BAB 2
TINJAUAN PUSTAKA

2.1. Profil Perusahaan

PT. Xyclus Cipta Teknologi berada di Negara Indonesia, Provinsi Jawa Barat Kota Bandung, alamat : Jalan Bojong Kacor 01, telepon (+6222) 2505663. Perusahaan ini dapat di tempuh dengan kendaraan umum lalu dilanjutkan dengan berjalan kaki jika tidak memiliki kendaraan pribadi.

2.1.1. Sejarah Perusahaan

PT. Xyclus Cipta Teknologi adalah perusahaan yang bergerak di bidang pelayanan jasa teknologi yang berdiri sejak tahun 2009. Secara spesifik perusahaan ini berperan sebagai konsultan bisnis sebagai pihak ketiga yang memanfaatkan teknologi informasi untuk men-*support* kegiatan bisnis dipihak pertama.

2.1.2. Visi dan Misi

Visi adalah impian suatu organisasi dalam bentuk cita-cita yang ingin dicapai dimasa mendatang, guna menjamin kesejahteraan dalam jangka panjang. Adapun visi dari PT. Xyclus Cipta Teknologi adalah memasyarakatkan teknologi informasi pada perusahaan pihak pertama yang masih menggunakan teknik konvensional dalam proses bisnisnya, dengan harapan menghasilkan kerja sama yang baik terhadap, saling produktif dan menguntungkan.

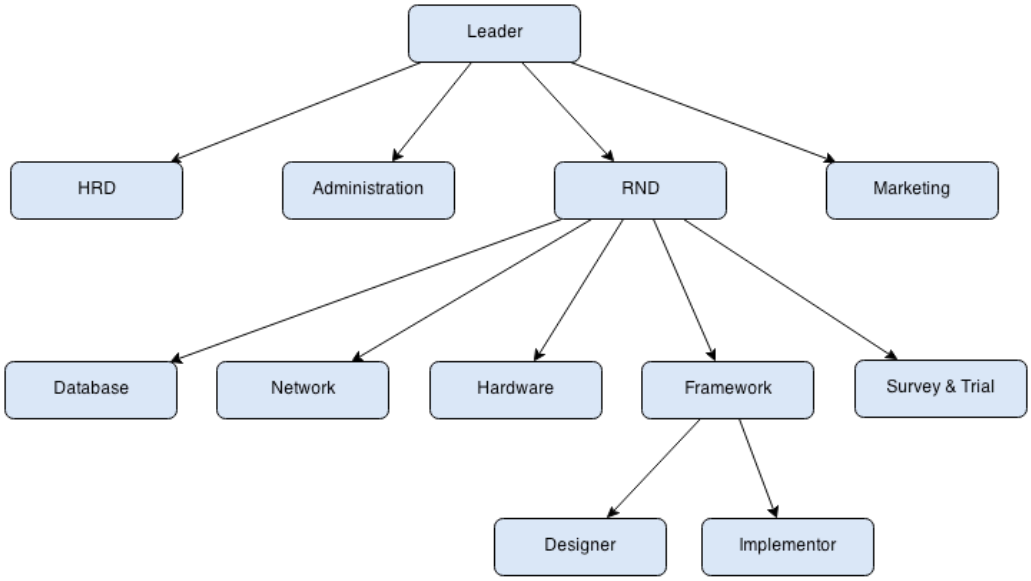
Misi adalah titik poin tentang apa yang harus dilakukan suatu organisasi untuk mencapai Visi yang dimilikinya. Adapun Misi dari PT. Xyclus Cipta Teknologi adalah sebagai berikut :

1. Meningkatkan sumber daya materi, manusia dengan cara yang baik dan benar.
2. Meningkatkan potensi peluang usaha baik internal maupun konsumen.
3. Mengembangkan potensi karyawan dengan saling membahu tugas yang di bebankan, secara tim maupun individu.

4. Menghasilkan sikap yang baik terhadap beban pekerjaan, nyaman, terarah, terbuka, dan kekeluargaan.
5. Senantiasa memelihara profesionalisme dalam menyelesaikan beban tugas atau kepercayaan konsumen.

2.1.3. Struktur Organisasi

Struktur Organisasi adalah suatu hubungan antara tiap bagian untuk menjalin kegiatan operasional dengan harapan agar tercapainya suatu tujuan. Struktur organisasi juga mencerminkan bagaimana pekerjaan dikelompokkan dan dikoordinasikan secara formal, berikut struktur organisasi di PT. Xyclus Cipta Teknologi terdapat pada **Gambar 2. 1**.



Gambar 2. 1. Struktur Organisasi PT. Xyclus Cipta Teknologi

2.1.4. Deskripsi Tugas

Deskripsi tugas menerangkan wewenang dan beban kerja yang dimiliki oleh tiap bagian yang diterangkan pada **Gambar 2. 1**, rincian beban pekerjaan adalah sebagai berikut :

1. *Leader* atau pimpinan dalam perusahaan ini memiliki tugas antara lain : pengambilan keputusan, membimbing karyawan, mengatur organisasi, mengatur program kerja, mengatur fasilitas sebagai kebutuhan, dan pengevaluasi pekerjaan secara global.

2. *HRD* atau *Human Resource Development* dalam perusahaan ini memiliki tugas antara lain : menysusn serta ngevaluasi anggaran biaya, bertanggung jawab terhadap perencaan sumber daya manusia, melakukan transfering, melakukan promosi kepada karyawan jika dianggap perlu, bertanggung jawab terhadap *payroll* karyawan.
3. *Marketing* atau tim pemasaran dalam perusahaan ini memiliki tugas antara lain : menysun perencaan target pemasaran produk, lalu mendapatkannya sebagai konsumen, menyusun laporan promosi, melakkukan laporan kunjungan.
4. *Administration* atau bagian administrasi dalam perusahaan ini memiliki tugas antara lain : menyusun dan merangkun segala bentuk pengeluar dan pemasukan terhadap internal perusahan, lalu menjadikannya laporan *invesment* perusahaan.
5. *RND* atau *Resource And Developmen*t dalam perusahaan ini memiliki tugas antara lain : merencanakan program kerja perangkat lunak dan perangkat keras sebagai pembangunan produk, melakukan pengawasan, lalu mengevaluasi produk, menganalisa hasil pekerjaan secara terperinci, lalu membuat laporan umum hasil pekerjaan.
6. *Database* atau bagian basis data dalam perusahaan ini memiliki tugas antara lain : membangun tatanan basis data yang diperlukan dalam men-*support* pembangunan perangkat lunak.
7. *Network* atau jaringan dalam perusahaan ini memiliki tugas antara lain : membangun, menata jaringan internet terhadap internal sebagai kebutuhan pokok dalam perencanaan pembangunan perangkat lunak.
8. *Hardware* atau perangkat keras dalam perusahaan ini memiliki tugas antara lain : menyediakan dan melakukan pemeliharan terhadap barang elekronik untuk segala keperluan internal seperti pembangunan ataupun kebutuhan pokok alat kerja seorang karyawan jika dibutuhkan.
9. *Framework* atau kerangka kerja dalam perangkat lunak di perusahaan ini memiliki tugas antara lain : menyusun bingkai-bingkai cetakan dalam bentuk perangkat lunak, membuat standar atau aturan khusus

sebuah praktikal pembangunan perangkat lunak agar mudah di implementasikan.

10. *Survey & trials* atau biasa disebut tim survei dan uji coba dalam perusahaan ini memiliki tugas antara lain : melakukan survei kepada calon konsumen, melakukan uji coba terhadap produk yang sudah mamasuki tahap masa uji, melakukan *resume* laporan uji coba.
11. *Designer* adalam perusahaan ini memiliki tugas antara lain : merancang antar muka suatu perangkat .
12. *Implementor* atau pelaksana kerja dalam perusahaan ini memiliki tugas antara lain : membantu pembangunan perangkat lunak dengan cara menysusn kode-kode pembangun perangkat lunak berdasarkan kerangka kerja yang sudah ditentukan.

2.2. Landasan Teori

Landasan teori merupakan kajian relevan yang telah diuji kebenarannya sehingga dapat dijadikan acuan penelitian, landasan teori digunakan untuk dijadikan bahan pertimbangan tentang variabel-variabel yang bersangkutan dan menjadi bahan pembanding yang akan dibahas secara kritis dalam suatu penelitian.

2.3. Definisi Sistem

Sistem dapat didefinisikan sebagai suatu kesatuan komponen-komponen yang saling berinteraksi, melakukan suatu proses guna mencapai kesamaan suatu tujuan tertentu.

2.3.1. Klasifikasi Sistem

Sistem dapat diklasifikasikan dari beberapa sudut pandangan menurut HM Jogiyanto dalam buku *Pengenalan Komputer* [8] adalah sebagai berikut :

1. Sistem abstrak dan fisik

Sistem abstrak adalah sistem tidak kasat mata, biasanya sistem ini merupakan pemikiran atau ide-ide sedangkan sistem fisik merupakan sistem tampak secara fisik.

2. Sistem alamiah dan buatan

Sistem alamiah adalah sistem yang terbentuk melalui proses alam dan identik terjadi dengan sendirinya, sedangkan sistem buatan manusia merupakan sistem yang dibuat oleh manusia berdasarkan rekayasa dan pemikiran.

3. Sistem tertentu dan tak tentu

Sistem tertentu merupakan sistem yang beroperasi dengan prediksi yang sesuai dengan harapan sedangkan sistem tak tentu merupakan operasian sebuah sistem yang masih mengandung probabilitas.

4. Sistem tertutup dan terbuka

Sistem tertutup merupakan sistem yang tidak berhubungan dengan lingkungan luar, sedangkan sistem terbuka adalah sistem yang memiliki hubungan dengan lingkungan luarnya sehingga hal ini dapat.

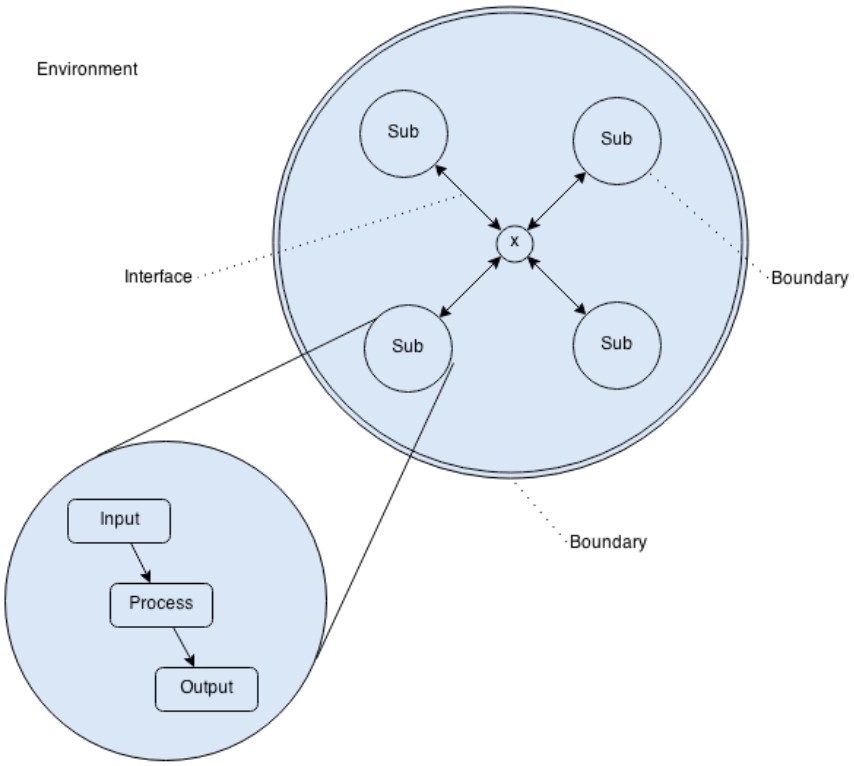
2.3.2. Karakteristik Sistem

Menurut penjelesan Jogiyanto, HM dalam buku *Pengenalan Komputer* [8], karaksteristik suatu sistem dapat merupakan bagian yang memiliki sifat ataupun perilaku tertentu, beberapa bagian diantaranya adalah :

1. *Component* merupakan suatu bagian yang merupakan pembentuk suatu sistem, bagian tersebut dinamakan sub, sub saling bekerja sama membentuk satu kesatun.
2. *Boundary* atau batas sistem merupakan daerah yang membatasi antara suatu sistem dengan hal selain dirinya.
3. *Environments* atau lingkungan luar merupakan hal apapun yang tidak merupakan bagian dari suatu sistem.
4. *Interface* atau penghubung merupakan suatu hubungan atau interaksi antara sub-sub dalam suatu sistem.
5. *Input* atau masukan adalah hal yang selanjutnya akan diolah dalam sub.
6. *Output* atau keluaran adalah hasil lain yang merupakan pengolahan dari suatu masukan.

7. *Process* atau pengolahan merupakan aktifitas yang dilakukan oleh sub dalam mengolah masukan, pengolahan suatu hal akan menghasilkan keluaran.
8. *Objective* atau sasaran sistem merupakan suatu tujuan yang diinginakan sebagai suatu tujan sebuah sistem.

Berikut gambaran umum tentang karakteristik sistem beserta interaksi yang dilakukannya, dapat dilihat pada **Gambar 2. 2**.



Gambar 2. 2. Karakteristik sistem

2.3.3. Definisi Data

Data menyatakan suatu kejadian, data juga merupakan bentuk mentah dari suatu informasi fakta yang dapat berupa teks, grafik, suara atau video.

Menurut Fred McFadden, dkk pada buku *Modern Database Management*, menyebutkan [9] :

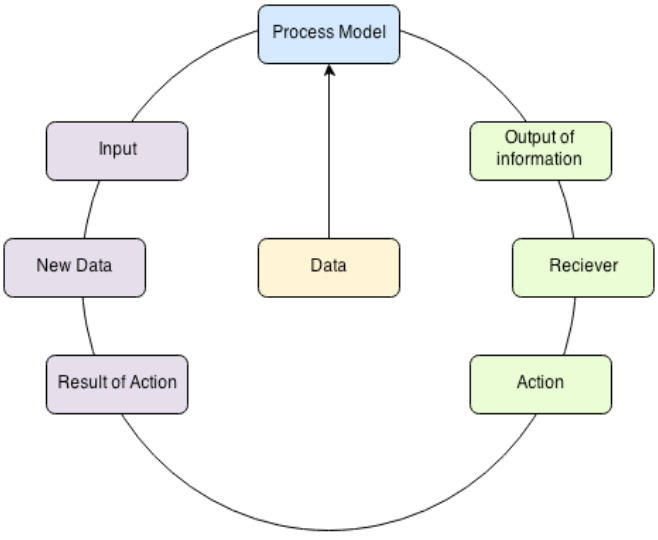
> "Data adalah fakta tentang sesuatu di dunia nyata yang dapat direkam dan disimpan pada media komputer."

2.3.4. Definisi Informasi

Informasi secara umum dapat didefinisikan sebagai sebuah pengetahuan yang berisi kumpulan pesan yang dapat ditransmisikan atau di rekam pada media penyimpanan, informasi sangat terkait dengan sekumpulan data yang telah diolah dengan benar. Menurut Gordon B. Davis menyebutkan pada buku *Kerangka Dasar Sistem Informasi Manajemen Bagian 1* [10] :

> "Informasi adalah data yang telah diolah menjadi suatu bentuk yang penting bagi penerima dan mempunyai nilai yang nyata atau yang dapat dirasakan dalam keputusan-keputusan yang sekarang atau keputusan yang akan datang".

Siklus informasi dimulai dari sekumpulan data yang diolah oleh suatu model proses sehingga menghasilkan informasi, penerima informasi menerima informasi lalu mengambil suatu keputusan untuk tindak-lanjutnya, keputusan penerima akan menghasilkan sebuah tindakan berupa data baru, sehingga data baru akan dianggap sebagai inputan baru yang selanjutnya akan kembali diolah oleh model proses.



Gambar 2. 3. **Siklus Informasi**

2.3.5. Sistem Informasi

Berdasarkan studi literatur terhadap buku *Analisis dan Desain Sistem Informasi* [13], sistem informasi dapat didefinisikan secara umum sebagai alat yang berfungsi mengolah informasi untuk mendukung operasional atau manajemen tertentu dalam aktifitas yang dilakukan oleh suatu organisasi dengan

bantuan teknologi informasi, sistem informasi juga dimanfaatkan sebagai alat kontrol manusia dalam aktifitas operasional.

2.4. Basis Data

Menurut studi literatur terhadap buku *Database System: Design, Implementation, and Managent* [5], basis data atau sering disebut sebagai *database* merupakan sekumpulan informasi yang terorganisasi dan tersimpan dalam media penyimpanan, basis data memanfaatkan suatu perangkat lunak untuk memanajemen atau mengoperasikan informasi yang terdapat didalamnya.

2.4.1. Sistem Basis Data

Menurut studi literatur terhadap buku *Database System: Design, Implementation, and Managent* [5], sistem basis data adalah sebuah sistem yang mengatur sekumpulan basis data, sekumpulan basis data yang merupakan subsistem itu sendiri di pakai oleh pemakai dengan tujuan mendapatkan informasi sesuai yang diperintahkan dengan menggunakan bantuan perangkat lunak untuk memanajemen.

2.4.2. Keuntungan Basis Data

Keuntungan basis data bedasarkan studi literatur *Database System: Design, Implementation, and Management* [5], secara umum teknologi ini dimanfaatkan untuk mengatasi permasalahan-permasalah sebuah data, berikut keuntungan yang dimilki basis data antara lain :

1. Mengatasi kerangkapan atau redudansi data.
2. Menghindari terjadinya ketidak-konsistensian data
3. Menyusun format yang standar dari sebuah data.
4. Penggunaan dapat dilakukan oleh banyak pengguna.
5. Untuk melakukan perlindungan dan pengamanan data dengan dilakukannya otorisasi pengguna (*DBA, Database Administrator*).
6. Integritas data dapat terpelihara, pengendalian data disusun agar sistem dapat bekerja sesuai aturan, dan batasan yang diinginkan.

7. Penciptaan abstraksi data (*View*) yang merupakan gabungan data yang bersifat hanya dapat dilihat (*read-only*).

2.4.3. *Entity Relationship Diagram* (ERD)

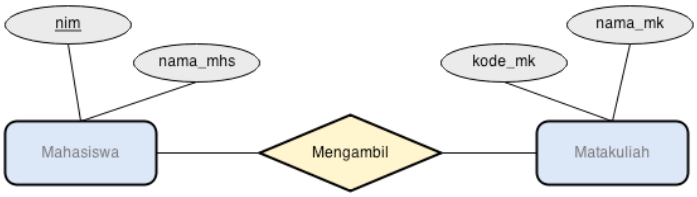
Entity Relationship Diagram (atau diagram E-R) menurut Ir. Fathansyah dalam buku *Basis Data* [11]. merupakan visualisasi hubungan antara data dalam sebuah basis data. Berikut elemen pembetuk suatu ERD :

1. *Entity* atau entitas digambarkan dengan persegi panjang yang merupakan sesuatu yang terdapat dalam sistem. Nama entitas dapat menggunakan kata benda, pelaku, atau tempat, ataupun suatu proses kegiatan.
2. *Relationship* atau relasi digambarkan dengan sebuah bentuk belah ketupat, relasi merupakan hubungan alamiah yang terjadi antara entitas, umumnya diberi nama dengan suatu kata kerja.
3. *Attribute* atau atribut merupakan elemen pembangun suatu entitas, entitas harus memiliki minimal satu atribut yang mencirikan suatu entitas menjadi berbeda dengan karakteristik entitas lainnya. Namun pada kasus tertentu atribut bisa saja digunakan sebagai elemen pembentuk suatu relasi.
4. *Cardinality* atau kardinalitas digunakan untuk menyatakan rasio suatu entitas terhadap entitas lainnya yang saling terkait dalam suatu relasi. Beberapa diantaranya jenis kardinalitas dalam suatu relasi, yaitu :
    a. *One to one* atau satu ke satu, merupakan rasio hubungan yang menyatakan bahwa satu kejadian pada entitas pertama hanya mempunyai satu hubungan dengan satu kejadian pada entitas yang kedua atau sebaliknya.
    b. *One to many* atau satu ke banyak, merupakan rasio hubungan yang menyatakan bahwa satu kejadian pada entitas dapat mempunyai banyak hubungan dengan kejadian pada entitas yang kedua.
    c. *Many to many* atau banyak ke banyak, merupakan rasio hubungan yang menyatakan bahwa satu kejadian pada sebuah entitas akan mempunyai banyak hubungan dengan kejadian pada entitas

lainnya, baik dilihat dari sisi entitas yang pertama maupun dilihat dari sisi yang kedua.

5. *Key* atau kunci merupakan atribut spesial yang nilainya mengidentifikasikan keunikan dalam suatu entitas, key pada sebuah atribut menggunakan garis bawah pada namanya. Umumnya suatu kunci memiliki beberapa jenis, yaitu *primary key* (utama), *foreign* key (tamu).

Berikut contoh penggunaan *Entity relationship diagram* untuk entitas mahasisawa terhadap matakuliah terdapat pada Gambar 2. 4.



Gambar 2. 4. Contoh penggunaan *ERD*

2.4.4. SQL

Berdasarkan studi literatur terhadap *Database System: Design, Implementation, and Managent*, SQL adalah sebuah bahasa permintaan database yang terstruktur, bahasa ini dibuat sebagai bahasa yang dapat merelasikan data baik dalam sebuah basis itu sendiri maupun di luar [5] dalam bentuk pemanggilan atau perintah (*query*). Bentukan perintah pada SQL terbagi menjadi tiga bagian, antara lain :

1. DDL (*Data Definition Language*)

DDL adalah sebuah metode perintah dalam SQL yang berguna untuk mendefinisikan data pada sebuah basis data, perintah yang dimiliki yaitu :

a. *CREATE* (penciptaan tabel dan database).
b. *DROP* (penghapusan tabel maupun database).
c. *ALTER* (perubahan struktur tabel yang telah dibuat). *Alter* atau perubahan dalam DML dapat berupa : Penambahan *field* (*add*), penggantian nama *field* (*change*), penamanaan ulang *field* (*rename*), penghapusan *field* (*drop*)

2. DML (*Data Manipulation Language*)

DML adalah sebuah metode perintah yang cakupannya hanya terhadap sekumpulan data. DML diamsumsikan dapat dilakukan jika DDL telah diciptakan, berikut contoh umun DML dalam SQL :

a. *CREATE* (pembuatan)
b. *INSERT* (penambahan).
c. *UPDATE* (peremajaan atau perubahan).
d. *DELETE* (penghapusan).

3. DCL (*Data Control Language*)

Data kontrol merupakan manajemen penggunaan seperti sistem keamanan otoritas seorang administrator dalam mengakses data, integritas pemeliharaan konsistensi, pengembalian data pada saat terjadi kesalahan pada pengolahan basis data atau perangkat keras yang digunakan.

2.4.5. NoSQL

NoSQL merupakan sebuah bahasa permintaan atau perintah untuk mengolah basis data. Berbeda dengan *SQL* yang hanya merupakan bahasa permintaan terstruktur, *NoSQL* memilki pesan yang mengartikan tidak hanya bahasa permintaan tersturktur, hal ini sering disebut sebagai *Not-only SQL*.

NoSQL sering digunakan dalam basis data berjenis *non*-relasional, meskipun begitu beberapa manajemen basis data memungkinkan melakukan relasional. *NoSQL* sampai saat ini masih dalam pengembangan karena basis data tipe *NoSQL* tidak memiliki aturan baku seperti basis data relasional. Basis data tipe *NoSQL* lebih diperuntukkan untuk memanajemen basis data dengan skalablitas data yang besar dan dipakai dalam pengembangan perangkat lunak berbasis web.

Menurut studi literatur terhadap *NoSQL Database* [3], istilah *NoSQL* pertama kali digunakan pada tahun 1998 oleh Carlo Strozzi untuk basis data relasional dengan menghilangkan *SQL*, lalu Eric Evans pada tahun 2009 kembali memperkenalkan *NoSQL* ketika Jon Oskarsson (salah satu developer *Last.fm*) mengatur acara untuk membahas pendistribusian basis data *open-source*.

Pada perkembangannya *NoSQL* memiliki banyak varian, *NoSQL* menawarkan fitur yang disebut *BASE* (*Basically Available, Soft state,* dan *Eventually consistent*), hal ini sangat bertentangan dengan konsep relasional basis data yang memiliki konsep *ACID* (*Atomic*, *Consistent*, *Isolate*, dan *Durability*).

Saat ini pengklasifikasian tentang *NoSQL* memiliki sudut pandang tertentu terhadap tujuan masing-masing, berikut contoh pengklasifikasian data model *NoSQL* menurut *Scofield* dan *Popescu* [3] :

1. *Key-value store*

Key-value store adalah tipe basis data yang berkonsep seperti sebuah larik (*array*) asosiatif, pengolahannya menggunakan pementaan atau yang disebut *array mapping* seperti halnya dalam suatu buku kamus. Manajemen basis data ini memungkinkan untuk melakukan pengurutan data dengan bantuan *key* yang dimiliki dalam tiap-tiap data, beberapa vendor lain menggunakan kemampuan *RAM* untuk melakukan penyimpanan data.

Berikut contoh sitem manajemen basis data yang cukup populer untuk jenis *Key-value* : *Dynamo, FoundationDB, MemcacheDB, Redis, Riak, FairCom c-treeACE*.

2. *Column store*

Column store memiliki sinonim dengan istilah *Wide-column store*. Tipe basis data ini menerapkan konsep penyimpannan data dengan dukungan menyimpan tabel data dalam sebuah *record table* (*column*), atau dapat diibaratkan sebuah *record* data bukan hanya dapat diisi dengan nilai yang konstan, melainkan dapat diisi dengan arsitektur sebuah tabel lainnya yang mana tabel tersebut dapat memiliki data juga (data tabel dalam *record*).

Berikut contoh sitem manajemen basis data yang cukup populer untuk jenis *Coulmn store* : *Accumulo, Cassandra, Druid, Hbase*.

3. *Document store*

Document store memiliki konsep yang menganggap bahwa data dapat diibaratkan seperti tumpukan dokumen yang dikelompkokkan dengan

penamaan yang disebut koleksi. *Document store* menerapkan konsep ini untuk format pengkodean atau *encoding* dari setiap data dalam sebuah informasi, umumnya format yang ditawarkan untuk jenis basis data ini menggunakan *XML*, *YAML*, *JSON* ataupun *BSON* (bentuk binari dari *JSON*).

Berikut contoh sitem manajemen basis data yang cukup populer untuk jenis *Document-store* : *Clusterpoint, Apache CouchDB, Couchbase, MarkLogic, MongoDB*.

4. *Graph data store*

Graph data store diciptakan berdasarkan teori keterkaitan variabel dalam sebuah bagan grafik, manajemen basis data ini dirancang untuk tumpukan data yang saling berelasi. Relasional data yang terdapat didalanya distimulasikan atau direpresentasikan dalam sebuah grafik yang saling mengaitkan data-data yang saling terhubung.

Berikut contoh sitem manajemen basis data yang cukup populer untuk jenis *Graph data store* : *Allegro, Neo4J, InfiniteGraph, OrientDB, Virtuoso, Stardog*.

Secara garis besar pengklasifikasian tentang manajemen basis data tipe *NoSQL* dapat dilihat pada Tabel 2. 1.

Tabel 2. 1. Perbandingan manajemen basis data NoSQL

Model Data	*Performance*	*Scalability*	*Fleksibelity*	*Complexity*	*Fungsionality*
Key-value store	*High*	*High*	*High*	*None*	*Variable (None)*
Column store	*High*	*High*	*Moderate*	*Low*	*Minimal*
Doument store	*High*	*Variable (High)*	*High*	*Low*	*Variable (Low)*
Graph data store	*High*	*High*	*High*	*Low*	*Graph Theory*

2.4.5.1. Pengolahan Data *Document Store* NoSQL MongoDB

Berdasarkan studi literatur [33] ditemukan penjelasan tentang pengolahan data *document store* NoSQL MongoDB, berikut perbandingan penggunaan dan pengoperasian bahasa perintah SQL RDBMS dan NoSQL MongoDB :

a. Perbandingan pengeksekusian *database server*, dapat dilihat pada Tabel 2. 2 :

Tabel 2. 2. *Database server* SQL RDBMS terhadap NoSQL MongoDB

Executeable	**MySQL**	**NoSQL MongoDB**
Server executeable	`mysqld`	`mongod`
Shell executeable	`mysql`	`mongo`

b. Perbandingan struktur atau aturan model, dapat dilihat pada Tabel 2. 3 :

Tabel 2. 3. Aturan model SQL RDBMS terhadap NoSQL MongoDB

SQL RDBMS	**NoSQL MongoDB**
Table	*Collection*
Row Data	*Document (Format BSON)*
Field / Column	*Field / Obejct keys*
Index	*Index*
Join table query	*Embeded & Reference (data modeling)*
Primary key	*Primary key (_id : <ObjectId>)*
Field / Column	*Field / Obejct keys*

c. Perbandingan operator yang digunakan dalam bahasa perintah (*query language*) dapat dilihat pada Tabel 2. 4 :

Tabel 2. 4. Operator SQL RDBMS terhadap NoSQL MongoDB

SQL RDBMS	**NoSQL MongoDB**	**Pelafalan**
`=`	`:`	*is*
`!=`	`$ne`	*negation equal*
`<`	`$lt`	*little than*
`>`	`$ge`	*greater than*
`<=`	`$lte`	*little than or equal*
`>=`	`$gte`	*greater than or equal*
`AND`	`$elemMatch`	*element match*
`IN`	`$in`	*in*
`NOT IN`	`$nin`	*not in*
`OR`	`$or`	*or*
`LIKE "%m"`	`: /^m/`	*regular expression*
`MOD`	`$mod`	*modular*
`NULL`	`null`	*null*
	`$type`	*field data type*
	`$size`	*(size / length) of value*
	`$exist`	*field exsist*
	`$inc`	*increment*
	`$set`	*setting field value*
	`$unset`	*unset field*
	`$push`	*append element to array*
	`$pop`	*remove array element*
	`$pull`	*remove a matching array element*

d. Perbandingan kata kunci secara umum untuk bahasa perintah (*query language*) dapat dilihat pada Tabel 2. 5 :

Tabel 2. 5. Kata kunci SQL RDBMS terhadap NoSQL MongoDB

SQL RDBMS	NoSQL MongoDB
ALTER	.update()
COUNT	.count()
CREATE INDEX ON	.ensureIndex()
CREATE TABLE	.createCollection()
DELETE	.remove()
DISTINCT	.distinct()
EXPALAIN	.explain()
FROM	
INSERT INTO	.insert()
LIMIT	.limit()
ORDER BY *[field]* ASC	.sort(*[direction]*)
ORDER BY *[field]* DESC	.sort(*[direction]*)
SELECT	.find()
SET	
SKIP	.skip()
UPDATE	.update()
WHERE	

e. Contoh perbandingan penggunaan bahasa perintah (*query language*) dapat dilihat pada Tabel 2. 6 :

Tabel 2. 6. Contoh *query* SQL RDBMS terhadap NoSQL MongoDB

SQL RDBMS	NoSQL MongoDB
CREATE INDEX ON users (name ASC, age DESC)	db.users.ensureIndex({name: 1, age: −1})
CREATE INDEX ON users (name ASC)	db.users.ensureIndex({name: 1})
CREATE TABLE users (name VARCHAR(128), age NUMBER)	db.createCollection("users")
DELETE FROM users WHERE name = 'Bob'	db.users.remove({name: "Bob"})
EXPLAIN SELECT ∗ FROM users WHERE age = 32	db.users.find({age: 32}).explain()
INSERT INTO users VALUES ('Bob', 32)	db.users.find({age: 33}, {name: 1, age: 1, _id:0})
SELECT ∗ FROM users	db.users.insert({name: "Bob", age: 32})
SELECT ∗ FROM users LIMIT 1	db.users.findOne()
SELECT ∗ FROM users LIMIT 10 SKIP 20	db.users.find().skip(20).limit(10)
SELECT ∗ FROM users ORDER BY name DESC	db.users.find().sort({name: −1})
SELECT ∗ FROM users WHERE age <= 33	db.users.find({}, {name: 1, age: 1, _id:0})
SELECT ∗ FROM users WHERE age = 32 AND name = 'Bob'	db.users.find({age: 32, name: "Bob"})
SELECT ∗ FROM users WHERE age = 33 OR name='Bob'	db.users.find({$or:[{age:33}, {name: name = 'Bob' "Bob"}]})
SELECT ∗ FROM users WHERE age = 33 ORDER BY name ASC	db.users.find({age: 33}).sort({name: 1})

SELECT * FROM users WHERE age > 33	db.users.find({age: {$gt: 33}})
SELECT * FROM users WHERE age > 33 AND age < 40	db.users.find({age: {$gt: 33, $lt: 40}})
SELECT * FROM users WHERE name LIKE '%Joe%'	db.users.find({name: /Joe/})
SELECT * FROM users WHERE name LIKE 'Joe%'	db.users.find({name: /^Joe/})
SELECT COUNT(*) FROM users	db.users.count()
SELECT COUNT(*) FROM users WHERE AGE > 30	db.users.find({age: {$gt: 30}}).count()
SELECT COUNT(AGE) FROM users	db.users.find({age: {$exists: true}}). count()
SELECT DISTINCT name FROM users	db.users.distinct("name")
SELECT name, age FROM users	db.users.find({age: {$lte: 33}})
SELECT name, age FROM users WHERE age = 33	db.users.find()
UPDATE users SET age = 33 WHERE name = 'Bob'	db.users.update({name: "Bob"}, {$set: {age: 33}}, {multi: true})
UPDATE users SET age = age + 2 WHERE name = 'Bob'	db.users.update({name: "Bob"}, {$inc: = 'Bob' {age: 2}}, {multi: true})

2.4.5.2. *References Data Model* pada *Document Store* NoSQL MongoDB

References atau data linking merupakan bentuk pemodelan data dengan bentuk normal, dalam konsep basis data non-relasional *NoSQL* yang berorientasi *document store* model ini memiliki arti bahwa data dikelompokkan berdasarkan entitasnya dalam suatu *collection* [33]. Bentuk pemodelan ini mereferensikan *primary key* pada suatu *collection* ke *collection* lain yang merupakan korespondesi data dari *collection* yang terkait, konsep ini memungkinkan abstraksi relasi dengan adanya penggunaan primary key dari suatu induk collection yang bertamu pada *collection* lain [33], berikut contoh pemodelan dengan metode *references* :

```
//Collection tipe_kontak
{
    _id  : “tipekontak001”,
    tipe : “seluler”
},
{
    _id  : “tipekontak002”,
    tipe : “mail”
}
```

```
//Collection konsumen
{
    _id    : “konsumen001”,
    nama   : “Nama orang”,
    umur   : 17,
    alamat : “Jalan Cikutra 07”,
    kota   : “Bandung”,
    status : “pelajar”
}
```

```
//Collection kontak_konsumen
{
    tipeKontak_id : “tipekontak001”,
    kontak_id     : “konsumen001”,
```

```
    nilai         : “0899xxxx”
},
{
    tipeKontak_id : “tipekontak002”,
    kontak_id     : “konsumen001”,
    nilai         : “contoh@mail.com”
}
```

Sebagai jenis pemodelan, *references* memiliki kearakteristik dalam non-relasional NoSQL MongoDB, berikut karakterstik yang dimaksud adalah :

1. Pemodelan ini akan optimal untuk jenis *collection* yang sering mengalami penambahan data.
2. Redudansi data dapat diminimalisir dikarenakan metode ini seperti model relasional RDBMS pada umumnya.
3. Dapat dibuat kostumisasi fungsi sesuai kebutuhan dan keahlian programing seorang DBA untuk menerapkan *constrain* penggunaan *foreign key*.
4. Model ini membutuhkan ukuran penyimpanan yang lebih kecil dibandingkan metode permodelan *embedded*.
5. Relasional 1 ke N akan mudah dengan menggunakan konsep larik, hal ini karena data yang berbeda dalam suatu *field* dapat diringkas dalam suatu *field array*, sehinga tidak membutuhkan baris data baru untuk penggunaan *key* dengan jumlah kardinalitas yang lebih dari satu.
6. Pembacaan data akan sedikit lambat karena harus menggunakan fungsi buatan guna men-*join* (dengan menggunakan konsep *map, reduce*) dalam pemanggilan data relasi karena MongoDB tidak memiliki standar dalam penyelesaian kasus ini.
7. Untuk menghasilkan data yang atomik pada penambahan data ataupun peremajaan, seorang DBA harus mendefinisikan fungsi untuk mengatasi kendala tersebut.

2.4.5.3. *Embedded Data Model* pada *Document Store* NoSQL MongoDB

Embedded data merupakan permodel data dalam bentuk *denormalized* (tidak ternormalisasi), hal ini mirip seperti konsep *view* atau abstraksi dari himpunan data secara keseluruhan dalam *RDBMS* pada umumnya [33]. *Denormalized* dalam konsep basis data non-relasional *NoSQL* yang berorientasi

document store mengganggap suatu *collection* dalam basis data memiliki hirarki anak yang merupakan hirarki sebuah data seperti baris dari *collection* lain yang berhubungan secara teknis dengan baris data yang induknya [33]. Berikut contoh pemodelan data yang dilakukan dengan menggunakan k1onsep *embedded*.

```
{ //Collection tipe_kontak
    _id  : “tipekontak001”,
    tipe : “seluler”
},
{
    _id  : “tipekontak002”,
    tipe : “mail”
}
```

```
{ //Collection konsumen
    _id    : “konsumen001”,
    nama   : “Nama orang”,
    alamat : “Jalan Cikutra 07”,
    kota   : “Bandung”,
    status : “pelajar”
}
```

```
{ //Collection kontak_konsumen
    _id    : “konsumen001”,
    nama   : “Nama orang”,
    alamat : “Jalan Cikutra 07”,
    kota   : “Bandung”,
    status : “pelajar”
    kontak : {
       tipe : “selluler”,
       nilai: “0899xxxx”
    }
}
```

Seperti halnya pemodelan data jenis *references*, pemodelan jenis *embedded* pun memiliki karakteristik dalam non-relasional NoSQL MongoDB, dalam hal ini sebagai berikut :

1. Pemodelan ini akan optimal untuk jenis *collection* yang jarang mengalami penambahan data namun sering digunakan (pembacaan data).
2. Fleksibilitas data, pemodelan ini tidak memerlukan data master karena data master akan terkoleksi sendiri, tanpa harus mendefinisikan.
3. Pengunaan banyak *collection* pada jenis penggunaan pemodelan ini tidak menguntungkan karena secara bentuk pemodelan informasi yang terdapat dalam sebuah baris data terhimpun penuh, sehingga menyebabkan penerapan terhadap pemodelan ini harus jelas, misalnya untuk data transaksi saja.

4. Data master yang tidak pernah digunakan akan hilang dikarenakan *key* pada data tersebut tidak pernah digunakan pada baris data pada entitas lain.
5. Baris data akan sangat banyak, bertingkat-tingkat, sehingga akan sulit membaca ataupun mengolah data.
6. Konsistensi terhadap data begitu rapuh, hal ini dikarenakn tidak ada penggunaan *foreign key*.
7. Redudansi akan begitu banyak dijumpai di banyak baris data.
8. Akibat poin sebelumnya, ukuran database akan membengkak, seiiring bertambahnya data yang beranak-induk.

2.5. Object Oriented Programming

Object oriented programming (OOP), merupakan paradigma pemrograman dengan orintasi objek. Pemrograman berorientasi objek pertama kali diperkenalkan di MIT pada akhir 1950-an dan awal 1960-an [31].

Konseptual OOP diciptakan untuk memudahkan proses *development* sebuah program diwaktu yang akan datang. Aspek terpenting dalam OOP adalah penurunan sifat (atribut) dan tingkah laku (*behaviour* atau disebut metode) yang dapat dilakukan sebuah objek, beberapa elemen penting dalam OOP adalah :

1. *Interface* merupakan *skleton* (*implement*) dari sebuah *class*, *Interface* hanya memiliki atribut dan motode yang tidak termasuk isinya (hanya definisi nama). Ketika sebuah *class* mengimplentasi sebuah *interface* maka *class* tersebut harus mendifinisikan ulang setiap metode yang dimiliki *interface*-nya, sedangkan *Interface* tidak dapat digunakan untuk menginstasiasi *object.*
2. *Class* merupakan *blueprint* atau prototipe atau cetakan untuk membuat (instansiasi) sebuah objek. *Class* memiliki atribut yang merupakan sifat dan metode yang merupakan tingkah laku. Sebuah *class* dapat melakukan penurunan atau pewarisan (*Inheritence*) sifat dan metode pada dalam pembuatan *class* lain. *Class* juga dapat menginstansiasi dirinya dalam pembuatan objek baru dengan kata kunci *new*.

3. *Abstract class* merupakan abstraksi umum sebuah *class*, *abstract class* memiliki atribut dan metode umum yang seharusnya dimiliki pada turunan *class* anak dan *object*.
4. *Object* merupakan instansiasi yang berasal dari blueprint suatu *class*, *object* akan selalu memiliki sifat dasar (atribut) dan tingkah laku (metode) yang dimiliki *blueprint* sebuah induk (*class* atau *abstract class*).
5. *Inheritence* merupakan sebuah proses penurunan atau pewarisan sifat saat pembuatan (instansiasi) objek baru.
6. *Encapsulation* merupakan sebuah aturan yang memungkinkan aksesibelitas sebuah *class*, atribut ataupun metode. Ada tiga macam enkasulasi dalam pemrograman berorientasi objek pada umumnya : yaitu *public, protected,* dan *private*. Tipe *public* dapat digunakan bebas tanpa harus melalui proses *inheritence*, sedangkan *protected* hanya dapat digunakan ketika menggunakan konsep *inheritnence* dan instansiasi. Tipe *private* hanya dapat digunakan untuk kelas itu sendiri.
7. *Polymorphism* merupakan pendefinisian ulang sebuah metode atau atribut yang berasal dari induknya (*Interface, class* atau *abstract class*). Ada dua macam pendefinisian ulang yaitu :
    a. *Override* yang merupakan pendefinisian ulang suatu metode oleh *class* anak. Syarat *override* yaitu nama metode, tipe return, dan parameter harus sama. Jika tidak sama maka bukan dianggap sebagai override tetapi metode yang baru pada class anak.
    b. *Overload* yang merupakan pendefinisian ulang suatu metode dalam class yang sama. Syarat *overload* yaitu nama metode dan tipe parameter harus berbeda dalam class yang sama.

2.5.1. *Javascript Design Pattern*

Javascript bukan merupakan sebuah bahasa pemrograman melainkan hanya merupakan skrip untuk memerintahkan sesuatu, *javascript* umumnya digunakan untuk memerintah objek HTML, namun pada pengembangannya *javascript* kini

digunakan berbagai macam perintah lain. Dalam skrip setiap fungsi (metode) merupakan sebuah variabel yang perisi perintah, sedangkan variabel non-fungsi merupakan nilai tetap atau konstansa.

```
function iniFungsi1 (parameter) {}; //cara pertama membuat fungsi
var iniFungsi2 = function (parameter) {}; //cara kedua membuat fungsi
var hanyaVariabel = 1;
```

Sebuah skrip umumnya tidak memilki pradigma pemrograman *based-class language*, melainkan hanya merupakan *prototype-based language.* Istilah *prototype-based language* digunakan untuk meniru pradigma pemrograman berorientasi objek [31]. Berikut perbedaan yang dimiliki *javascript prototype-based language* dengan *based-class language* pada umunya yaitu :

1. *Interface. Javascript* tidak memilki sistem untuk membuat sebuah *interface*, namun dapat dibuatkan sebuah kelas untuk validasi sebuah *interface* [32]:

```
var Interface = function (name, methods) {
  if (arguments.length != 2) {
    throw new Error(
      "Interface constructor called with " + arguments.length +
      "arguments, but expected exactly 2."
    );
  }
  this.name = name;
  this.methods = [];
  for (var i = 0, len = methods.length; i < len; i++) {
    if (typeof methods[i] !== 'string') {
      throw new Error(
        "Interface constructor expects method names to be " +
        "passed in as a string."
      );
    }
    this.methods.push(methods[i]);
  }
};
```

```
// Static class method.
Interface.ensureImplements = function (object) {
  if (arguments.length < 2) {
    throw new Error(
      "Function Interface.ensureImplements called with " +
      arguments.length + "arguments, but expected at least 2."
    );
  }
  for (var i = 1, len = arguments.length; i < len; i++) {
    var interface = arguments[i];
    if (interface.constructor !== Interface) {
      throw new Error(
        "Function Interface.ensureImplements expects arguments" +
        "two and above to be instances of Interface."
      );
    }
    for (j=0;methodsLen=interface.methods.length;j<methodsLen; j++) {
      var method = interface.methods[j];
      if (!object[method] || typeof object[method] !== 'function') {
        throw new Error(
```

```
          "Function Interface.ensureImplements: object " +
          "does not implement the " + interface.name +
          " interface. Method " + method + " was not found."
        );
      }
    }
  }
};
```

Dikarenakan *Interface* hanya merupakan definisi sebuah atribut dan motode, sehingga untuk membuat sebuah *interface* dalam *javascript* dapat dilakukan dengan :

```
var Composite = new Interface('Composite', ['add', 'remove']);
var FormItem = new Interface('FormItem', ['save']);
```

2. *Class.* Dalam *javascript*, *class* merupakan sebah fungsi dengan paradigma prototipe yang digunakan untuk membuat metode atau atribut, sebagi contoh pembuatan kelas dalam *javascript* adalah :

```
var Kelas1 = function () {
  //enkasulasi tipe private
  var privateAttr = "hello";
  var privateFunc = function (parameter) {};

  //enkasulasi tipe protected
  this.protectedAttr = 1;
  this.protectedFunc = function (parameter) {};
};

//enkasulasi tipe public
Kelas1.prototype.publicAttr = 2;
Kelas1.prototype.publicFunc = function (parameter) {};
```

Atribut `publicAttr` dapat dipanggil menggunakan `Kelas1.prototype.publicAttr`, begitu juga untuk metode `publicFunc`.

Sedangkan `protectedAttr` dan `protectedFunc` hanya dapat digunakan ketika mendefiniskan objek baru (instansiasi objek).

```
var objek = new Kelas1();
```

3. *Abstract class.* Istilah *abstract class* tidak dimiliki oleh *javascript*, pada penjelasan sebelumnya, *abstract class* merupakan sebuah kelas yang memiliki deskripsi umum yang akan digunakan untuk objek atau *sub-class*, sehingga pembuatan *abstract class* dalam *javascript* tidak berbeda seperti pembuatan *class* seperti biasanya dalam *javascript*.
4. *Object. Object* dalam *prototype-based language javascript* merupakan sebuah variabel yang merupakan instasiasi dari suatu kelas, seperti :

```
var objek = new Kelas1();
```

5. *Inheritence. Inheritence* dalam *javascript akan* selalu dilakukan dalam setiap instansiasi objek, objek yang merupakan instansi *class* memiliki dan dapat digunakan ulang untuk semua sifat atau tingkah laku umum (*public encapsulation*) yang dimilki *class* asalnya, sedangkan untuk *private encapsulation* tidak dapat digunakan ulang dalam instansiasi objek baru.
6. *Encaptulation. Encaptulation* dalam penjelasan sebulumnya memiliki tiga tipe yaitu *public*, *protected*, dan *private*. Berikut contoh *encapsulation* dalam *javascript* :

```
var Kelas1 = function () { //Kelas1 akan selalu bersifat public
//enkasulasi tipe private
  var privateAttr = "hello";
  var privateFunc = function (parameter) {};

  //enkasulasi tipe protected
  this.protectedAttr = 1;
  this.protectedFunc = function (parameter) {};
};

//enkasulasi tipe public
Kelas1.prototype.publicAttr = 2;
Kelas1.prototype.publicFunc = function (parameter) {};
```

percobaan untuk *encaptulation* tipe *private* :

```
//tanpa instansiasi
Kelas1.prototype.privateAttr;   //undefined!
Kelas1.prototype.privateFunc;   //undefined!

//dengan instansiasi
new Kelas1().privateAttr;       //undefined!
new Kelas1().privateFunc;       //undefined!
```

percobaan untuk *encaptulation* tipe *protected* :

```
//tanpa instansiasi
Kelas1.prototype.protectedAttr; //undefined!
Kelas1.prototype.protectedFunc; //undefined!

//dengan instansiasi
new Kelas1().protectedAttr;     //2
new Kelas1().protectedFunc;     //function (parameter) {};
```

percobaan untuk *encaptulation* tipe *public* :

```
//tanpa instansiasi
Kelas1.prototype.publicAttr;    //1
Kelas1.prototype.publicFunc;    //function (parameter) {}

//dengan instansiasi
new Kelas1().publicAttr;        //1
new Kelas1().publicFunc;        //function (parameter) {}
```

8. *Polymorphism*

   Polymorphism dalam *javascript* tidak memilki konsep *override* atau *overload* seperti penjelasan *polymorphism* pada umumnya,

polymorphism dalam *javascript* dapat dilakukan pada atribut atau metode yang bersifat *public* dan *protected*. Ketika sebuah objek diinstasiasikan dari sebuah kelas, maka untuk pendefinisian ulang akan dilakukan sangat bebas yaitu objek akan merubah keseluruhan atribut atau metode yang dimiliki *class* induk menjadi pendefinisian baru yang diinginkan. Berikut contoh :

```
var object = new Kelas1();

//bukan polimorphysm karena tidak mempunyai nilai awal
object.privateAttr = "this is not";
object.privateFunc = function () {};

//polimorphysm, pendefinisian ulang.
object.protectedAttr = "this is yep!";
object.protectedFunc = "this is yep too!";
object.publicAttr = "also, this is yep bro!";
object.publicFunc = "yep, we are!";
```

2.6. Unified Modeling Language (UML)

Unified Modeling Language merupakan bahasa standar yang bekerja dalam *object-oriented* untuk menentukan permodelan perangkat lunak dengan memrepresentrasikan atau memvisualisasikannya dalam bentuk diagram. *UML* mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar pemrograman berorientasi objek sejak tahun 1980-an. Saat ini versi stabil UML adalah 2.4 dan sampai sekarang *UML* masih dalam pengembangan. *UML* juga merupakan dasar dari desain sebuah perangkat lunak yang dibangun dengan konsep pembangunan berorientasi objek yang direkomendasikan IBM dalam *UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition)* [29].

2.6.1. Klasifikasi UML

Klasifikasi *UML* dibagi menjadi dua, yaitu berdasarkan bentuk struktur (*structure diagrams*) dan bentuk perilaku atau karakteristiknya (*behavior diagrams*) [30]:

2.6.1.1. *Structure diagrams*

Structure diagrams menyatakan struktur statis dari suatu hubungan antara sistem dan bagiannya pada tingkat pelaksanaan. Unsur yang terdapat di dalam

Structure diagrams mewakili konsep atau makna tertentu dari suatu sistem, mungkin termasuk abstraksi sebuah konsep pelaksanaan. Salah satu yang temasuk umum dalan jenis struktur digram dari *UML* adalah *Class diagram* [30].

Class diagram adalah struktur dari suatu sistem pada tingkat pengklasifikasian (kelas, interface, dll). Diagram ini menunjukkan beberapa pengklasifikasian seperti hubungan dan operasi antara atribut subsistem atau komponen.

2.6.1.2. *Behaviour diagrams*

Behaviour diagrams menyatakan suatu perilaku objek secara dinamis atau karakteristik objek dalam perangkat lunak suatu sistem untuk menyatakan perubahan dari waktu ke waktu. Pada umumnya yang termasuk dalam jenis struktur digram dari *UML* adalah [30]:

1. *Use case diagram* adalah diagram *UML* yang menggambarkan tertekaitan atau hubungan antara aktor dengan system.
2. *State machine* pada versi UML sebelumnya dikenal dengan *state chart*. Diagram ini digunakan untuk memodelkan transisi atau perubahan perilaku dari keadaan satu ke keadaan lainnya.
3. *Activity diagram* merupakan diagram yang memodelkan perilaku *Use Case* dan objek di dalam suatu sistem, diagram ini juga dapat digunakan untuk menjelaskan proses bisnis dan alur kerja operasional suatu sistem.
4. *Interaction diagram* adalah diagram yang menjelaskan interaksi yang terjadi dalam sistem, beberapa jenis *interaction diagram* antara lain :
   a. *Sequence diagram*

      Sequence diagram digunakan untuk memodelkan pengiriman pesan (*message*) antar objek, dan juga digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respon dari sebuah kejadian (event) untuk menghasilkan *output* tertentu.

   b. *Communication diagram*

Communication diagram atau lebih dikenal sebagai *Collaboration digram* merupakan pemodelan interaksi antar objek. Diagram ini mirip seperti *sequence diagram*, tetapi lebih menekankan pada peran yang dimiliki suatu objek, dan bukan pada *timing* suatu kejadian (*event*) untuk menyampaikan pesan (*message*).

2.7. *Tools*

Tools atau alat adalah sesuatu benda nyata (ataupun tidak) yang berguna untuk membantu suatu pekerjaan, kegiatan. Berikut *tools* yang digunakan dalam penelitian ini adalah sebegai berikut :

1. *Command Line Shell*

Menurut *Wikipedia* [14], *Command line shell* adalah bentuk antar-muka pengguna dalam bentuk konsol text yang berguna untuk memberi perintah langsung terhadap sistem operasi.

2. Sistem Operasi

Sistem operasi adalah sebuah sistem inti dengan bentuk perangkat lunak yang memanajemen perangkat keras dan menjadi pondasi atau acuan untuk perangkat lunak kecil lain yang beroperasi didalamnya [15].

3. *Text Editor*

Text Editor adalah perangkat lunak komputer yang berguna untuk mengolah *file* dengan format teks.

4. *World Wide Web* (WWW atau W3)

World wide web atau sering disebut *web*, adalah sebuah sistem yang menghubungkan dokumen *hypertext* yang diakses menggunakan *web client* dengan bantuan Internet [18].

5. *Web Client*

Web client adalah sebuah perangkat lunak yang erat hubungannya dengan jaringan komputer, salah satu contoh bentuk umum dari *web client* adalah *web browser* (*seperti : Mozzilla Firefox, Google Chrome, Internet*

Explorer, dll). *Web client* dapat mengakses dokumen yang tersedia dalam *web server* melalui jaringan komputer atau *internet*.

6. *Web Server*

Web server adalah sebuah perangkat lunak yang menyediakan konten yang dapat diakses oleh *web client*, *web serv*er tidak akan memberi konten cuma-cuma tanpa adanya permintaan dari *web client*, secara simultan *web server* akan me-*respons* hal yang di-*request* oleh *web client (request - response)*.

7. *Hypertext Markup Languange* (HTML)

HTML adalah bahasa markah (*markup*) yang digunakan untuk mengakses file teks berformat tertentu pada halaman *World Wide Web* [16].

8. *Cascading Markup Language* (CSS)

CSS adalah suatu bahasa yang digunakan untuk mengatur tampilan suatu dokumen yang ditulis dalam HTML. Salah satu tujuan dari CSS adalah untuk memisahkan presentasi dari sebuah dokumen dari struktur dan isi, spesifikasi CSS diatur oleh W3C, dan menjadikan rekomendasi untuk mengatur gaya visual pada halaman *World Wide Web* [17].

9. *JavaScript* (JS)

JavaScript merupkan bahasa populer dengan gaya *scripting*, yakni bahasa pemrograman yang mudah untuk dipahami dan sangat ringan, *javascript* tidak memerlukan proses *compiling* karena secara mendasar *javascript* dapat otomatis berjalan di jenis *web cilent*, *javascript* digunakan untuk memberi perintah terhadap terhadap *web client*.

10. *Extensible Markup Language* (XML)

XML merupakan sebuah bahasa markup untuk pemdeskripsian data. *XML* merupakan versi mini dari *SGML* (*Standard Generalized Markup Language*).

SGML merupakan aturan strandar dalam pembuatan tag dalam format dokumen, ISO 8879:1986. *SGML* banyak dipakai untuk memanjemen dokumen dalam jumlah besar, frekuensi revisi tinggi dan dibutuhkan dalam beragam format.

11. *Javascript Object Notation* (JSON)

JSON adalah suatu teks atau file teks dengan format *.json*. *JSON* dipopulerkan oleh *Douglas Crockford* dengan tujuan agar mudah terbaca oleh manusia. Tujuan utama *JSON* digunakan untuk merepresentasikan struktur data sederhana dengan larik (*array*) asosiatif (yang disebut tipe data objek dalam *javascript*).

JSON juga sering digunakan untuk mentransmisikan data, pengaplikasian utamanya adalah pada pemrograman aplikasi *web,* yakni *AJAX* (*Asyncronous Javascript XHR*) yang berperan sebagai alternatif terhadap penggunaan tradisional format XML [23].

12. HTTP dan HTTPS

HTTP atau *Hypertext Transfer Protocol* adalah protokol internet yang berjalan pada layer atau level aplikasi yang berguna untuk pendistribusian, pengkolaborasian jenis dokumen *hypermedia*, *HTTP* telah digunakan sejak 1990 dalam *world wide web* [20].

Sedangkan *HTTPS* atau *Hypertext Transfer Protocol Secure* hanya merupakan bentuk tertutup dari HTTP dengan kelebihan pengenkripsian data. *HTTP* menetapkan sembilan metode (kadang disebut "*verbs*") yang mengartikan tindakan yang ingin dilakukan terhadap *web server* [20].

a. *HEAD :* Meminta tanggapan yang identik dengan tanggapan yang sesuai dengan permintaan *GET*, namun tanpa badan tanggapan. Ini berguna untuk mengakses informasi *meta* yang tertulis dalam kepala tanggapan tanpa perlu mengangkut seluruh konten.
b. *GET :* Meminta representasi sumber tertentu. Permintaan menggunakan GET (dan beberapa metode *HTTP* lain) tidak boleh memiliki kepentingan melakukan tindakan selain pengaksesan.

c. *POST :* Mengirimkan data untuk diproses (misalnya dari bentuk *HTML*) ke sumber teridentifikasi. Data dimasukkan dalam badan permintaan. Ini dapat menghasilkan pembentukan sumber baru atau pemutakhiran sumber yang sudah ada atau keduanya.
d. *PUT :* Meng-upload representasi ke sumber tertentu.
e. *DELETE :* Menghapus sumber tertentu.

13. XML HTTP Request (XHR)

Menurut Wikipedia [21], *XMLHttpRequest* adalah *API* (*Application Programming Interface*) yang digunakan *web client* dengan menggunakan perintah yang dijalankan oleh *javascript*, digunakan untuk mengirimkan permintaan melalui jalur *HTTP*/*HTTPS* kepada *web server* dan memuat data respon dari *web server* yang diterima kembali oleh *javascript*. Interprtasi data yang paling umum proses XHR dapat berupa *XML*, *JSON* atau teks.

14. Node.js

Node.js pertama kali dibuat oleh *Ryan Dahl* pada tahun 2009 yang kemudian berkembang pesat di bawah licensi *Open Source MIT* oleh sebuah perusahaan bernama *Joyent Inc* [24] [25].

Node.js dikembangkan berdasarkan teknologi *Google V8 JavaScript engine* serta berisi kompilasi skrip dan banyak modul siap pakai (disebut *NPM*) yang bermanfaat, sehingga pengguna tidak perlu melakukan *coding* dan mendesainnya dari awal [24].

Node.js menggunakan *javascript* sebagai bahasa pemrogaman dan dapat dengan mudah menghasilkan *throughput* / pemrosesan tingkat tinggi melalui *non-blocking I/O*, *Node.js* juga memiliki fitur built-in *HTTP server library* yang menjadikannya mampu menjadi sebuah *web server* tanpa bantuan perangkat lunak lainnya seperti *Apache* atau *Nginx* [24].

15. Node.js *Package Manager* (NPM)

NPM merupakan *default package manager* atau modul pemrograman (dapan juga disbut sebagai *API*) yang digunakan *Node.js* sejak versi 0.6.3,

NPM sudah diikutsertakan secara *default* ketika proses instalasi *Node.js* [26].

16. MySQL

MySQL merupakan sebuah perangkat lunak sebagai penyedia layanan basis daya guna memanajemen atau mengolah data dengan menggunakan bahasa perintah terstruktur (*SQL*). *MySQL* memiliki dua bentuk lisensi, yaitu gratis dan *shareware*. *MySQL* lisensi gratis berada dibawah lisensi *GNU* / *GPL* (*General Public Lincense*) [27].

17. MongoDB

MongoDB pertama kali dikembangkan oleh perusahaan asal New York, *10gen* (kini disebut *MongoDB Inc*.) pada bulan Oktober 2007 sebagai bagian dari *platform* yang awalnya direncanakan sebagai produk jasa, namun kini perusahaan tersebut bergeser ke model pembangunan *open-source* pada tahun 2009 dengan meirilis di bawah kombinasi dari *GNU Affero General Public License* (AGPL) dan *Lisensi Apache*. Meskipun begitu, *10gen* tetap menawarkan dukungan komersial dan layanan lainnya [28].

Berbeda dengan jenis manajemen basis data lainnya, *MongoDB* merupakan sebuah perangkat lunak sebagai penyedia layanan basis data guna memanajemen atau mengolah data dengan *mengacu* pada konsep bahasa perintah terstruktur, atau sering diklasifikasikan sebagai basis data "*NoSQL*", yang memiliki arti *not-only SQL*.

MongoDB menghindari struktur basis data relasional, dan menggantinya dengan teknologi teks tradisional yang mendukung *JSON*, struktur yang dipakai *MongoDB* adalah *BSON* atau *Binnary JSON* [28].

MongoDB memiliki konsep basis data dengan berorientasi dokumen, artinya pola data yang terdapat didalam basis data ini akan berpola seperti kumpulan dokumen, hal ini dapat diibaratkan dengan suatu ruangan tertentu yang terdapat banyak rak buku, dengan rak buku terdapat barisan buku yang

tersusun dalam sub-raknya masing-masing, buku yang terdapat didalamnya dapat dianggap suatu data.

MongoDB dapat dianggap tidak memiliki skema / pola khusus dalam pengolahan basis datanya, hal ini disebabkan karena fleskibelitas yang dimilikinya.