BAB II

TINJAUAN PUSTAKA

II.1. *Data mining*

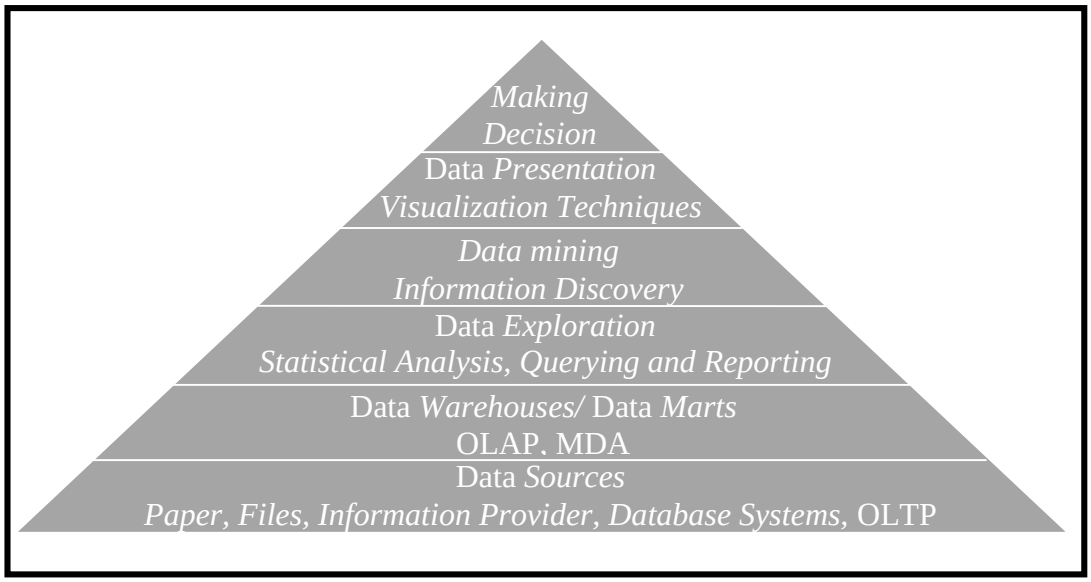
Data mining adalah proses yang mempekerjakan satu atau lebih teknik pembelajaran komputer (*machine learning*) untuk menganalisis dan mengekstraksi pengetahuan (*knowledge*) secara otomatis. Definisi lain diantaranya adalah pembelajaran berbasis induksi (*induction-based learning*) adalah proses pembentukan definisi-definisi konsep umum yang dilakukan dengan cara mengobservasi contoh-contoh spesifik dari konsep-konsep yang akan dipelajari. (Hermawati, 2009: 3).

Knowledge Discovery in Databases (KDD) adalah kegiatan yang meliputi pengumpulan data, pemakaian data historis untuk menentukan keteraturan, pola atau hubungan dalam data berukuran besar. Keluaran *data mining* ini bisa dipakai untuk membantu pengambilan keputusan di masa mendatang. (Fitri Nurchalifatun, 2015:2).

Data mining merupakan proses iteratif dan interaktif untuk menemukan pola atau model baru yang sahih (sempurna), bermanfaat dan dapat dimengerti dalam suatu *database* yang sangat besar (*massive database*).

1. *Sahih*, yaitu dapat digeneralisasi untuk masa yang akan datang.
2. Baru, yaitu apa yang sedang tidak diketahui.
3. Bermanfaat, yaitu dapat digunakan untuk melakukan suatu tindakan.
4. Iteratif, memerlukan sejumlah proses yang diulang.
5. Interaktif, memerlukan interaksi manusia dalam prosesnya.

Data mining berisi pencarian *trend* atau pola yang diinginkan dalam *database* besar untuk membantu pengambilan keputusan di waktu yang akan datang. Pola-pola ini dikenali oleh perangkat tertentu yang dapat memberikan suatu analisa data yang berguna dan berwawasan yang kemudian dapat dipelajari dengan lebih teliti, yang mungkin saja menggunakan perangkat pendukung keputusan yang lainnya.



Gambar II.1 *Data mining* dan Teknologi *Database* Lainnya
(Sumber : Hermawati ; 2009: 4)

Dari gambar di atas terlihat bahwa teknologi data *warehouse* digunakan untuk melakukan OLAP (*Online Analytic Processing*), sedangkan *data mining* digunakan untuk melakukan *information discovery* yang informasinya lebih ditujukan untuk seorang *Data Analyst* dan *Business Analyst* (dengan ditambah visualisasi tentunya). Dalam prakteknya, *data mining* juga mengambil data dari data *warehouse*. Hanya saja aplikasi dari *data mining* lebih khusus dan lebih spesifik dibandingkan OLAP mengingat *database* bukan satu-satunya bidang ilmu yang mempengaruhi *data mining*, banyak lagi bidang ilmu yang turut

memperkaya *data mining* seperti : *information science* (ilmu informasi), *high performance computing*, visualisasi, *machine learning*, statistik, *neural network (*jaringan syaraf tiruan), pemodelan matematika, *information retrieval* dan *information extraction* serta pengenalan pola. Bahkan pengolahan citra (*image processing*) juga digunakan dalam rangka melakukan *data mining* terhadap data *image/ spatial.*

Alasan mengapa melakukan *data mining* dari sudut pandang komersial karena :

1. Meledaknya volume data yang dihimpun dan disimpan dalam data *warehouse* seperti data *web*, *e-commerce*, penjualan di *departement store*, transaksi bank/ *credit card*.
2. Proses komputasi yang dapat diupayakan.
3. Kuatnya tekanan kompetitif untuk dapat menyediakan yang lebih baik, layanan-layanan *custom*-isasi dan informasi sedang menjadi produk yang berarti.

II.1.1. Operasi *Data mining*

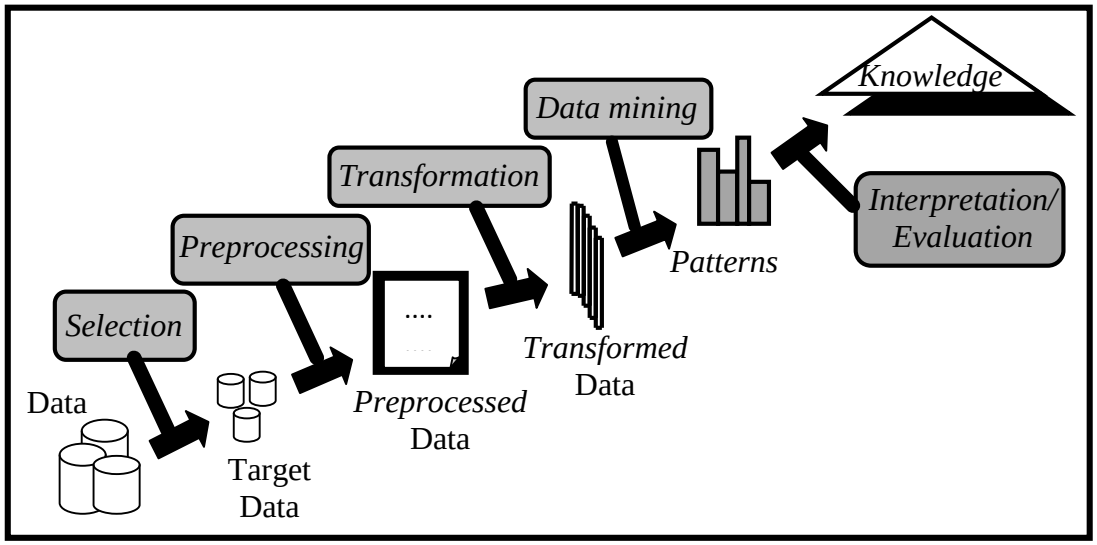
Operasi *data mining* menurut sifatnya dibedakan menjadi dua yaitu :

1. Prediksi (*prediction driven*), untuk menjawab pertanyaan apa dan sesuatu yang bersifat transparan. Operasi prediksi digunakan untuk validasi hipotesis, *querying* dan pelaporan, analisis multidimensi, OLAP (*Online Analytic Processing*) serta analisis statistik.
2. Penemuan (*discovery driven*) bersifat transparan dan untuk menjawab pertanyaan “mengapa?”. Operasi penemuan digunakan untuk analisis data

eksplorasi, pemodelan prediktif, segmentasi *database*, analisis keterkaitan (*link analysis*) dan deteksi deviasi.

Tahapan proses dalam penggunaan *data mining* yang merupakan proses *Knowledge Discovery in Databases* (KDD) seperti yang terlihat pada gambar 2.2 dapat diuraikan sebagai berikut :

1. Memahami *domain* aplikasi untuk mengetahui dan menggali pengetahuan awal serta apa sasaran pengguna.
2. Membuat target data-set yang meliputi pemilihan data dan fokus pada sub-set data.
3. Pembersihan dan transformasi data meliputi eliminasi derau, *outliers*, *missing value* serta pemilihan fitur dan reduksi dimensi.
4. Penggunaan *algoritma data mining* yang terdiri dari asosiasi, sekuensial, klasifikasi, klasterisasi, dll.
5. Interpretasi, evaluasi dan visualisasi pola untuk melihat apakah ada sesuatu yang baru dan menarik dan dilakukan iterasi jika diperlukan.



Gambar II.2 Proses KDD
(Sumber : Hermawati ; 2009: 6)

II.1.2. Tantangan Dalam *Data mining*

Tantangan dalam *data mining* meliputi :

1. *Scalability*, yaitu besarnya ukuran *basis data* yang digunakan.
2. *Dimensionality*, yaitu banyaknya jumlah *atribut* dalam data yang akan diproses.
3. *Complex and Heterogeneous Data*, yaitu data yang kompleks dan mempunyai variasi yang beragam.
4. Data *Quality*, kualitas data yang akan diproses seperti data yang bersih dari *noise*, *missing value*.
5. Data *Ownership and Distribution*, yaitu siapa yang memiliki data dan bagaimana distribusinya.
6. *Privacy Preservation*, yaitu menjaga kerahasiaan data yang banyak diterapkan pada data nasabah perbankan.
7. *Streaming Data*, yaitu aliran data itu sendiri. (Hermawati ; 2009:19).

II.1.3. Teknik *Data mining*

Beberapa teknik dan sifat *data mining* adalah sebagai berikut :

1. *Classification*
2. *Clustering*
3. *Association Rule Discovery*
4. *Sequential Pattern Discovery*
5. *Regression*

II.2. *Algoritma Apriori*

Algoritma apriori termasuk jenis aturan asosiasi pada data *mining*. Aturan yang menyatakan asosiasi antara beberapa atribut sering disebut *affinity analysis* atau *market basket analysis*. Analisis asosiasi atau *association rule mining* adalah teknik *data mining* untuk menemukan aturan suatu kombinasi *item.* Contoh aturan asosiatif dari analisis pembelian di suatu pasar swalayan adalah dapat diketahuinya berapa besar kemungkinan seorang pelanggan membeli roti bersamaan dengan susu. (Kusrini, dkk, 2009: 149).

Penting tidaknya suatu asosiasi dapat diketahui dengan dua tolak ukur, yaitu *support* dan *confidence*. *Support* (nilai penunjang) adalah persentase kombinasi *item* tersebut dalam *database*, sedangkan *confidence* (nilai kepastian) adalah kuatnya hubungan antar *item* dalam aturan asosiasi. (Heroe Santoso, dkk, 2016: 20).

Proses utama yang dilakukan dalam *algoritma apriori* untuk mendapat *frequent itemset* yaitu :

1. *Join* (penggabungan)

   Proses ini dilakukan dengan cara pengkombinasian *item* dengan yang *item* lainya hingga tidak bisa terbentuk kombinasi lagi.

2. *Prune* (pemangkasan)

   Proses pemangkasan yaitu hasil dari *item* yang telah dikombinasikan kemudian dipangkas dengan menggunakan minimum *support* yang telah ditentukan.

II.2.1. Konsep Aturan (*Association Rule*)

Analisis asosiasi atau *association rule mining* adalah teknik data *mining* untuk menemukan aturan asosiasi antara kombinasi *item*. Dengan pengetahuan tersebut pemilik pasar swalayan dapat mengatur penempatan barangnya atau merancang kampanye pemasaran dengan memakai kupon diskon untuk kombinasi barang tertentu (Heroe Santoso, dkk, 2016: 20).

II.2.2. Metodologi Dasar Analisis Asosiasi

1. Analisis Pola Frekuensi Tinggi dengan *Algoritma* Apriori

Tahap ini mencari kombinasi *item* yang memenuhi syarat minimum dari nilai *support* dalam basis data. Nilai *support* sebuah *item* diperoleh dengan menggunakan rumus berikut :

$$Support\ A = \frac{Jumlah\ transaksi\ mengandung\ A}{Total\ transaksi} \times 100\%$$

Contoh : $Support = \frac{3}{40} \times 100\ \% = 7.5\ \%$

Pada rumus 1 menjelaskan bahwa nilai *support* diperoleh dengnn cara mencari jumlah transaksi yang mengandung *item A* dengan jumlah seluruh transaksi. Sedangkan nilai *support* dari 2 item diperoleh dari rumus 2 berikut :

$$Support\ (A,B) = \frac{\sum Transaksi\ Mengandung\ A\ dan\ B}{\Sigma\ Transaksi} \times 100\%$$

Contoh : $Support = \frac{5}{40} \times 100\ \% = 12.5\%$

Pada rumus 2 mennjelaskan bahwa nilai *support* diperoleh dengan cara mencari jumlah transaksi yang mengandung *item A* dan *item B* (*item* pertama bersama dengan *item* yang lain) dibagi dengan keseluruhan transaksi. (Riangga Duta Jayapana, Yuniarsi Rahayu, 2015:2).

2. Pembentukan Aturan Asosiasi

Setelah semua pola frekuensi tinggi ditemukan, barulah dicari aturan asosiasi yang memenuhi syarat minimum untuk *confidence* dengan menghitung *confidence* aturan asosiatif "jika A maka B". Nilai *confidence* dari aturan "jika A maka B" diperoleh dengan rumus berikut *:*

$$Confidence\ P(B|A) = \frac{\sum Total\ transaksi\ mengandung\ A\ dan\ B}{\sum Transaksi\ mengandung\ A} \times 100\%$$

Contoh : $Confidence = \frac{5}{17} \times 100\ \% = 29\ \%$

Untuk menentukan aturan asosiasi yang akan dipilih maka harus diurutkan berdasarkan *Support* × *Confidence*. Aturan diambil sebanyak *n* aturan yang memiliki hasil terbesar (Heroe Santoso, dkk, 2016: 21).

II.3. Konsep Database

II.3.1. *SQL Server* 2008 R2

SQL Server merupakan suatu *Relational Database Management Systems* (RDBMS) yang digunakan untuk menyimpan data. Data yang disimpan pada *database* bisa dalam skala kecil maupun besar. Selain itu, penyajiannya

merupakan penyajian pada level fisik karena kita akan menyimpan langsung data pada *database* dengan kondisi yang sebenarnya, yaitu disimpan pada tabel apa, kolom mana, dan menggunakan tipe data saat penyimpanan (Benardo, dkk, 2015: 94).

Pada *SQL Server* 2008 terdapat fitur-fitur yang dapat mengembangkan performa dari *database* tersebut. Beberapa fitur tersebut, yaitu :

1. *Date Data Type*

   Digunakan untuk menyimpan data tanggal saja sehingga akan menghemat *space* pada *server*.

2. *Data Compression*

   Digunakan untuk melakukan *compress* data sehingga ukuran data yang disimpan dalam hal *space hardisk* akan lebih kecil.

3. *Sparse Column*

   Digunakan untuk menyimpan data yang memiliki lebih banyak data *NULL* dengan lebih efisien.

4. *Row Constructor*

   Digunakan untuk melakukan *insert* beberapa data sekaligus dengan satu perintah *INSERT*.

5. *Table Valued Parameter*

   Digunakan untuk melakukan *parsing array* pada bahasa pemrograman, dimana satu variable diberikan data-data yang akan diproses setelahnya.

II.3.1 Tentang *Database*

Pangkalan data atau basis data (bahasa Inggris : *database*), atau sering pula dieja basis data, adalah kumpulan informasi yang disimpan di dalam komputer secara sistematik sehingga dapat diperiksa menggunakan suatu program komputer untuk memperoleh informasi dari basis data tersebut. Perangkat lunak yang digunakan untuk mengelola dan memanggil kueri (*query*) basis data disebut sistem manajemen basis data (*database management system*, DBMS). Sistem basis data dipelajari dalam ilmu informasi.

Selain itu juga dalam mengambil data dari *server* lain akan mengalami penurunan performa. Tetap dengan menggunakan terdistribusi, bisa dengan cepat melakukan akses untuk data pada *database server* yang didistribusikan. Sedangkan untuk tersentralisasi, karena *database*-nya hanya satu dan terpusat (misalnya di *head office*) maka seluruh *client* dari manapun akan mengambil data tersebut dari satu *database*. Dengan demikian data yang diambil tidak akan bermasalah dalam hal konsistensi karena berada dalam satu sumber, tetapi akan membutuhkan *hardware* yang jauh lebih besar dan *bandwidth* yang lebih tinggi. Hal ini dikarenakan *server* tersebut berfungsi untuk menampung penggunaan connection yang sangat banyak.

Database atau basis data adalah kumpulan data yang disimpan secara sistematis di dalam komputer dan dapat diolah atau dimanipulasi menggunakan perangkat lunak (program aplikasi) untuk menghasilkan informasi. Pendefinisian basis data meliputi spesifikasi berupa tipe data, struktur, dan juga batasan-batasan data yang akan disimpan. Basis data merupakan aspek yang sangat penting dalam sistem informasi dimana basis data merupakan gudang penyimpanan data yang

akan diolah lebih lanjut. Basis data menjadi penting karena dapat menghidari duplikasi data, hubungan antar data yang tidak jelas, organisasi data, dan juga update yang rumit. Untuk penyimpanan *database*, biasanya digunakan *relational database*, yaitu suatu mekanisme penyimpanan data pada suatu tabel tertentu yang terhubung antara tabel yang satu dengan tabel lainnya dengan menggunakan *references data*. Data tersebut berupa *field* atau kolom pada tabel yang menghubungkan tabel yang satu dengan tabel yang lain (Benardo, dkk, 2015: 94).

II.4. *Microsoft Visual Studio* 2010

Visual Basic.Net 2010 merupakann *core* dari pembuatan aplikasi berbasis *.Net*. yang merupakan lingkungan pemrograman yang mempermudah tahapan *desain, development, debugging*, dan *deployment* dari aplikasi berbasis *.Net* dan *XML web service*, serta meningkatkan efisiensi *developer* dengan menyediakan lingkungan pemrograman yang sudah biasa digunakan.

.NET Framework adalah teknologi inti yang menyediakan berbagai library untuk digunakan oleh aplikasi di atasnya. Komponen inti *.NET Framework* adalah *Common Language Runtime* (CLR) yang menyediakan *run time environment* untuk aplikasi yang dibangun menggunakan *Visual Studio.NET*, terlepas dari jenis bahasa pemrogramannya (Benardo, dkk, 2015: Hal. 93).

II.5. *Unified Modelling Language* (UML)

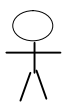
Pada perkembangan teknologi perangkat lunak, diperlukan adanya bahasa yang digunakan untuk memodelkan perangkat lunak yang akan dibuat dan perlu adanya standarisasi agar orang diberbagai negara dapat mengerti pemodelan

perangkat lunak. Seperti yang kita ketahui bahwa menyatukan banyak kepala untuk menceritakan sebuah ide dengan tujuan untuk memahami hal yang sama tidaklah mudah, oleh karena itu diperlukan sebuah bahasa pemodelan perangkat lunak yang dapat dimengerti oleh banyak orang. Hasil pemodelan pada *Object Oriented Analysis And Design* (OOAD) terdokumentasikan dalam bentuk *Unified Modeling Language* (UML). UML adalah bahasa spesifikasi standar yang dipergunakan untuk mendokumentasikan, menspesifikasikan dan membangun perangkat lunak.

II.5.1. *Use Case Diagram*

Use case atau diagram *use case* merupakan pemodelan untuk melakukan (*behavior*) sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Dapat dikatakan, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu. Berikut adalah simbol-simbol yang ada pada diagram *use case* :

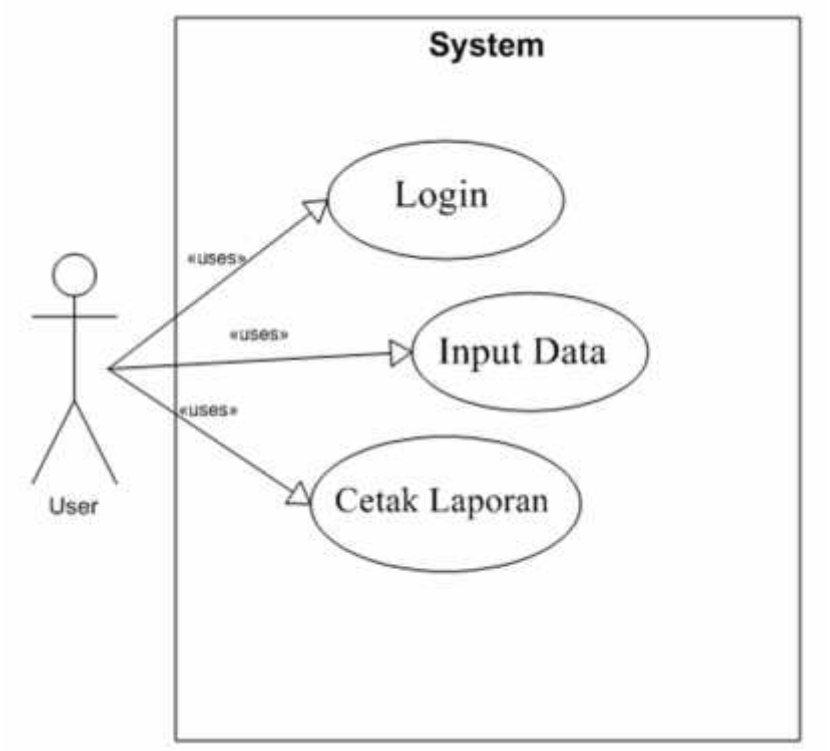
Tabel II.1. Tabel *Use Case Diagram*

Gambar	Keterangan
	*Use case* menggambarkan fungsionalitas yang disediakan sistem sebagai unit-unit yang bertukar pesan antar unit dengan aktor, biasanya dinyatakan dengan menggunakan kata kerja di awal nama *use case*.
	Aktor adalah *abstraction* dari orang atau sistem yang lain yang mengaktifkan fungsi dari target sistem. Untuk mengidentifikasikan aktor, harus ditentukan pembagian tenaga kerja dan tugas-tugas yang berkaitan dengan peran pada konteks target sistem. Orang atau sistem bisa muncul dalam beberapa peran. Perlu dicatat bahwa aktor berinteraksi dengan *use case*, tetapi tidak memiliki control terhadap *use case*.

———	Asosiasi antara aktor dan *use case*, digambarkan dengan garis tanpa panah yang mengindikasikan siapa atau apa yang meminta interaksi secara langsung dan bukannya mengidikasikan aliran data.
⟶	Asosiasi antara aktor dan *use case* yang menggunakan panah terbuka untuk mengidinkasikan bila aktor berinteraksi secara pasif dengan sistem.
--------->	*Include*, merupakan di dalam *use case* lain (*required*) atau pemanggilan *use case* oleh *use case* lain, contohnya adalah pemanggilan sebuah fungsi program.
<--------	*Extend*, merupakan perluasan dari *use case* lain jika kondisi atau syarat terpenuhi.

(Sumber : Gellysa Urva dan Helmi Fauzi Siregar : 2015)

Untuk ilustrasi *actor*, *usecase* dan *system* ditunjukkan pada gambar II.3 berikut :



Gambar II.3 *Usecase* Diagram

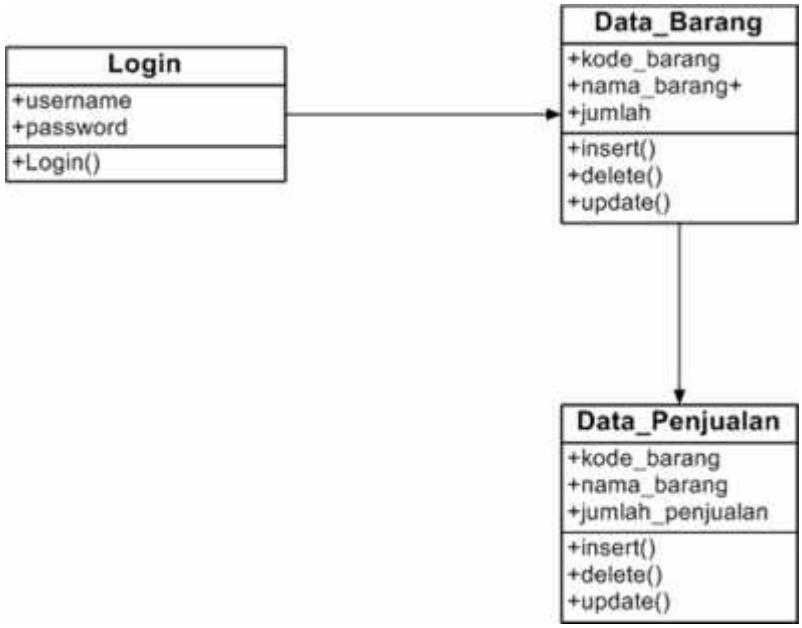
II.5.2. *Class Diagram*

Merupakan hubungan antar kelas dan penjelasan detail tiap-tiap kelas di dalam model desain dari suatu sistem, juga memperlihatkan aturan-aturan dan tanggung jawab entitas yang menentukan perilaku sistem.

Tabel II.2. *Multiplicity Class Diagram*

Multiplicity	**Penjelasan**
1	Satu dan hanya satu
0..*	Boleh tidak ada atau 1 atau lebih
1..*	1 atau lebih
0..1	Boleh tidak ada, maksimal 1
n..n	Batasan antara. Contoh 2..4 mempunyai arti minimal 2 maksimum 4

Contoh diagram *class* dapat dilihat pada gambar II.4 dibawah ini:

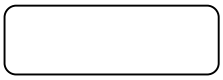
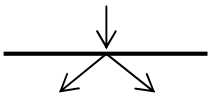
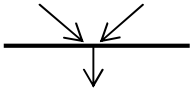
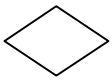


Gambar II.4 *Class* Diagram

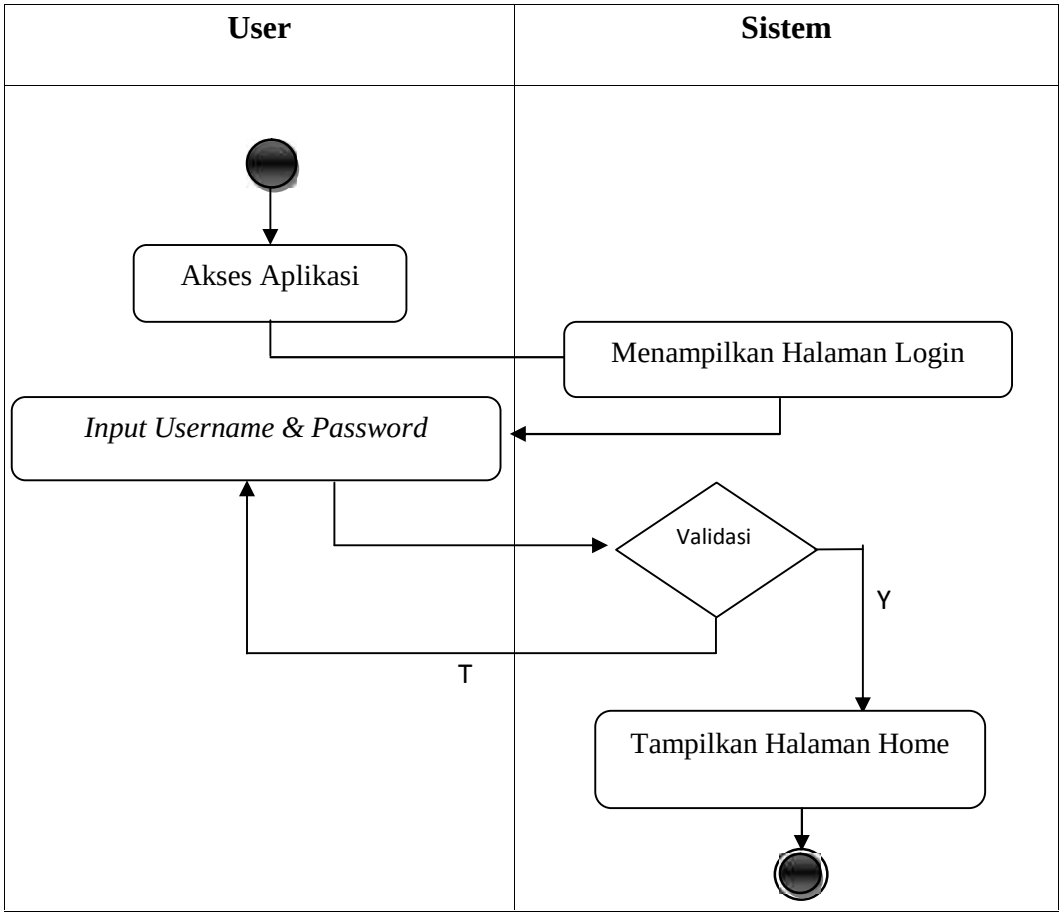
II.5.3. *Activity Diagram*

Activity Diagram menggambarkan *workflow*(aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis.

Tabel II.3. Tabel *Activity Diagram*

Gambar	Keterangan
	*Start point*, diletakkan pada pojok kiri atas dan merupakan awal aktifitas.
	*End point*, akhir aktifitas.
	*Activites*, menggambarkan suatu proses/kegiatan bisnis.
	*Fork* (Percabangan), digunakan untuk menunjukkan kegiatan yang dilakukan secara parallel atau untuk menggabungkan dua kegiatan pararel menjadi satu.
	*Join* (penggabungan) atau rake, digunakan untuk menunjukkan adanya dekomposisi.
	*Decision Points*, menggambarkan pilihan untuk pengambilan keputusan, *true, false.*
	*Swimlane*, pembagian *activity* diagram untuk menunjukkan siapa melakukan apa.

Contoh gambar untuk *Activity* Diagram dapat dilihat pada gambar II.5 :

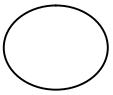


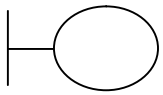
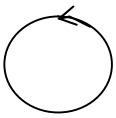
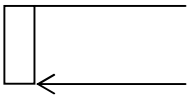
Gambar II.5 *Activity* Diagram

II.5.4. *Sequence Diagram*

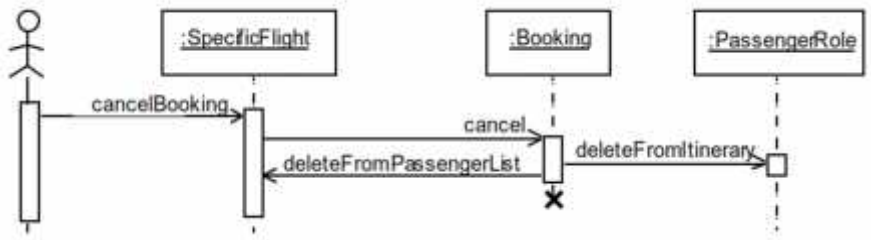
Sequence diagram menggambarkan kelakuan objek pada *usecase* dengan mendeskripsikan waktu hidup objek dan pesan yangc dikirimkan dan diterima antar objek.

Tabel II.4. Tabel *Sequence Diagram*

Gambar	Keterangan
	*Entity Class*, merupakan bagian dari sistem yang berisi kumpulan kelas berupa entitas-entitas yang membentuk gambaran awal sistem dan menjadi landasan untuk menyusun basis data.

	*Boundary Class,* berisi kumpulan kelas yang menjadi *interface* atau interaksi antara satu atau lebih aktor dengan sistem, seperti tampilan formentry dan *form* cetak.
	*Control class,* suatu objek yang berisi logika aplikasi yang tidak memiliki tanggung jawab kepada entitas, contohnya adalah kalkulasi dan aturan bisnis yang melibatkan berbagai objek.
	Message, simbol mengirim pesan antar *class*.
	*Recursive,* menggambarkan pengiriman pesan yang dikirim untuk dirinya sendiri.
	*Activation, activation* mewakili sebuah eksekusi operasi dari objek, panjang kotak ini berbanding lurus dengan durasi aktivitas sebuah operasi.
	Lifeline, garis titik-titik yang terhubung dengan objek, sepanjang *lifeline* terdapat *activation.*

Contoh diagram *class* dapat dilihat pada gambar II.6 dibawah ini:

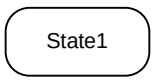


Gambar II.6 *Sequence* Diagram

II.5.5. *Statechart Diagram*

State machinediagram atau *statechart diagram* atau dalam bahasa Indonesia disebut diagram mesin status atau sering juga disebut diagram status digunakan untuk menggambarkan perubahan status atau transisi status dari sebuah mesin atau sistem atau objek.(Rosa A.S ,Dkk. 2013, Hal :163).

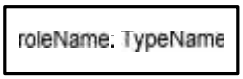
Tabel II.5. Tabel *Staechart Diagram*

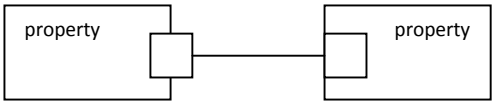
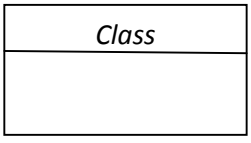
Simbol	Deskripsi
Start / Status Awal (*Initial State*) 	*Start* atau *initial state* adalah*state* atau keadaan awal pada saat sistem mulai hidup.
End / Status Akhir (*Final State*) 	*End* atau *final state* adalah *state* keadaan akhir dari daur hidup suatu sistem.
Event 	*Event* adalah kegiatan yang menyebabkan berubahnya status mesin.
State 	*State* atau status adalah keadaan sistem pada waktu tertentu.state dapat berubah jika ada event tertentu yang memicu perubahan tersebut.

II.5.6. *Composite Structure Diagram*

Diagram ini dapat digunakan untuk menggambarkan struktur dari bagian-bagian yang saling terhubung maupun mendeskripsikan struktur pada saat berjalan (*runtime*) dari *instance* yang saling terhubung.(Rosa A.S ,Dkk. 2013,Hal : 150).

Tabel II.6. Tabel *Composite Stucture Diagram*

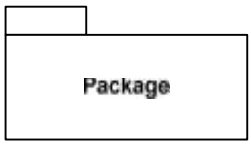
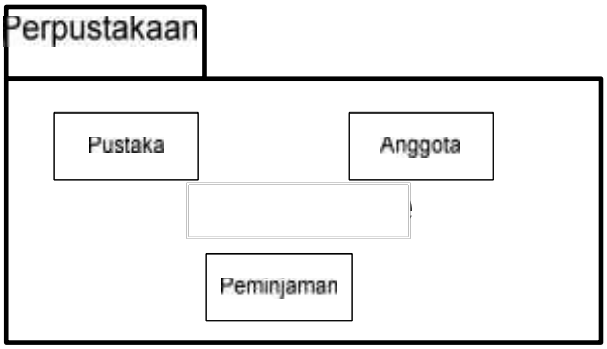
Simbol	Deskripsi
Property  	*Property* adalah satu set dari suatu *instance.* *Rolename* : peran/nama/identitas dari *property* (opsional) *TypeName* :tipe kelas dari *property* (harus ada)
Conenctor [**multiplicity1**] [**multiplicity2**] [**roleName1**][**roleName2**]	*Connector* adalah cara komunikasi dari 2 buah *instance.* *connName* : nama *connector*(opsional)

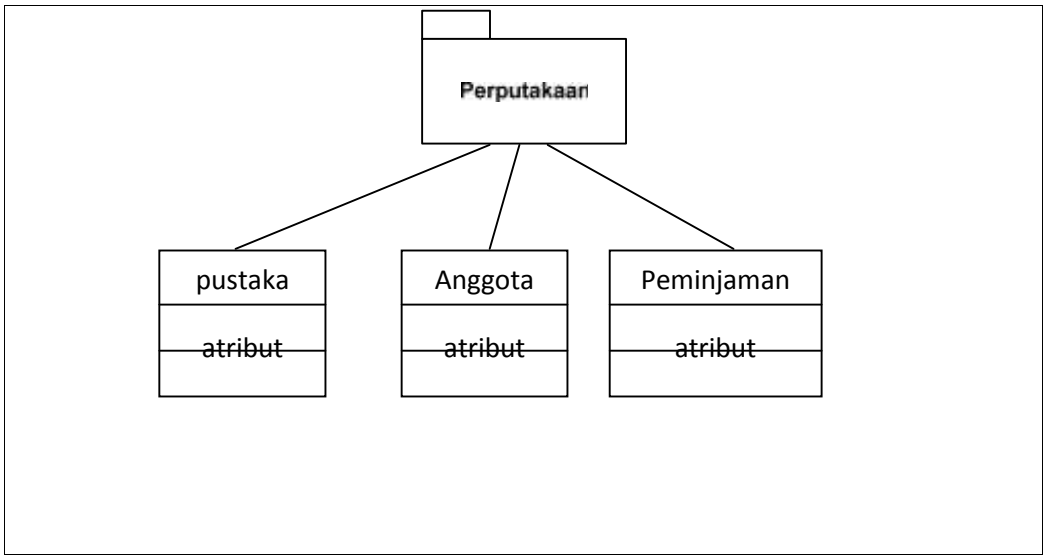
nama connector : tipe connector	*connType* : tipe *connector*(opsional)
Port  portName: EntityName[n] pemakainnya adalah sebagai berikut: 	*Port* adalah cara yang digunakan dalam diagram *composite structure* tanpa menampilkan detail internal dari suatu *system.* *Port*digambarkan dalam bentuk kotak kecil yang menempel atau didalam suatu *property.* *Port* digambarkan menempel *property* jika fungsi tersebut dapat diakses *public*. Sedangkan*Port* digambarkan di dalam suatu *property* jika fungsi tersebut bersifat *protected.*
Class 	Kelas; jika yang akan dijabarkan strukturnya adalah sebuah kelas

II.5.7. *Package Diagram*

Package diagram menyediakan cara mengumpulkan elemen-elemen yang saling terkait dalam diagram UML. Hampir semua diagram dalam UML dapat dikelompokkan menggunakan *Package diagram*.(Rosa A.S ,Dkk. 2013, Hal :153).

Tabel II.7. Tabel *Package Diagram*

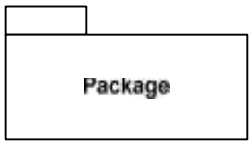
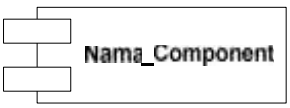
Simbol	Deskripsi
Package 	*Package* merupakan sebuah bungkusan dari satu atau lebih kelas atau elemen diagram UML lainnya.
Elemen dalam *package* digambarkan di dalam *package* 	
Elemen dalam *package* digambarkan diluar *package*	



II.5.8. *Component Diagram*

Diagram komponen atau *component diagram* dibuat untuk menunjukkan organisasi dan ketergantungan diantara kumpulan komponen dalam sebuah sistem. Diagram komponen fokus pada komponen sistem yang dibutuhkan dan ada didalam sistem.(Rosa A.S ,Dkk. 2013, Hal :148).

Tabel II.8. Tabel *Component Diagram*

Simbol	Deskripsi
Package 	*Package* merupakan sebuah bungkusan dari satu atau lebih komponen
Komponen 	Komponen sistem
Kebergantungan / *Depedency* 	Kebergantungan antar komponen,arah panah mengarah pada komponen

	yang dipakai
Antarmuka / *interface*  Nama_interface	Sama dengan konsep *interface*pada pemrograman berorientasi objek,yaitu sebagai antarmuka komponen agar tidak mengakses langsung komponen
Link _____	Relasi antar komponen

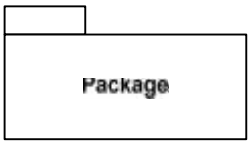
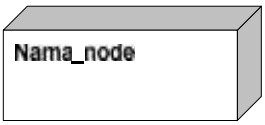
II.5.9. *Deployment Diagram*

Diagram deployment atau *deploymen diagram* menunjukkan konfigurasi komponen dalam proses eksekusi aplikasi. *Diagram deployment* juga dapat digunakan untuk memodelkan hal-hal berikut:

1. Sistem tambahan (embedded system) yang menggambarkan racangan device, node, dan hardware.
2. Sistem client/server
3. Sistem terdistribusi murni
4. Rekayasa ulang aplikasi.(Rosa A.S ,Dkk. 2013, Hal :154).

Tabel II.9. Tabel *Deployment Diagram*

Simbol	Deskripsi
Package	*Package* merupakan sebuah

	bungkusan dari satu atau lebih node.
Node 	Biasanya mengacu pada perangkat keras (*hardware*), perangkat lunak yang tidak dibuat sendiri (*software*), jika di dalam *node* disertakan komponen untuk mengkonsistensikan rancangan maka komponen yang diikutsertakan harus sesuai dengan komponen yang telah didefinisikan sebelumnya pada diagram komponen
Kebergantungan / *depedency* 	Kebergantungan antar *node,* arah panah mengarah pada *node* yang dipakai
Link 	Relasi antar *node*