BAB II

TINJAUAN PUSTAKA

2.1. *Association Rule*

Association rule mining digunakan untuk mencari hubungan korelasi atau asosiasi antaritem pada dataset ((Mahmudah & Aribowo, 2014). Salah satu penerapan *association rules* adalah *market basket analysis*, dimana bertujuan untuk menemukan bagaimana item yang dibeli pelanggan dalam supermarket atau toko saling berhubungan. Pencarian *association rules* dilakukan melalui dua tahap, yaitu pencarian *frequent itemset* dan penyusunan rules. Penting tidaknya *association rules* dapat diketahui dengan dua parameter, yaitu *support* (nilai penunjang) dan *confidence* (nilai kepastian). Support adalah ukuran yang menunjukkan tingkat dominasi itemset dari keseluruhan transaksi. Persamaan 1 untuk menentukan nilai *support*

$$Support = \frac{(X \cup Y)\ count}{n} \quad (1)$$

dimana $(X \cup Y)\ count$ adalah jumlah transaksi yang mengandung x dan y, dan n adalah total transaksi. Parameter lainnya adalah *confidence* yaitu nilai ukuran seberapa besar valid tidaknya *association rules*. Sebuah *association rules* dengan *confidence* sama atau lebih besar dari minimum *confidence* dapat dikatakan sebagai valid *association rules*. Persamaan 2 untuk menentukan nilai *confidence,* dimana X count adalah jumlah kemunculan item X pada seluruh transaksi.

$$Confidence = \frac{(X \cup Y).count}{X.count} \quad (2)$$

2.2. Algoritma *Frequent Pattern Growth (FP-Growth)*

Penggalian itemset yang *frequent* dengan menggunakan algoritma FP-Growth akan dilakukan dengan cara membangkitkan struktur data tree atau disebut dengan FP-Tree (Han & Kamber, 2000). Algoritma FP-Growth memiliki tiga tahapan utama yaitu, (a) pembangkitan *conditional pattern base*, (b) pembangkitan *conditional pattern tree*, dan (c) pencarian *frequent itemse*t. Pseudocode dari algoritma FP- Growth sebagai berikut:
(dikutip dari: Han, J., Kamber, M. (2000). Data mining: Concepts and techniques)

```
Procedure FP_growth(Tree, α)

(1) if tree contains a single path P then
(2)   for each combination (denoted as β) of the nodes in the path P
(3)     generate pattern β ∪ α with support_count = minimum support count
f nodes in β
(4) else for each a_i in the header of Tree {
(5)   generate pattern β = a_i ∪ α with support_count = a_i.support_count;
(6)   construct β´s conditional pattern base and then β´s conditional FP_tree
ree.
```

2.3. *UML (Unified Modelling Language)*

Unified Modelling Language (UML) adalah sebuah "bahasa" yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem piranti lunak. UML menawarkan sebuah standar untuk merancang model sebuah sistem.

UML terdiri atas banyak elemen-elemen grafis yang digabungkan membentuk diagram. Tujuan representasi elemen-elemen grafis ke dalam diagram adalah untuk menyajikan beragam sudut pandang dari sebuah sistem berdasarkan fungsi masing-masing diagram tersebut. Kumpulan dari beragam sudut pandang inilah yang kita sebut sebuah model. UML mendefinisikan diagram-diagram sebagai berikut:

2.3.1. *Use Case Diagram*

*Use case diagram*menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah “apa”yang diperbuat sistem, dan bukan “bagaimana”. Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem Seorang/sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu.

a. Bisnis *Use Case Diagram*

Diagram *usecase* bisnis digunakan untuk memodelkan aktivitas bisnis organisasi sebagai landasan pembuatan *use case* sistem. Digram *use case* bisnis digambarkan menurut

perspektif organisasi. Ia tidak memebedakan apakah aktivitas tersebut dilakukan secara manual atau otomatis menggunakan perangkat lunak.

Gambar 2.1: Contoh Bisnis *Use Case* Diagram

Sumber: (Sholiq, 2010)

b. S*ystem use case diagram*

Diagram use case sistem menyajikan interaksi antara use case dan aktor dalam sistem yang akan dikembangkan.

Gambar 2.3: Contoh System *Use Case Diagram*

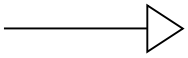
Sumber: (Sholiq, 2010)

Adapun notasi *use case* diagram dijelaskan sebagai berikut :

Tabel 2.1. Notasi Dalam *Use case*

No	Nama	Gambar	Fungsi
1	*Actor*		Menggambarkan segala pengguna software aplikasi (sistem).

2.3.2. *Class Diagram*

2	*Use case*		Menjelaskan urutan kegiatan yang dilakukan aktor dan sistem untuk mencapai suatu tujuan tertentu.
3	*Directed Association*		Menunjukkan baik aliran pesan atau informasi antar objek maupun hubungan antar objek.
4	*Dependency*		Relasi yang menunjukkan bahwa perubahan pada salah satu elemen memberi pengaruh pada elemen lain.
5	*Generalization*		Disebut juga *inheritance* (pewarisan), sebuah elemen dapat merupakan spesialisasi dari elemen lainnya.
6	*Association*		Menghubungkan *link* antar elemen.
7	*Include*		Kelakuan yang harus terpenuhi agar sebuah *event*dapat terjadi, dimana kondisi ini sebuah *use case*adalah bagian *use case*lain.
8	*Extend*		Kelakuan yang hanya berjalan di bawah kondisi tertentu.

Class Diagram adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah obyek dan merupakan inti dari pengembangan dan desain berorientasi obyek. *Class* menggambarkan keadaan (*atribut*/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). Sebuah *Class* memiliki tiga area pokok:

1. Nama, merupakan nama dari sebuah kelas
2. *Atribut*, merupakan peroperti dari sebuah kelas. *Atribut* melambangkan batas nilai yang mungkin ada pada obyek dari *class*
3. Operasi, adalah sesuatu yang bisa dilakukan oleh sebuah *class* atau yang dapat dilakukan oleh *class* lain terhadap sebuah *class.*

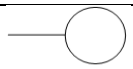
Atribut dan metoda dapat memiliki salah satu sifat berikut :

1. *Private*, tidak dapat dipanggil dari luar *class* yang bersangkutan.
2. *Protected*, hanya dapat dipanggil oleh *class* yang bersangkutan dan anak-anak yang mewarisinya.
3. *Public*, dapat dipanggil oleh siapa saja.
4. *Package*, hanya dapat dipanggil oleh instance sebuah class pada paket yang sama.

Dalam *class diagram*, terdapat Multiplisitas atau *Multiplicity* yang berarti bahwa jumlah banyaknya obyek sebuah *class* yang berelasi dengan sebuah obyek lain pada *class* lain yang berasosiasi dengan *class* tersebut. Untuk menyatakan multiplisitas kita dapat meletakkannya diatas garis asosiasi berdekatan dengan *class* yang sesuai. Adapun macam-macam multiplisitas yang dipakai adalah sebagai berikut :

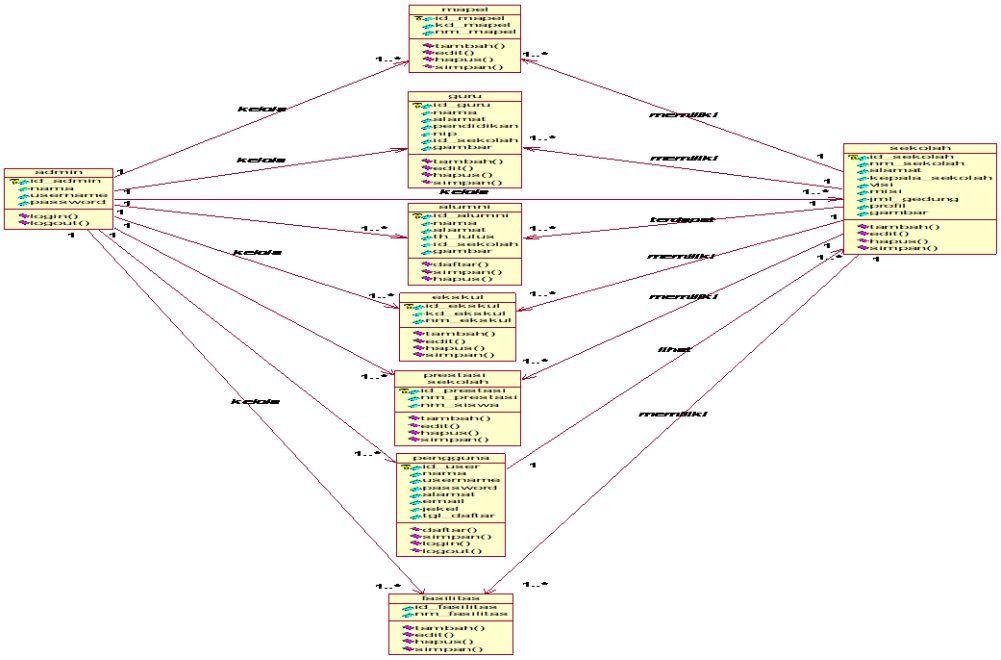
a. Zero "0" mengandung arti bahwa nilai multiplisitasnya adalah kosong
b. One "1" mengandung arti bahwa nilai multiplisitasnya adalah satu
c. Zero to one "0.. 1" mengandung arti bahwa nilai multiplisitasnya adalah kosong atau satu
d. Zero to more "0.. *" mengandung arti bahwa nilai multiplisitasnya adalah kosong atau banyak
e. One to more "1.. *" mengandung arti bahwa nilai multiplisitasnya adalah satu atau banyak
f. More "*" mengandung arti bahwa nilai multiplisitasnya adalah banyak.

Tabel 2.2 : Notasi pada *Class Diagram*

No	Nama	Gambar	Fungsi
1	*Class*		Merupakan kumpulan objek yang memiliki atribut dan operasi yang sama atau mengabstraksikan elemen-elemen yang sedang dibangun.
2	*Interface*		Kumpulan operasi tanpa implementasi dari suatu *class*.
3	*Association*		Hubungan struktural yang menggambarkan sehimpunan mata rantai antar objek.
4	*Directed Association*		Menunjukkan baik aliran pesan atau informasi antar objek maupun hubungan antar objek.
5	*Aggregation*		Hubungan yang menyatakan bagian (terdiri atas). Digambarkan dengan garis dan belah ketupat di ujung.

6	*Generalisasi*		Hubungan spesialisasi di mana objek dari elemen khusus (anak) merupakan pengganti untuk objek elemen umum (induk).

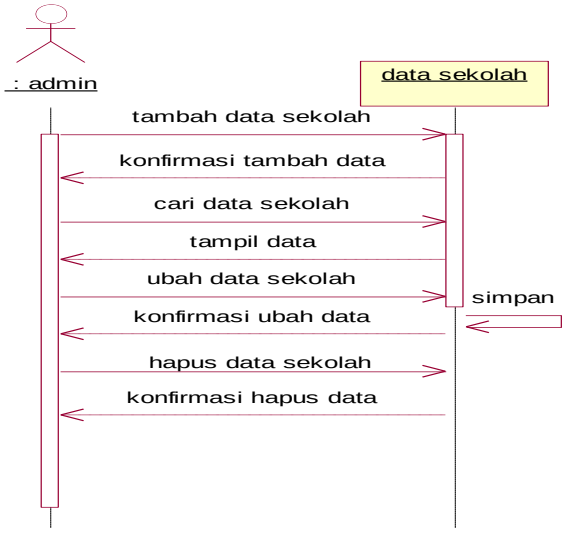
Berikut adalah contoh penggambaran dari *Class Diagram :*



Gambar 2.4 : Contoh *Class Diagram*

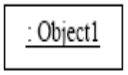
2.3.3. *Sequence Diagram*

Squence Diagram digunakan untuk menunjukkan alur fungsionalitas yang melalui sebuah use case yang disusun dalam urutan waktu. (Sholiq, 2010)



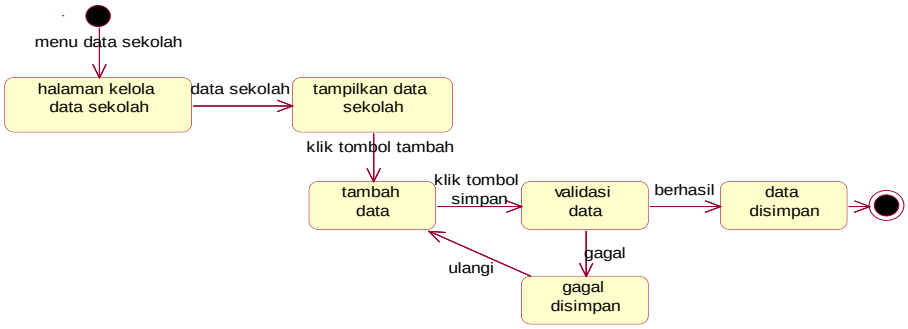
Gambar 2.5 : Contoh *Squence Diagram*

Tabel 2.3. Notasi *Sequence Diagram*

No	Nama	Gambar	Fungsi
1.	*Object*		*Object* merupakan instance dari sebuah *class* dan dituliskan tersusun secara *horizontal.* Digambarkan sebagai sebuah *class* (kotak) dengan nama obyek didalamnya yang diawali dengan sebuah titik koma.
2.	*Actor*		*Actor* juga dapat berkomunikasi dengan object, maka actor juga dapat diurutkan sebagai kolom. Simbol *Actor* sama dengan simbol pada *Actor Use Case Diagram.*
3.	*Lifeline*		*Lifeline* mengindikasikan keberadaan sebuah object dalam basis waktu. Notasi untuk *Lifeline* adalah garis putus-putus *vertikal* yang ditarik dari sebuah obyek.
4.	*Activation*		*Activation* dinotasikan sebagai sebuah kotak segi empat yang digambar pada sebuah *lifeline. Activation* mengindikasikan sebuah obyek yang akan melakukan sebuah aksi.
5.	*Massage*		*Message,* digambarkan dengan anak panah *horizontal* antara *Activation. Message* mengindikasikan komunikasi antara *object-object.*

2.3.4. *Statechart Diagram.*

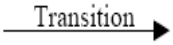
Statechart diagram memungkinkan untuk memodelkan bermacam-macam state (urutan keadaan sesaat) yang mungkin dialami oleh obyek tunggal (Sholiq, 2010).



Gambar 2.6 : Contoh *Statechart Diagram*

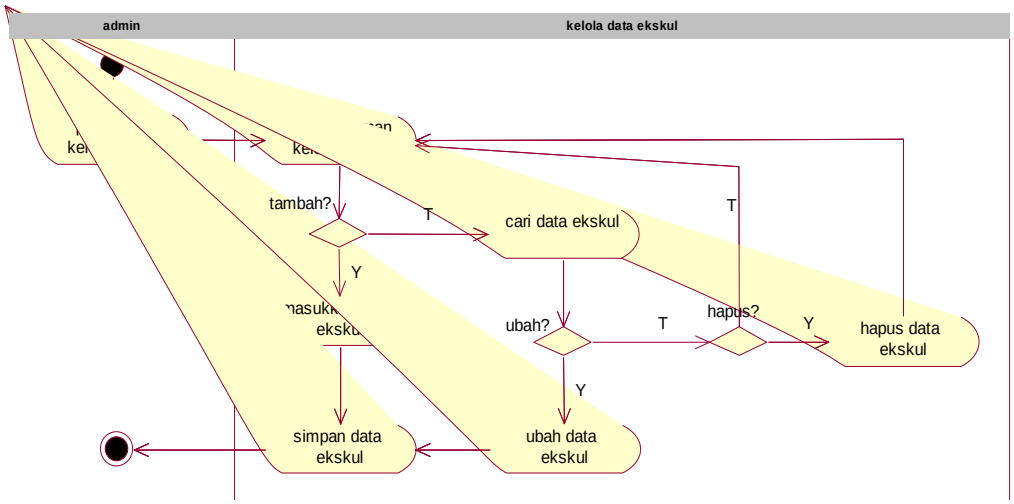
Berikut adalah notasi dari *statechart diagram :*

Tabel 2.4. Notasi – Notasi *State Diagram*

No	Nama	Gambar	Fungsi
1.	*State*		*Notasi State* menggambarkan kondisi sebuah *entitas*, dan digambarkan dengan segiempat yang pinggirnya tumpul dengan nama *state* didalamnya.
2.	*Transition*		Sebuah *Transition* menggambarkan sebuah perubahan kondisi *objek* yang disebabkan oleh sebuah *event*. *Transition* digambarkan dengan sebuah anak panah dengan nama *event* yang ditulis diatasnya, dibawahnya atau sepanjang anak panah tersebut.
3.	*Initial state*		*Initial State* adalah sebuah kondisi awal sebuah object sebelum ada perubahan keadaan. Initial *State* digambarkan dengan sebuah lingkaran *solid*. Hanya satu *Initial State* yang diizinkan dalam sebuah diagram
4.	*Final State*		*Final State* menggambarkan ketika objek berhenti memberi *respon* terhadap sebuah *event*. *Final State* digambarkan dengan lingkaran *solid* didalam sebuah lingkaran kosong.

2.3.5. *Activity Diagram*

*Activity diagram*mendefinisikan dari mana alur kerja (*workflow*) dimulai dan di mana alur kerja berakhir. Aktivitas apa saja yang terjadi di dalam alur kerja, dan apa saja yang dilakukan saat sebuah aktivitas terjadi. Berikut contoh dari *activity* diagram:

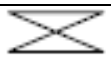


Gambar 2.7 : Contoh *Activity Diagram*

Adapun notasi dari activity diagram dijelaskan dalam tabel berikut :

Tabel 2.5. Notasi *Activity Diagram*

No	Simbol	Keterangan
1.		Titik Awal
2.		Titik Akhir
3.		*Activity*
4.		Pilihan Untuk mengambil Keputusan
5.		*Fork*; Digunakan untuk menunjukkan kegiatan yang dilakukan secara paralelatau untuk menggabungkan dua kegiatan paralel menjadi satu.

6.		*Rake*;Menunjukkan adanya dekomposisi
7.		Tanda Waktu
8.		Tanda pengiriman
9.		Tanda penerimaan
10.		Aliran akhir (*Flow Final*)