang boleh menggunakan *database* beserta objek–objek di dalamnya dan menentukan jenis–jenis operasi apa saja yang boleh dilakukannya.

7. ***Sharebility***

   Pemakai *database* seringkali tidak terbatas pada satu pemakai saja, atau di satu lokasi saja atau oleh satu sistem/aplikasi saja. *Database* yang dikelola oleh sistem yang mendukung *multi-user,* akan dapat memenuhi kebutuhan, tetapi tetap dengan menjaga/menghindari munculnya persoalan baru seperti inkonsistensi data (data yang sama diubah oleh banyak pemakai pada saat yang bersamaan) atau kondisi *deadlock* (ada banyak pemakai yang saling menunggu untuk menggunakan data).

2.2.2 Metodologi Perancangan Sistem Basis Data

Metodologi perancangan adalah sebuah pendekatan yang menggunakan bantuan prosedur, teknik, alat dan dokumentasi secara terstruktur untuk mendukung dan memudahkan proses *design*. Metodologi *design* juga membantu perancang untuk merencanakan, mengatur dan mengevaluasi laporan pengembangan sistem basis data.

Proses metodologi perancangan dibagi menjadi 3; *conceptual, logical* dan *physical design*. *Conceptual design* adalah sebuah proses konstruksi data model dari seluruh data yang digunakan oleh seluruh proses bisnis dari suatu sistem. Tahap *logical design* adalah proses konstruksi data model dari data yang digunakan oleh seluruh proses bisnis dan akan dirancang dalam sebuah model data tertentu. Sedangkan *physical design* adalah proses menghasilkan deskripsi dari implementasi sistem basis data pada penyimpanan sekunder. Proses *physical*

design menggunakan relasi data, penyusunan *file*, index, pengamanan serta batasan integritas untuk menghasilkan efisiensi akses ke data yang maksimal.

2.2.3.1 Conceptual Design

Tujuan dari *conceptual design* adalah untuk membuat model data konseptual yang dibutuhkan oleh pengguna. Langkah-langkah dalam *conceptual design* adalah mengidentifikasi entitas, mengidentifikasi tipe relasi, mengidentifikasi dan menghubungkan atribut dengan entitas atau tipe relasi, menentukan domain atribut, menentukan atribut *candidate, primary dan alternate key*, memeriksa redundansi serta validasi model data dengan persyaratan dari pengguna.

2.2.3.2 Logical Design

Tujuan dari *logical design* adalah untuk menterjemahkan model data *conceptual* ke dalam model data *logical* yang sudah ditentukan di persyaratan dari pengguna. Langkah-langkah dalam menghasilkan model data *logical* adalah mendapatkan relasi dari entitas yang sudah ada di model data *conceptual*, relasi bisa terbagi menjadi banyak kategori, kategori tersebut mencakup relasi *one-to-many* (1:*), relasi *one-to-one* (1:1), *many-to-many* (*:*), relasi *superclass/subclass* dan atribut *multi-valued*.

Langkah selanjutnya untuk menciptakan model data *logical* adalah validasi relasi menggunakan normalisasi, validasi relasi dengan proses transaksi pengguna di scenario nyata, pengecekan batasan integritas dan melakukan tinjauan kembali dengan sang pengguna.

2.2.3.3 Physical Design

Seperti yang sudah dijelaskan diatas, tujuan dari *physical design* adalah untuk menghasilkan deskripsi dari implementasi sistem basis data pada penyimpanan sekunder serta mencapai efisiensi dan efektivitas dari akses data dengan menggunakan relasi data, penyusunan *file*, index, pengamanan serta batasan integritas.

Langkah-langkah yang harus dilakukan untuk menciptakan sebuah model data *physical* adalah menerjemahkan model data *logical* ke dalam DBMS yang sudah ditentukan, hal ini bisa dicapai dengan merancang relasi

dasar, merancang representasi dari data yang dimaksud serta merancang batasan umum dari DBMS tersebut. Lalu, perancang harus merancang indeks dan penyusunan data dengan menganalisa transaksi yang dilakukan oleh pengguna, memilih cara penyusunan data dan indeks serta memperkirakan keperluan ukuran tempat penyimpanan.

Langkah selanjutnya adalah merancang *user view*, merancang mekanisme pengamanan, melakukan pengaturan redundancy dalam sistem serta memantau dan mengatur sistem operasi yang dibutuhkan untuk menjalankan sistem.

2.2.3 Normalisasi

Menurut (Connolly & Begg, 2015), normalisasi adalah sebuah cara untuk menghasilkan sebuah hubungan antar entitas dengan sifat yang sudah tertera di persyaratan aplikasi yang diminta oleh pengguna. Karakteristik dari relasi antar entitas tersebut harus memenuhi syarat-syarat tertentu, syarat-syarat tersebut adalah :

- Harus memenuhi jumlah atribut minimal yang diperlukan untuk mendukung persyaratan data dari pengguna
- Atribut dengan relasi *logical* yang dekat harus dalam satu relasi.
- *Redundancy* seminimal mungkin, dengan setiap atribut hanya direpresentasikan sekali saja dalam sistem. Pengecualian terjadi jika ada atribut yang membentuk seluruh atau sebagian dari sebuah *foreign key*.

Bentuk-bentuk normalisasi menurut (Connolly & Begg, 2015, pp. 466-472) antara lain:

1. ***Unnormalized Form* (UNF)**

   Bentuk ini merupakan bentuk awal dari kumpulan tabel-tabel yang masih memiliki satu kumpulan data yang berulang atau lebih. Cara membuat bentuk UNF adalah dengan memindahkan data dari sumber informasi yang ke dalam tabel dengan format baris dan kolom.

2. ***First Normal Form* (1NF)**

   Merupakan sebuah bentuk normalisasi tahap pertama dimana tidak ada lagi data yang berjumlah lebih dari satu dalam kumpulan data tersebut.

3. ***Second Normal Form*** **(2NF)**

Merupakan bentuk dimana tabel-tabel dalam bentuk 1NF sudah memiliki sebuah *primary key* dan data-data yang bukan merupakan *primary key* bergantung pada *primary key* tersebut.

4. ***Third Normal Form*** **(3NF)**

Merupakan bentuk dimana tabel-tabel pada bentuk 2NF sudah tidak memiliki relasi yang bersifat *transitive* serta tidak ada atribut yang bukan bersifat *primary key* bergantung pada *primary key*.