BAB 2
LANDASAN TEORI

2.1. *Structured Query Language*

SQL atau juga sering disebut sebagai *query* merupakan suatu bahasa (*language*) yang digunakan untuk mengakses *database*. SQL dikenalkan pertama kali dalam IBM pada tahun 1970 dan sebuah standar ISO dan ANSII ditetapkan untuk SQL. Standar ini tidak tergantung pada mesin yang digunakan (IBM, Microsoft atau Oracle). Hampir semua *software database* mengenal atau mengerti SQL. Jadi, perintah SQL pada semua software database hampir sama [6]. Terdapat 3 (tiga) jenis perintah SQL, yaitu:

1. DDL atau *Data Definition Language*

   DDL merupakan perintah SQL yang berhubungan dengan pendefinisian suatu struktur *database*, dalam hal ini *database* dan tabel. Beberapa perintah dasar yang termasuk DDL ini antara lain CREATE, ALTER, RENAME, dan DROP.
2. DML atau *Data Manipulation Language*

   DML merupakan perintah SQL yang berhubungan dengan manipulasi atau pengolahan data atau *record* dalam tabel. Perintah SQL yang termasuk dalam DML antara lain SELECT, INSERT, UPDATE, dan DELETE.
3. DCL atau *Data Control Language*

   DCL merupakan perintah SQL yang berhubungan dengan manipulasi *user* dan hak akses (*privileges*). Perintah SQL yang termasuk dalam DCL antara lain GRANT dan REVOKE.

2.1.1. *Data Definition Language*

Data Definition Language (DDL) pada basis data merupakan perintah SQL yang berhubungan dengan pendefinisian, pengubahan dan penghapusan suatu struktur basis data, dalam hal ini basis data dan tabel. Contoh penerapan DDL adalah sebagai berikut.

1. Membuat dan menghapus basis data
   a. Membuat basis data

      Sintak umum SQL untuk membuat suatu *database* dapat dilihat pada Gambar 2.1.

```
CREATE DATABASE [IF NOT EXISTS] nama_database;
```

Gambar 2.1 Sintak Umum Pembuatan Basis Data

Perintah pada gambar 2.1 akan membuat basis data baru dengan nama "nama_database". Aturan penamaan sebuah basis data sama seperti aturan penamaan sebuah *variabel*, dimana secara umum nama basis data boleh terdiri dari huruf, angka dan *under-score* (_). Jika basis data yang akan dibuat sudah ada, maka akan muncul pesan *error*. Namun jika ingin otomatis menghapus basis data yang lama jika sudah ada, maka diikuti perintah **IF NOT EXISTS** [6]. Contoh perintah untuk membuat basis data baru dengan nama "penjualan" dapat dilihat pada Gambar 2.2.

```
CREATE DATABASE penjualan;
```

Gambar 2.2 Contoh *Query* Pembuatan Basis Data

   b. Menghapus basis data

      Untuk menghapus suatu database, sintaks umumnya dapat dilihat pada Gambar 2.3.

```
DROP DATABASE [IF EXISTS] nama_database;
```

Gambar 2.3 Sintak Umum Pembuatan Basis Data

Perintah pada Gambar 2.3 akan menghapus basis data dengan nama "nama_database". Jika basis data ada maka basis data dan juga seluruh

tabel di dalamnya akan dihapus. Jika nama basis data yang akan dihapus tidak ditemukan, maka akan ditampilkan pesan *error*. Untuk menghidari *error* maka dapat diikuti perintah **IF EXISTS** untuk memastikan bahwa suatu basis data benar-benar ada [6]. Contoh perintah untuk menghapus basis data dengan nama "penjualan" dapat dilihat pada Gambar 2.4.

```
DROP DATABASE penjualan;
```

Gambar 2.4 Contoh *Query* Penghapusan Basis Data

2. Membuat, Mengubah dan menghapus tabel
   a. Membuat Tabel Baru

   Bentuk umum SQL untuk membuat suatu tabel secara sederhana dapat dilihat pada Gambar 2.5.

```
CREATE TABLE nama_tabel (
field1 tipe(panjang),
...
fieldn tipe(panjang),
PRIMARY KEY (field_key)
);
```

Gambar 2.5 Sintak Umum Perintah Pembuatan Tabel

Perintah pada Gambar 2.5 merupakan bentuk umum pembuatan tabel yang sudah disederhanakan. Penamaan tabel dan *field* memiliki aturan yang sama dengan penamaan basis data. Contoh stuktur tabel yang akan dibuat dapat dilihat pada Tabel 2.1.

Tabel 2.1 Contoh Struktur Tabel Pelanggan

No	Nama Field	Tipe	Panjang
1	id_pelanggan	Varchar	5
2	nm_pelanggan	Varchar	30

3	alamat	Text	-
4	telepon	Varchar	20
5	email	Varchar	50

Untuk membuat tabel pada basis data sesuai struktur yang sudah dibuat seperti pada Tabel 2.1, *query* atau perintah SQL-nya dapat dilihat pada Gambar 2.6.

```
CREATE TABLE pelanggan (
        id_pelanggan varchar(5) NOT NULL,
        nm_pelanggan varchar(30) NOT NULL,
        alamat text,
        telepon varchar (20),
        email varchar (50),
        PRIMARY KEY(id_pelanggan)
);
```

Gambar 2.6 Contoh Query Membuat Tabel

Perintah pada Gambar 2.6 terdapat beberapa hal yang perlu diperhatikan, antara lain:

1. CREATE TABLE merupakan perintah dasar dari pembuatan table.
2. pelanggan merupakan nama tabel yang akan dibuat.
3. id_pelanggan, nm_pelanggan, alamat, telepon dan email merupakan nama field.
4. varchar dan text merupakan tipe data dari field
5. NOT NULL merupakan option untuk menyatakan bahwa suatu field tidak boleh kosong.
6. PRIMARY KEY merupakan perintah untuk menentukan field mana yang akan dijadikan primary key pada tabel.
7. 5, 10, 30 dan 50 di belakang tipe data merupakan panjang maksimal dari suatu field.

8. Untuk tipe data date dan text (juga date dan blob) panjang karakter maksimalnya tidak perlu ditentukan.
9. Perintah dengan titik-koma (;).

b. Mengubah struktur tabel dengan ALTER

Untuk mengubah struktur suatu tabel, bentuk umum perintah SQL-nya dapat dilihat pada Gambar 2.7.

```
ALTER TABLE nama_tabel alter_options;
```

Gambar 2.7 Sintak Umum Mengubah Struktur Tabel

Perintah pada Gambar 2.7 memiliki beberapa hal yang perlu diperhatikan, yaitu:

1. **ALTER TABLE** merupakan perintah dasar untuk mengubah tabel.
2. **nama_tabel** merupakan nama tabel yang akan diubah strukturnya.
3. **alter_options** merupakan pilihan perubahan tabel. *Option* yang bisa digunakan, beberapa di antaranya sebagai berikut.
   a. **ADD** definisi_field_baru
      Option ini digunakan untuk menambahkan *field* baru dengan "definisi_field_baru" (nama *field*).
   b. **ADD INDEX** nama_index
      Option ini digunakan untuk menambahkan *index* dengan nama "nama_index" pada tabel.
   c. **ADD PRIMARY KEY** (field_kunci)
      Option untuk menambahkan *primary key* pada tabel.
   d. **CHANGE** field_yang_diubah definisi_field_baru
      Option untuk mengubah "field_yang_diubah" menjadi "definisi_field_baru".
   e. **MODIFY** definisi_field
      Option untuk mengubah suatu *field* menjadi "definisi_field".
   f. **DROP** nama_field

Option untuk menghapus *field* "nama_field".

g. **RENAME TO** nama_tabel_baru

Option untuk mengganti nama tabel.

Beberapa contoh variasi perintah ALTER untuk mengubah struktur suatu tabel antara lain:

1. Menambah *field*

   Contoh *query* untuk menambah *field* "tgllahir" ke tabel pelanggan dapat dilihat pada Gambar 2.8.

```
ALTER TABLE pelanggan ADD tgllahir date NOT NULL;
```

Gambar 2.8 Contoh *Query* Menambah *Field*

2. Menambahkan *primary key* pada suatu tabel

   Contoh *query* untuk menambah primary key pada tabel dapat dilihat pada Gambar 2.9.

```
ALTER TABLE pelanggan ADD PRIMARY
KEY(id_pelanggan);
```

Gambar 2.9 Contoh *Query* Menambah *Primary Key*

3. Mengubah tipe *field*

   Contoh *query* untuk mengubah tipe *field* tgllahir menjadi varchar dalam tabel pelanggan dapat dilihat pada Gambar 2.10.

```
ALTER TABLE pelanggan MODIFY tgllahir varchar(8) NOT
NULL;
```

Gambar 2.10 Contoh *Query* Mengubah Tipe *Field*

4. Menghapus *field* tgllahir dari tabel pelanggan
   Contoh *query* untuk menghapus *field* tgllahir dari tabel pelanggan dapat dilihat pada Gambar 2.11.

```
ALTER TABLE pelanggan DROP tgllahir;
```

Gambar 2.11 Contoh *Query* Menghapus *Field*

c. Mengubah nama tabel
   Untuk mengubah nama suatu tabel, dapat menggunakan perintah SQL yang dapat dilihat pada Gambar 2.12.

```
RENAME TABLE pelanggan TO plg;
```

Gambar 2.12 Contoh Query Mengubah Nama Tabel

d. Menghapus Tabel
   Untuk menghapus sebuah tabel, bentuk umum dari perintah SQL dapat dilihat pada Gambar 2.13.

```
DROP TABLE nama_tabel;
```

Gambar 2.13 Sintak Umum Menghapus Tabel

Sebagai contoh untuk menghapus tabel dengan nama “pelanggan” maka perintah SQL-nya dapat dilihat pada Gambar 2.14.

```
DROP TABLE pelanggan;
```

Gambar 2.14 Contoh Query Menghapus Tabel

2.2. *Case Folding*

Case folding adalah proses menyeragamkan semua teks ke dalam *case* yang sama menjadi huruf kecil (*lowercase*) atau huruf capital (*uppercase*) [7]. Pada penelitian ini huruf dalam teks diseragamkan menjadi huruf kecil. Contoh dari tahap *case folding* dapat dilihat pada Tabel 2.2. Tabel tersebut menjelaskan bahwa semua huruf kapital yang ada pada teks input akan diubah menjadi huruf kecil.

Tabel 2.2 Contoh Tahap *Case Folding*

Sebelum	Sesudah
Buatkan Database Dengan Nama dbmahasiswa	**b**uatkan **d**atabase **d**engan **n**ama dbmahasiswa

Jadi, penelitian ini menggunakan tahap *case folding* untuk menyeragamkan semua teks menjadi huruf kecil (*lowercase*).

2.3. *Cleansing*

Cleansing merupakan proses dimana teks selain karakter ‘a’ sampai ‘z’, ‘0’ sampai ‘9’, ‘_‘, ‘,’ dan spasi akan dihilangkan. Berikut adalah contoh tahap *cleansing.*

Tabel 2.3 Contoh Tahap *Cleansing*

Sebelum	Sesudah
buatkan database dengan nama dbmahasiswa**!**	buatkan database dengan nama dbmahasiswa

Jadi, penelitian ini menggunakan tahap *cleansing* untuk menghilangkan karakter lain selain karakter ‘a’ sampai ‘z’, ‘0’ sampai ‘9’, ‘_‘, ‘,’ dan spasi.

2.4. *Tokenizing* Kata

Tokenizing kata adalah proses memindai *string* input dan dipotong berdasarkan tiap kata. Pemecahan kalimat menjadi kata-kata tunggal dilakukan dengan memindai kalimat dan memisahkan kata demi kata dengan spasi. Hasil keluaran dari proses *tokenizing* kata akan digunakan sebagai masukan dalam tahap transformasi teks. Tabel 2.4 dibawah ini merupakan contoh proses *tokenizing* kata.

Tabel 2.4 Contoh Tahap *Tokenizing* Kata

Sebelum	Sesudah
buatkan database dengan nama dbmahasiswa	buatkan | database | dengan | nama | dbmahasiswa

Jadi, penelitian ini menggunakan tahap *tokenizing* kata dengan memotong teks keseluruhan menjadi per-kata dengan *delimiter* spasi.

2.5. *Stemming*

Stemming merupakan suatu proses yang mentransformasi kata-kata yang berimbuhan ke kata-kata dasarnya (*root word*) [7]. Sebagai contoh, kata "menyenangkan", "menyenangi", dapat dikatakan serupa atau satu kelompok dan dapat diwakili oleh satu kata umum "senang". Proses *stemming* pada teks berbahasa Indonesia berbeda dengan stemming pada teks berbahasa Inggris. Pada teks berbahasa Inggris, proses yang diperlukan hanya proses menghilangkan sufiks. Sedangkan pada teks berbahasa Indonesia, selain sufiks, prefiks, dan konfiks juga dihilangkan. Pada umumnya kata dasar pada bahasa Indonesia terdiri dari kombinasi:

$$\text{Prefiks 1} + \text{Prefiks 2} + \text{Kata dasar} + \text{Sufiks 3} + \text{Sufiks 2} + \text{Sufiks 1} \quad (2.1)$$

Pada penelitian ini, algoritma yang digunakan dalam proses *stemming* yaitu Nazief dan Andriani yang telah dikembangkan oleh Sastrawi. Algoritma *Stemming* Bahasa Indonesia M. Adriani dan B. Nazief ini mempunyai aturan imbuhan sendiri dengan model sebagai berikut.

$$[[[\text{AW}+]\text{AW}+]\text{AW}+]\ \text{Kata-Dasar}\ [[+\text{AK}][+\text{KK}][+\text{P}]] \quad (2.2)$$

Dimana:

AW : Awalan

AK : Akhiran

KK : Kata Ganti kepunyaan

P : Partikel

Algoritma yang dibuat oleh Bobby Nazief dan Mirna Adriani ini memiliki tahapan sebagai berikut.

1. Mencari kata yang akan distem dalam kamus. Jika ditemukan maka diasumsikan bahwa kata tesebut adalah *root word*. Maka algoritma berhenti.
2. *Inflection Suffixes* (“-lah”, “-kah”, “-ku”, “-mu”, atau “-nya”) dibuang. Jika berupa *particles* (“-lah”, “-kah”, “-tah” atau “-pun”) maka langkah ini diulangi lagi untuk menghapus *Possesive Pronouns* (“-ku”, “-mu”, atau “-nya”), jika ada.
3. Hapus *Derivation Suffixes* (“-i”, “-an” atau “-kan”). Jika kata ditemukan dikamus, maka algoritma berhenti. Jika tidak maka ke langkah 3a.
    a. Jika “-an” telah dihapus dan huruf terakhir dari kata tersebut adalah “-k”, maka “-k” juga ikut dihapus. Jika kata tersebut ditemukan dalam kamus maka algoritma berhenti. Jika tidak ditemukan maka lakukan langkah 3b.
    b. Akhiran yang dihapus (“-i”, “-an” atau “-kan”) dikembalikan, lanjut ke langkah 4.
4. Hapus *Derivation Prefix*. Jika pada langkah 3 ada sufiks yang dihapus maka pergi ke langkah 4a, jika tidak pergi ke langkah 4b.
    a. Periksa tabel kombinasi awalan-akhiran yang tidak diijinkan. Jika ditemukan maka algoritma berhenti, jika tidak pergi ke langkah 4b.
    b. For i = 1 to 3, tentukan tipe awalan kemudian hapus awalan. Jika *root word* belum juga ditemukan lakukan langkah 5, jika sudah maka algoritma berhenti. Catatan: jika awalan kedua sama dengan awalan pertama algoritma berhenti.

5. Melakukan *Recoding*.
6. Jika semua langkah telah selesai tetapi tidak juga berhasil maka kata awal diasumsikan sebagai *root word*. Proses selesai.

Tipe awalan ditentukan melalui langkah-langkah sebagai berikut.

1. Jika awalannya adalah: “di-”, “ke-”, atau “se-” maka tipe awalannya secara berturut-turut adalah “di-”, “ke-”, atau “se-”.

2. Jika awalannya adalah “te-”, “me-”, “be-”, atau “pe-” maka dibutuhkan sebuah proses tambahan untuk menentukan tipe awalannya.
3. Jika dua karakter pertama bukan “di-”, “ke-”, “se-”, “te-”, “be-”, “me-”, atau “pe-“ maka berhenti.
4. Jika tipe awalan adalah “none” maka berhenti. Jika tipe awalan adalah bukan “none” maka awalan awalan jika ditemukan.

Sebagai contoh dapat dilihat pada Tabel 2.5 kata “buatkan” akan diubah menjadi kata “buat” karena pada kata “buatkan” terdapat imbuhan –kan [8].

Tabel 2.5 Contoh Tahap *Stemming*

Sebelum	Sesudah
buatkan database dengan nama dbmahasiswa	buat database dengan nama dbmahasiswa

Pada penelitian ini digunakan *library stemming* bahasa Indonesia untuk PHP dari Sastrawi. Karena *library* ini menggunakan tiga algoritma yaitu algoritma Nazief dan Adriani, algoritma CS Stemmer dan algoritma ECS *Stemmer*. Dimana algoritma CS *Stemmer* dan algoritma ECS *Stemmer* memodifikasi beberapa aturan yang terdapat di algoritma Nazief dan Adriani. Pada penelitian ini juga beberapa kata tidak dilakukan proses *stemming* karena proses *stemming* hanya untuk mendapatkan kata kunci.

2.6. *Removing Stopword*

Stopword merupakan kata yang diabaikan dalam pemrosesan, kata-kata ini biasanya disimpan ke dalam *stop lists*. Karakteristik utama dalam pemilihan *stopword* biasanya adalah kata yang mempunyai frekuensi kemunculan yang tinggi misalnya kata penghubung seperti “dan”, “atau”, “tapi”, “akan” dan lainnya. Tidak ada aturan pasti dalam menentukan *stopword* yang akan digunakan, penentuan s*topword* bisa disesuaikan dengan kasus yang sedang diselesaikan [9]. Beberapa kata yang termasuk *stopword* dapat dilihat pada Tabel 2.6.

Tabel 2.6 Beberapa Daftar Kata Yang Termasuk *Stopword*

No	Kata *Stopword*	No	Kata *Stopword*
1	coba	11	untuk
2	tolong	12	dengan
3	ada	13	apa
4	agar	14	maka
5	akan	15	lagi
6	berapa	16	lain
7	dan	17	jadi
8	yang	18	jika
9	pada	19	maka
10	yaitu	20	itu

Pada Tabel 2.6 merupakan beberapa daftar kata yang termasuk *stopword*, penelitian ini hanya menggunakan 320 kata dasar dari 758 kata *stopword* yang digunakan oleh Fadillah Z Tala [9] yang dapat dilihat pada lampiran C-1 dan disesuaikan dengan menghapus 3 kata ("buat", "tambah", dan "semua") karena kata tersebut digunakan pada kamus kata kunci untuk menentukan perintah DDL dan menambah 2 kata ("tolong" dan "coba") karena kata tersebut tidak digunakan pada kamus kata kunci.

2.7. *Black Box Testing*

Black box testing adalah pengujian yang dilakukan hanya mengamati hasil eksekusi melalui data uji dan memeriksa fungsional dari perangkat lunak. Jadi dianalogikan seperti kita melihat suatu koatak hitam, kit hanya bisa melihat penampilan luarnya saja, tanpa tau ada apa dibalik bungkus hitam nya. Sama seperti pengujian black box, mengevaluasi hanya dari tampilan *interface*-nya, fungsionalitasnya, tanpa mengetahui apa sesungguhnya yang terjadi dalam proses detilnya (hanya mengetahui *input* dan *output*).

Black box testing adalah metode pengujian perangkat lunak yang menguji fungsionalitas aplikasi yang bertentangan dengan struktur internal atau kerja (lihat pengujian *white-box*). Pengetahuan khusus dari kode aplikasi / struktur internal dan pengetahuan pemrograman pada umumnya tidak diperlukan. Uji kasus dibangun di sekitar spesifikasi dan persyaratan, yakni, aplikasi apa yang seharusnya dilakukan. Menggunakan deskripsi eksternal perangkat lunak, termasuk spesifikasi, persyaratan, dan desain untuk menurunkan uji kasus. Tes ini dapat menjadi fungsional atau non-fungsional, meskipun biasanya fungsional. Perancang uji

memilih input yang valid dan tidak valid dan menentukan output yang benar. Tidak ada pengetahuan tentang struktur internal benda uji itu.

Metode uji dapat diterapkan pada semua tingkat pengujian perangkat lunak: unit, integrasi, fungsional, sistem dan penerimaan. Ini biasanya terdiri dari kebanyakan jika tidak semua pengujian pada tingkat yang lebih tinggi, tetapi juga bisa mendominasi unit testing juga.

Pengujian pada *Black Box* berusaha menemukan kesalahan seperti:

1. Fungsi-fungsi yang tidak benar atau hilang
2. Kesalahan *interface*
3. Kesalahan dalam struktur data atau akses *database* eksternal
4. Kesalahan *kinerja*
5. *Inisialisasi* dan kesalahan *terminasi*

Pada penelitian ini menggunakan pengujian *black box* untuk menguji fungsionalitas sistem.

2.8. Akurasi

Untuk menghitung akurasi dari hasil query dilakukan perhitungan dengan rumus berikut [10]

$$Akurasi = \frac{Jumlah\ hasil\ query\ sistem\ benar}{Jumlah\ sampel\ data\ uji}\ X\ 100\% \quad (2.3)$$

Perhitungan akurasi digunakan untuk mengetahui tingkat keberhasilan dari proses translasi yang dilakukan oleh sistem dengan membandingakan *query* harapan dan *query* hasil translasi.

2.9. UML

UML (*Unified Modeling Language*) adalah bahasa grafis untuk mendokumentasi, menspesifikasikan, dan membangun sistem perangkat lunak. UML berorientasi objek, menerapkan banyak level abstraksi, tidak bergantung proses pembangunan, tidak bergantung bahasa dan teknologi, pemaduan beberapa notasi di beragam metodologi, usaha bersama dari banyak pihak, didukung oleh kakas-kakas yang diintegrasikan lewat XML. Standar UML dikelola oleh OMG (*Object Management Group*).

UML menyediakan sejumlah diagram untuk mengekspresikan pemodelan berorientasi objek yang dilakukan. Kegunaan diagram pada pemodelan adalah untuk formalitasi ekspresi model objek secara koheren, presisi dan mudah dirumuskan. UML yang dibahas pada bab ini *adalah Use Case Diagram, Class Diagram dan Sequence Diagram* [11]. Pada penelitian ini menggunakan UML untuk mendokumentasi, menspesifikasikan, dan membangun sistem penerjemah DDL SQL dalam Bahasa Indonesia.

2.9.1. *Use Case Diagram*

Use Case Diagram merupakan salah satu diagram untuk memodelkan aspek perilaku sistem. Masing-masing diagram use case menunjukkan sekumpulan use case, actor dan hubungannya. Diagram ini sangat penting untuk memvisualisasikan, menspesifikasikanm dan mendokumentasikan kebutuhan perilaku sistem. Use case adalah interaksi anatara actor eksternal dan sistem, hasil yang dapat diamati oleh actor, berorientasi pada tujuan, dideskripsikan di diagram use case dan teks [11]. Diagram ini melibatkan:

1. Sistem yaitu sesuatu yang hendak bangun
2. *Actor,* entitas-entitas luar yang berkomunikasi dengan sistem
3. *Use case* adalah fungsionalitas yang dipersepsi oleh *actor*
4. Relasi adalah relasi antara *actor use case*

Elemen pada *use case diagram* adalah :

1. *Actor*

Actor adalah pemakai sistem, dapat berupa manusia atau sistem terotomatisasi lain. *Actor* adalah sesuatu atatu seseorang yang berinteraksi dengan sistem, yaitu siapa atau apa yang menggunakan sistem. Yang dimaksud berinteraksi adalah *actor* mengirim atau menerima pesan ke atau dari sistem, atau mempertukarkan informasi dengan sistem.

2. *Use Case*

Use case adalah cara spesifik penggunaan sistem oleh *actor*. *Use case* menspesifikasikan perilaku sistem atau bagian sistem dan merupakan deskripsi sekumpulan sekuen aksi termasuk varian-varian yang dilakukan oleh sistem untuk memproduksi hasil atau nilai ke *actor. Use case* berguna untuk menangkap

kebutuhan sistem, berkomunikasi dengan pemakai akhir dan pakar domain masalah, serta pengujian sistem.

Relasi dalam *Use* Case diantaranya adalah :

1. *Association*, menghubungkan link antar *element*.
2. *Generalization*, biasa diebut *inheritence*, sebuah *element* dapat merupakan spesialisasi dari *element* lainnya.
3. *Depedency*, sebuah *element* bergantung dalam beberapa cara ke *element* lainnya.
4. *Aggregation*, bentuk *association* dimana sebuah *element* berisi *element* lainnya.

Pada penelitian ini menggunakan *use case diagram* untuk memodelkan perilaku sistem penerjemah DDL SQL dalam Bahasa Indonesia.

2.9.2. *Activity Diagram*

Diagram aktivitas atau *Activity Diagram* menggambarkan aliran fungsionalitas sistem. Pada tahap pemodelan bisnis, diagram aktivitas dapat digunakan untuk menunjukkan aliran kerja bisnis atau menggambarkan aliran kejadian dalam *Use Case* [11]. Komponen pada diagram ini antara lain :

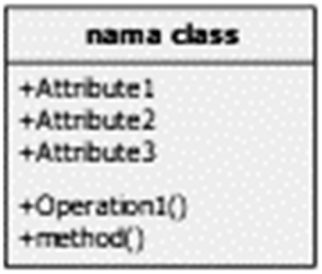
1. *Activity* atau *State* : menunjukkan aktivitas yang dilakukan
2. *Initial activity* atau *initial state* : Menunjukkan awal aktivitas dimulai
3. *Final activity* atau *final state* : Menunjukkan akhir dari aktivitas
4. *Decission* : Digunakan untuk menggambarkan *test* kondisi untuk memastikan bahwa control *flow* atau object *flow* mengalir ke satu jalur.
5. *Merge* : Berfungsi menggabungkan *flow* yang dipecah oleh *decission*
6. *Synchronization* : Dibagi menjadi dua yaitu *join* dan *fork*. *Fork* digunakan untuk memecah *behavior* menjadi *activity* atau *action* yang paralel. Sedangkan *join* untuk menggabungkan kembali aktivitas atau *action* yang paralel
7. *Swimlanes* : Memecah aktivitas diagram menjadi baris dan kolom untuk membagi tanggung jawab obyek-obyek yang melakukan aktivitas
8. *Transition* : Menunjukkan aktivitas selanjutnya setelah aktivitas sebelumnya.

Pada penelitian ini menggunakan *acivity diagram* untuk menggambarkan aliran fungsionalitas sistem penerjemah DDL SQL dalam Bahasa Indonesia.

2.9.3. *Class Diagram*

Class diagram adalah diagram yang digunakan untuk menampilkan beberapa kelas serta paket-paket yang ada dalam sistem/perangkat lunak yang sedang kita kembangkan dimana diagram ini memberi gambaran tentang sistem/perangkat lunak dan relasi-relasi yang ada di dalamnya [11].

Atribut merupakan properti dari sebuah kelas yang melambangkan nilai-nilai yang mungkin ada pada kelas tersebut. *Operation* atau *method* merupakan *behavior* (tingkah laku) atau fungsi yang dapat dilakukan oleh kelas tersebut.



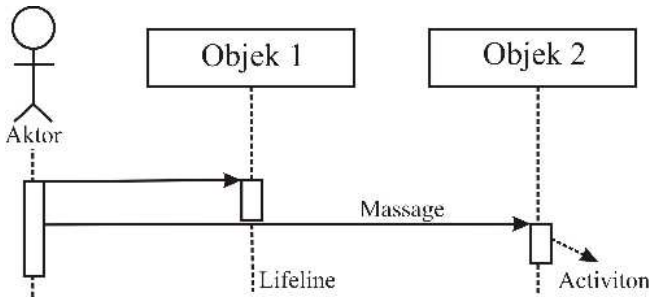
Gambar 2.15 Contoh *Class Diagram*

Pada Gambar 2.15, Kolom paling atas merupakan representasi dari nama kelas, kolom tengah berisi atribut-artribut yang dimiliki kelas tersebut, dan kolom terakhir berisi *operand* atau *method*. Pada penelitian ini menggunakan *class diagram* untuk menggambarkan kelas-kelas yang ada pada sistem penerjemah DDL SQL dalam Bahasa Indonesia.

2.9.4. *Sequence Diagram*

Sequence diagram digunakan untuk menggambarkan perilaku pada sebuah skenario. Diagram jenis ini memberikan kejelasan sejumlah objek dan pesan-pesan yang diletakkan diantaranya didalam sebuah *use case*. Komponen utamanya adalah objek yang digambarkan dengan kotak segi empat, *message* yang digambarkan dengan garis penuh, dan waktu yang ditunjukkan dengan progress *verticall* [11].

Sequence diagram digunakan untuk memberikan gambaran detail dari setiap *use case* diagram yang dibuat sebelumnya. Setiap objek yang terlibat dalam sebuah *use case* digambarkan dengan garis putus-putus vertikal, kemudian *message* yang dikirim oleh objek digambarkan dengan garis horizontal secara kronologis dari atas ke bawah. Contoh *sequence diagram* dapat dilihat pada Gambar 2.16.



Gambar 2.16 *Sequence diagram*

Pada penelitian ini menggunakan *sequence diagram* untuk menggambarkan perilaku skenario sistem penerjemah DDL SQL dalam Bahasa Indonesia.

2.10. ERD

Entity Relationship Diagram (ERD) adalah diagram dari sistem yang menggambarkan hubungan antar entitas beserta relasinya yang saling terhubung. ERD menyediakan bentuk untuk menunjukkan strukturkeseluruhan dari data user. Dalam ERD data-data tersebut digambarkan dengan menggunakan simbol *entity*. Dalam perancangan sistem ini terdapat beberapa *entity* yang saling terkait untuk menyediakan data-data yang dibutuhkan oleh sistem [1]. Dalam pembentukkan ERD terdapat 3 komponen yang akan dibentuk yaitu :

a. Entitas

   Entitas adalah suatu objek yang keberadaannya dapat dibedakan terhadap objek lain yang dapat diwujudkan dalam basis data.

b. Hubungan (relasi/*relationship*)

   Hubungan atau relasi adalah hubungan antara antara entitas dan direpresentasikan sebagai garis lurus yang menghubungkan dua entitas.

c. Atribut

   Atribut memberikan informasi lebih rinci tentang entitas. Atribut memiliki struktur internal berupa tipe data, jenis-jenis atribut diantaranya :

   1. *Key*

      Atribut *key* adalah satu atau gabungan dari beberapa atribut yang dapat membedakan semua baris data dalam tabel secara unik.

2. *Simple*

   Atribut yang bernilai atomic, tidak dapat dipecah/dipilah lagi.

3. *Multivalue*

   Nilai dari suatu atribut yang mempunyai lebih dari satu nilai dari atribut yang bersangkutan.

4. *Composite*

   Composite adalah suatu atribut yang terdiri dari beberapa atribut yang lebih kecil yang mempunyai arti tertentu yang masih bisa dipecah lagi atau mempunyai *sub attribute*.

5. *Deriativ*

   Atribut yang tidak harus disimpan dalam *database* contoh. Total, atau atribut yang dihasilkan dari atribut lain atau dari suatu *relationship*.

Derajat relasi atau kardinalitas rasio dalam ERD :

a. *One* to *One* (1:1)

   Setiap anggota entitas A hanya boleh berhubungan dengan satu anggota entitas B, begitu pula sebaliknya.

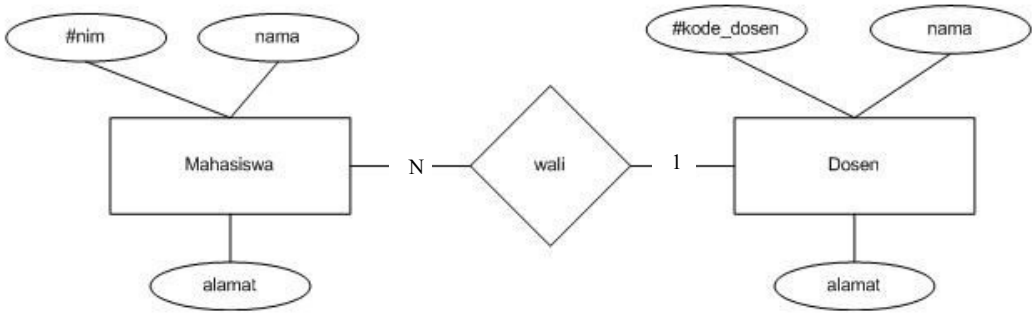
b. *One to Many* (1:N/Many)

   Setiap anggota entitas A dapat berhubungan dengan lebih dari satu anggota entitas B tetapi tidak sebaliknya.

c. *Many to Many* (N:N)

   Setiap anggota entitas A dapat berhubungan dengan banyak entitas himpunan entitas B dan demikian pula sebaliknya.

   Contoh ERD dapat dilihat pada Gambar 2.17.



Gambar 2.17 Contoh Skema ERD

Pada Gambar 2.17, mahasiswa dan dosen merupakan entitas yang memiliki hubungan/relasi *One to Many* antara dosen dengan mahasiwa. Karena beberapa mahasiswa dapat mempunyai wali dosen yang sama. Pada penelitian ini menggunakan *ERD* untuk menggambarkan basis data pendukung yang digunakan pada sistem penerjemah DDL SQL dalam Bahasa Indonesia.

2.11. Bahasa Pemrograman

Bahasa pemrograman adalah *software* bahasa komputer yang digunakan dengan cara merancang atau membuat program sesuai dengan struktur dan metode yang dimiliki oleh bahasa program itu sendiri. Komputer mengerjakan transformasi data berdasarkan kumpulan perintah program yang telah dibuat oleh program. Kumpulan perintah ini harus dimengerti oleh komputer, berstruktur tertentu (*syntax*), dan bermakna. Bahasa pemrograman merupakan notasi untuk memberikan secara tepat program komputer [12]. Pada penelitian ini bahasa pemograman yang digunakan adalah PHP karena bahasa pemrograman PHP tidak memerlukan *compile* dalam penggunaannya.

2.11.1. PHP

PHP adalah kependekan dari PHP *Hypertext Preprocessor*, bahasa *interpreter* yang mirip dengan bahasa C dan *Perl* yang memiliki kesederhanaan dalam perintah, PHP dapat digunakan bersama dengan HTML sehingga memudahkan dalam pembangunan aplikasi web dengan cepat. PHP dapat digunakan untuk meng-*update* basis data dan menciptakan basis data. *Interpreter* adalah sebuah program yang digunakan untuk membaca *file* yang berisi kode program yang akan dijalankan kemuadian *interpreter* tersebut akan meminta CPU untuk menlakukan perintah yang diterimanya.

Seperti halnya program *open source* lainnya, PHP dibuat di bawah lisensi GNU, *General Public License*, yang dapat di *download* gratis melalui situs http://www.php.net. Awalnya, PHP diciptakan oleh Andi Gutmans untuk menghitung jumlah pengunjung yang mengakses *homepage* yang dbuatnya. PHP banyak mendukung basis data, seperti MySQL, PostgresSQL, Interbase, ODBC,

mSQL, Oracle, dan Sybase [12]. Pada penelitian ini PHP digunakan sebagai bahasa pemograman utama dimana kode-kode PHP hanya akan dieksekusi di server kemudian hasilnya dikirim ke browser dalam format HTML.

2.11.2. *JavaScript*

Javascript adalah bahasa skrip yang ditempelkan pada kode HTML dan diproses di sisi klien. Dengan adanya bahasa ini, kemampuan dokumen HTML menjadi semakin luas. Sebagai contoh, dengan menggunakan *javascript* dimungkinkan untuk memvalidasi masukan - masukan pada formulir sebelum formulir dikirimkan ke *server*. *Javascript* bukanlah bahasa *Java* dan merupakan dua bahasa yang berbeda. *Javascript* diinterpretasikan oleh klien (kodenya bisa dilihat pada sisi klien), sedangkan kode *Java* dikompilasi oleh pemrogram dan hasil kompilasi yang dijalankan oleh klien [13]. Pada penelitian ini *javascript* diimplementasikan pada pembuatan antarmuka tambah data kata kunci serta proses pengiriman dan penerimaan data.

2.11.3. HTML

HTML adalah sebuah bahasa markup berupa kode-kode tag yang menginstruksikan *browser* untuk menghasilkan tampilan sesuai dengan yang diinginkan. HTML adalah sebuah standar yang digunakan secara luas untuk menampilkan halaman web dan HTML kini merupakan standar Internet yang saat ini dikendalikan oleh *World Wide Web Consortium* (W3C). Dalam penelitian ini HTML digunakan sebagai tambahan dalam pengimplementasian program.

2.12. Format Pertukaran Data

Pertukaran data elektronik (bahasa inggris: EDI atau *Electronic Data Interchange*, juga *Electronic Document Interchange*) adalah proses transfer data yang terstruktur, dalam format standar yang disetujui dari satu sistem komputer ke sistem komputer lainnya dalam bentuk elektronik [13]. Format pertukaran data yang digunakan dalam penelitian ini adalah JSON sebagai media pertukaran data untuk mangambil hasil dari proses translasi.

2.12.1. JSON (*Java Script Object Notation*)

JSON (*JavaScript Object Notation*) adalah format pertukaran data yang ringan, mudah dibaca dan ditulis oleh manusia, serta mudah diterjemahkan dan dibuat (*generate*) oleh komputer. Format ini dibuat berdasarkan bagian dari Bahasa Pemrograman *JavaScript*, Standar ECMA-262 Edisi ke-3 - Desember 1999. JSON merupakan format teks yang tidak bergantung pada bahasa pemprograman apapun karena menggunakan gaya bahasa yang umum digunakan oleh *programmer* keluarga C termasuk C, C++, C#, *Java*, *JavaScript*, *Perl*, *Python* dll. Oleh karena sifat-sifat tersebut, menjadikan JSON ideal sebagai bahasa pertukaran-data [13].

JSON terbuat dari dua struktur:

1 Kumpulan pasangan nama/nilai. Pada beberapa bahasa, hal ini dinyatakan sebagai objek (*object*), rekaman (*record*), struktur (*struct*), kamus (*dictionary*), tabel hash (*hash table*), daftar berkunci (*keyed list*), atau *associative array*.
2 Daftar nilai terurutkan (*an ordered list of values*). Pada kebanyakan bahasa, hal ini dinyatakan sebagai larik (*array*), vektor (*vector*), daftar (*list*), atau urutan (*sequence*).

Struktur-struktur data ini disebut sebagai data *universal*. Pada dasarnya, semua bahasa pemprograman *modern* mendukung struktur data ini dalam bentuk yang sama maupun berlainan. Hal ini pantas disebut demikian karena format data mudah dipertukarkan dengan bahasa-bahasa pemrograman yang juga berdasarkan pada data ini. Pada penelitian ini hasil dari proses translasi ditampilkan dalam bentuk JSON.