BAB 2

LANDASAN TEORI

2.1 Pengertian Sistem Basis Data

Berikut ini adalah definisi dari sistem basis data secara rinci, di mana definisi ini terbagi menjadi beberapa definisi, yaitu:

2.1.1 Definisi Sistem

Sistem adalah setiap kesatuan secara konseptual atau fisik yang terdiri dari bagian-bagian dalam keadaan saling tergantung satu sama lainnya.

Syarat-syarat sistem adalah :

a. Sistem harus dibentuk untuk menyelesaikan tujuan.
b. Elemen sistem harus mempunyai rencana yang ditetapkan.
c. Adanya hubungan diantara elemen sistem.
d. Unsur dasar dari proses (arus informasi, energi, dan material) lebih penting daripada elemen sistem.
e. Tujuan organisasi lebih penting daripada tujuan elemen.

2.1.2 Definisi Data

Data menurut Alter (1999, p8), data merupakan fakta, gambar atau suara yang berhubungan atau tidak berhubungan dan bermanfaat bagi tugas tertentu. Dapat dilihat bahwa data itu adalah suatu bentuk informasi yang belum diolah atau masih mentah, data itu

dapat diperoleh dari hasil kegiatan yang dilakuan sehari-harinya atau hasil dari transaksi-transaksi yang ada.

2.1.3 Definisi Sistem Basis Data

Basis data menurut Connolly dan Begg (2001, p14), adalah suatu bagian koleksi dari data logika yang berhubungan, dan deskripsi dari data ini dirancang untuk mendapatkan informasi yang diperlukan oleh sebuah organisasi. Basis data merupakan sebuah tempat penyimpanan besar data-data yang di mana data-data tersebut bisa digunakan secara bersamaan oleh banyak departemen dan pengguna.

2.2 DBMS (*Database Management System)*

Menurut Connolly dan Begg (2001, p16), DBMS adalah sebuah sistem perangkat lunak yang membisakan pengguna untuk mendefinisikan, membuat, mengelola, dan mengontrol akses ke basis data.

Menurut Silberschatz et al. (2002, p1), DBMS adalah sekumpulan dari data yang saling terkait dan sekumpulan program untuk mengakses data-data tersebut. Tujuan utama dari DBMS adalah menyediakan cara untuk menyimpan dan menerima balik informasi basis data yang nyaman dan efisien.

DBMS menyediakan berbagai fasilitas seperti :

a. Mengizinkan pengguna untuk mendefinisikan basis data;
b. Mengizinkan pengguna untuk memasukkan, mengubah, menghapus, dan mengambil data dari basis data;
c. Menyediakan akses kontrol ke basis data, contohnya:

1) *Security system* yang mencegah pengguna luar untuk mengakses basis data;
2) *Integrity system* yang menjaga konsistensi dari data yang tersimpan;
3) *Concurrency control system* yang mengizinkan pembagian akses basis data;
4) *Recovery control system* yang dapat men-*restore database* jika terjadi gangguan pada *hardware* atau pada *software;*
5) *User-accessible catalog* yang berisi deskripsi data di basis data.

Komponen-komponen dari lingkungan DBMS meliputi:

a. Perangkat keras

DBMS dan aplikasi-aplikasi memerlukan perangkat keras untuk dijalankan. Perangkat keras bisa berupa komputer pribadi, *single* mainframe, ataupun jaringan komputer.

b. Perangkat lunak

Perangkat lunak DBMS meliputi perangkat lunak DBMS itu sendiri dan aplikasi program, bersama sistem operasi, termasuk perangkat lunak jaringan jika DBMS tersebut digunakan dalam sebuah jaringan.

c. Data

Data bertindak sebagai penghubung komponen mesin dan manusia. Basis data berisi data operasional dan *metadata*, 'data yang menjelaskan data'.

d. Prosedur

Prosedur menuju pada intruksi dan peraturan yang membangun rancangan dan pengunaan basis data.

e. Manusia

Orang yang terlibat dalam pemakaian DBMS.

Keuntungan DBMS :

a. Data tidak terikat pada program : data dapat digunakan secara bebas;
b. Respon yang cepat dari permintaan informasi: karena data terintegrasi kedalam sebuah basis data, permintaan yang kompleks dapat ditangani lebih cepat daripada data yang lokasinya di dalam file yang terpisah-pisah dan tidak terintegrasi;
c. *Multiple Access*: *software* basis data dapat diakses dengan cara yang bervariasi dengan menggunakan beberapa bahasa pemrograman;
d. Biaya pembelajaran yang lebih rendah: pengguna dapat dengan lebih mudah mempelajari sistem seperti ini sehingga biaya pelatihan dapat dikurangi, sehingga dapat meningkatkan produktifitas pengguna;
e. Penghematan penyimpanan: data-data yang sama cukup disimpan satu kali saja. terkadang *developer* dan perancang basis data menggunakan normalisasi untuk mengurangi data yang redundan;
f. Data terintegrasi: data yang terdapat komputer-komputer yang berbeda dihubungkan sehingga datanya berhubungan satu sama lain;
g. Data mudah dimengerti: data yang ada diatur sesuai kebutuhan pengguna dan terdapat dalam susunan yang mudah sehingga tidak ada kesulitan menggunakan data melalui DBMS;
h. Validitas data: dengan menggunakan DBMS data lebih *valid* dan mengurangi data yang *orthodox*. Dan terdapat banyak penguji yg diaplikasikan untuk menguji data itu *valid* atau tidak ketika dimasukkan;
i. Keamanan data: menggunakan DBMS data lebih aman. Hanya beberapa pengguna yang dapat mengoperasikan perangkat lunak;

j. Fleksibilitas: karena program dan data independen, sehingga program tidak perlu di modifikasi ketika data yang tidak berhubungan ditambahkan atau dibuang dari basis data, atau ketika tempat penyimpanan fisik berubah.

Kerugian DBMS:

a. Membutuhkan dana yang cukup besar untuk membuat DBMS yang baik;
b. Kompleksitas yang tinggi.

2.3 DDL (*Data Definition Language*)

Menurut Connolly dan Begg (2001, p40), DDL adalah sebuah bahasa yang mengizinkan DBA (*Database Administrator*) atau pengguna untuk mendeskripsikan dan memberi nama pada entiti, atribut dan relasi yang diperlukan oleh aplikasi, bersama beberapa integritas yang terasosiasi dan *security constraints*.

DDL digunakan untuk mendefinisikan sebuah skema atau memodifikasi struktur data yang sudah ada.

2.4 DML (*Data Manipulation Language*)

Menurut Connolly dan Begg (2001, p41), DML adalah sebuah bahasa yang menyediakan kumpulan operasi untuk mendukung operasi pemanipulasian *basic* data di data yang tersimpan di basis data.

Menurut Silbershatz et al. (2002, p12), DML adalah suatu bahasa yang mengizinkan pengguna untuk mengakses atau memanipulasi data seperti yang diatur oleh data model yang cocok. Secara dasar ada dua tipe, yaitu:

a. *Procedural DMLs*, membutuhkan pengguna untuk menentukan data apa yang diperlukan dan bagaimana cara mendapatkan data tersebut;
b. *Declarative DMLs* (*nonprocedural DMLs*), membutuhkan pengguna untuk menentukan data apa yang diperlukan tanpa menjelaskan bagaimana cara mendapatkan data tersebut. Saat ini, tipe ini yang paling banyak dipakai.

Operasi manipulasi data biasanya mencakup :

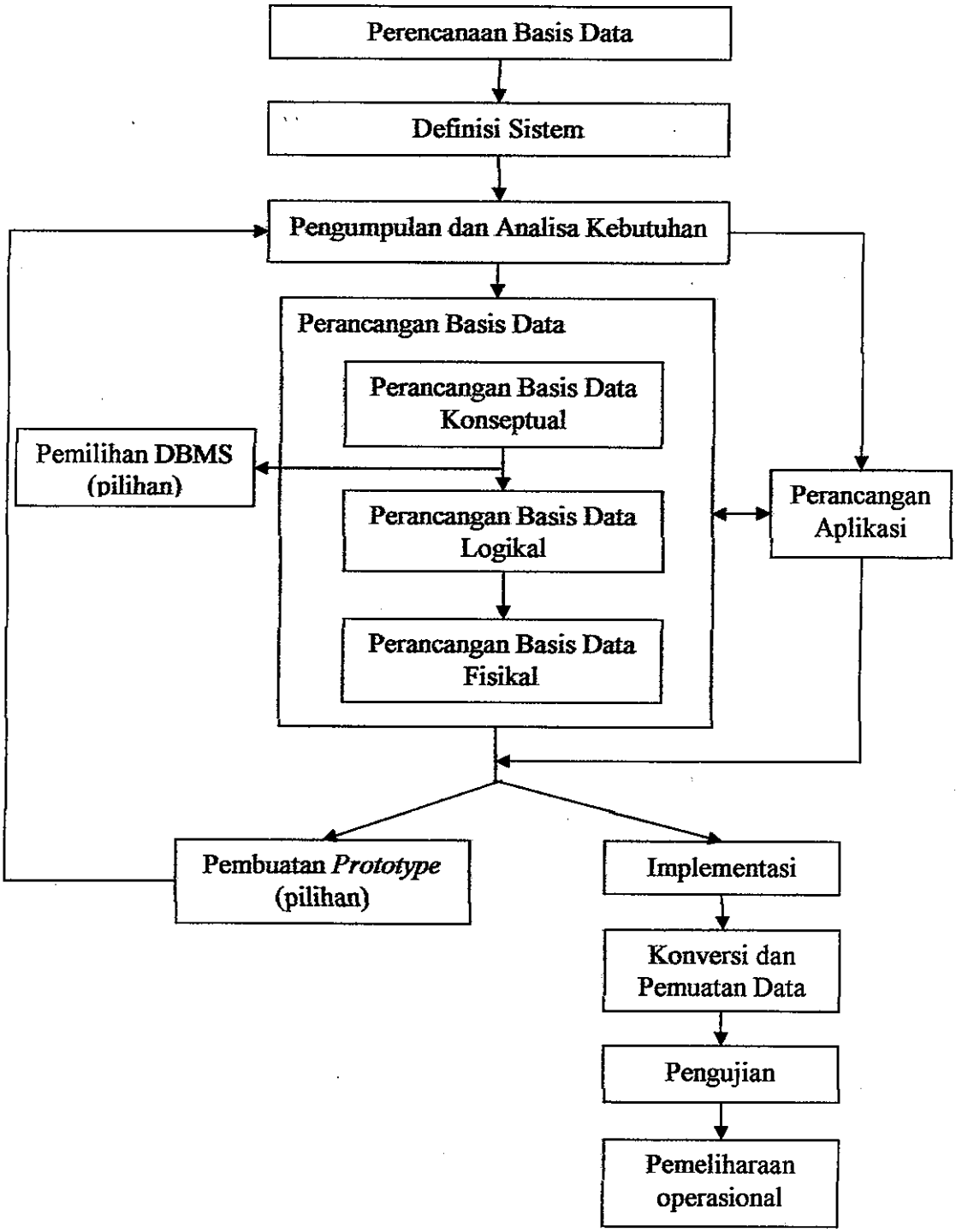
a. Pemasukan data baru ke basis data;
b. Pemodifikasian data yang tersimpan di basis data;
c. Pengambilan data yang berada di basis data;
d. Penghapusan data di basis data.

2.5 Siklus Hidup Basis Data (*Database Lifecycle*)

Sistem *database* adalah komponen penting dari sistem informasi organisasi luas yang lebih besar, siklus hidup basis data secara berpautan dekat dengan siklus hidup sistem informasi.

Penting untuk mengetahui bahwa tahapan dari siklus hidup *database* tidak sekuensial, tapi meliputi beberapa repetisi dari tahapan sebelumnya melalui *feed-back loops*. Sebagai contohnya, masalah yang terjadi pada rancangan basis data memerlukan tambahan koleksi dan analisis.

Untuk basis data yang kecil, dengan sedikit pengguna, siklus hidup tidak perlu terlalu kompleks. Namun, ketika merancang basis data yang menengah ke banyak dengan pengguna 10 sampai 1000, siklus hidupnya bisa menjadi sangat kompleks.



Gambar 2.1 Siklus Hidup Basis Data

Tabel 2.1 Tahapan dan Aktifitas Utama Dari Siklus Hidup Basis Data

Tahap	Aktifitas Utama
Perencanaan basis data	Merencanakan bagaimana tahapan dari siklus hidup dapat direalisasi paling efektif dan efisien.
Definisi sistem	Menentukan ruang lingkup dan batasan dari aplikasi basis data, penggunanya, dan wilayah aplikasi.
Pengumpulan dan analisa kebutuhan	Pengumpulan dan analisa kebutuhan pengguna dan wilayah aplikasi.
Perancangan basis data	Perancangan konseptual, logikal, dan fisikal dari basis data.
Pemilihan DBMS (pilihan)	Pemilihan DBMS yang cocok untuk aplikasi basis data.
Perancangan aplikasi	Merancang antar muka pengguna dan program aplikasi yang menggunakan dan mengolah basis data.
Pembuatan *prototype* (pilihan)	Membangun model aplikasi basis data yang mengizinkan perancang atau pengguna memvisualisasikan dan mengevaluasi bagaimana tampilan dan fungsi sistem akhir.
Implementasi	Membuat definisi basis data eksternal konseptual dan internal dan program aplikasi.
Konversi dan pemuatan data	Pemuatan data dari sistem lama ke sistem baru.
Pengujian	Aplikasi basis data diuji untuk dari kesalahan dan dicocokkan dengan keperluan yang ditentukan oleh pengguna.
Pemeliharaan operasional	Aplikasi basis data diimplementasikan secara menyeluruh. Sistem dipantau dan dipelihara secara terus-menerus. Jika perlu, kebutuhan baru dapat dimasukkan dalam aplikasi basis data melalui tahapan sebelumnya dari siklus hidup.

2.6 *Perencanaan Basis Data*

Perencanaan basis data mengatur aktifitas yang mengijinkan tahapan dari aplikasi basis data untuk direalisasikan seefisien dan seefektif mungkin.

Perencanaan basis data harus diintegrasi dengan keseluruhan strategi sistem informasi dari organisasi. Ada tiga hal yang terdapat dalam merumuskan strategi sistem informasi, yaitu:

a. Pengidentifikasian rencana dan tujuan perusahaan dengan penentuan berurut dari keperluan sistem informasi;
b. Pengevaluasian sistem informasi yang sedang berjalan untuk menentukan keberadaan kekuatan dan kelemahan;
c. Penilaian peluang IT yang bisa menghasilkan kompetisi yang menguntungkan.

2.7 Diagram Hubungan Entitas (*Entity Relationship Diagram*)

ERD adalah model yang dibuat berdasarkan anggapan bahwa dunia nyata terdiri dari koleksi obyek-obyek dasar yang dinamakan entitas (*entity*) serta hubungan (*relationship*) antara entitas-entitas itu. ERD berguna untuk membantu dalam memahami sesuatu hubungan atau relasi antara beberapa komponen-komponen atau entiti-entiti dalam suatu perancangan sebuah sistem.

ERD merupakan alat bantu yang digunakan untuk menggambarkan model data yang didapat dari spesifikasi kebutuhan. Model ER biasanya dinyatakan dalam ERD. Di dalam ERD terdapat tipe entiti, hubungan, dan atribut.

Skema ERD dibangun dari beberapa komponen utama, yaitu :

a. *Rectangles*, yang mewakili entity;

b. *Ellipses*, yang mewakili atribut;

c. *Diamonds*, yang mewakili hubungan (*relationship*);

d. *Lines*, garis yang menghubungkan atribut ke entiti dan entiti ke hubungan.

2.8 Tipe Entiti

Menurut Silberschatz et al. (2002, p27), entiti adalah sesuatu atau obyek pada dunia nyata yang bisa dibedakan dari semua obyek lain. Sebuah entiti memiliki sekumpulan sifat, dan nilai dari beberapa kumpulan sifat bisa mengenali sebuah entiti secara unik.

Menurut Connolly dan Begg (2005, p343) tipe entiti adalah sekumpulan objek yang memiliki *properties* yang sama yang mempunyai sebuah ketidaktergantungan eksistensi yang diidentifikasi oleh perusahaan. Kejadian entiti (*entity occurance*) adalah objek yang teridentifikasi di dalam tipe entiti. Entiti yang bisa didefinisikan antara lain : orang, tempat atau pun konsep. Beberapa contoh dari masing-masing tipe entiti adalah :

Orang : pegawai, mahasiswa. Tempat : cabang, kota.

Tipe entiti dibagi dalam dua jenis yaitu :

a. *Strong Entity Type* : tipe entiti yang keberadaannya tidak bergantung oleh adanya entiti lain.

b. *Weak Entity Type* : tipe entiti yang keberadaannya bergantung oleh adanya entiti lain.

2.9 Hubungan (*Relationship*)

Relationship menurut Connolly dan Begg (2005, p346) adalah suatu hubungan yang berarti antara satu atau lebih tipe entiti.

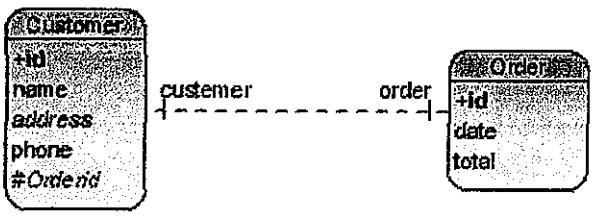
Menurut Silberschatz et al. (2002, p31), *relationship* adalah pembauran antara beberapa entitas. *Primary key* pada entiti digunakan untuk membedakan suatu entiti di antara bermacam-macam entiti.

Di dalam suatu relasi terdapat batasan-batasan yang disebut dengan *multiplicity*. *Multiplicity* adalah jangkauan nilai dari suatu kejadian tipe entiti yang mungkin berhubungan dengan kejadian tipe entiti lain. Jenis batasan *multiplicity* terdiri dari :

1) *One to one relationship*

Hubungan ini terjadi ketika satu *record* di sebuah tabel menunjuk hanya satu *record* di tabel yang lain

Contoh dalam gambar seperti di bawah ini. Dapat dilihat bahwa satu **customer** hanya dapat menunjuk ke satu **order** demikian juga sebaliknya.

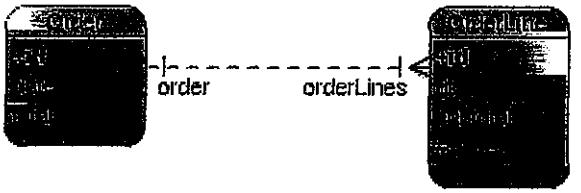


Gambar 2.2 Hubungan One_to_One

2) *One to Many Relationship*

Terjadi dimana ketika sebuah *record* di tabel A dapat menunjuk banyak *record* di tabel B tetapi tabel B hanya dapat ditunjuk oleh sebuah *record* dari tabel A. contohnya dapat dilihat seperti tabel di bawah ini.

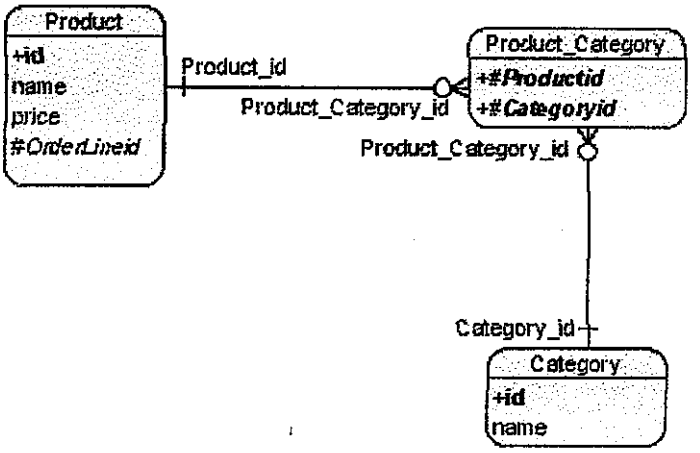
Tabel order dapat menunjuk beberapa orderLines tapi sebuah OrderLine hanya ditunjuk oleh sebuah Order.



Gambar 2.3 Hubungan One_to_Many

3) *Many to Many Relationship*

Dalam hubungan *many to many* dibutuhkan tabel pembantu untuk membantu hubungan antara dua tabel yang memiliki hubungan *many to many*. Dalam hubungan ini tiap tabel dapat menunjuk beberapa *record* di tabel yang lain. Contohnya dapat dilihat pada gambar di bawah ini.



Gambar 2.4 Hubungan Many_to_Many

2.10 *Perancangan Basis Data*

Proses dari pembuatan rancangan untuk basis data yang akan mendukung operasi dan tujuan perusahaan.

Dua pendekatan dalam perancangan basis data adalah *'bottom-up'* dan *'top-down'*. Pendekatan *bottom-up* cocok untuk dalam perancangan database dengan atribut yang sedikit. Sedangkan untuk perancangan basis data yang lebih kompleks, digunakan pendekatan *top-down*.

Perancangan basis data dibagi menjadi tiga proses, yaitu:

a. Perancangan Basis Data Konseptual

   Proses pembuatan model informasi yang digunakan oleh sebuah perusahaan, terpisah dari pertimbangan fisik. Meliputi pembuatan model data konseptual dari bagian perusahaan yang ingin ditiru. Data model dibuat sesuai dengan informasi yang berisi spesifikasi kebutuhan pengguna. Dalam pengembangan model data, dilakukan pengujian dan pengesahan terhadap kebutuhan pengguna.

b. Perancangan Basis Data Logikal

   Proses pembuatan sebuah model informasi yang digunakan dalam perusahaan sesuai dengan data model yang spesifik, tapi terpisah dari fakta DBMS dan pertimbangan fisik lainnya.

   Hasil dari perancangan basis data logikal adalah model data logikal dari bagian perusahaan yang ingin dicontoh. Dalam pengembangan model data, dilakukan pengujian dan pengesahan terhadap kebutuhan pengguna. Selanjutnya teknik normalisasi digunakan untuk menguji kebenaran dari model data logikal dan menghilangkan redundansi data.

c. *Perancangan Basis Data Fisikal*

Proses pembuatan deskripsi dari implementasi basis data di *secondary storage*; mendeskripsikan dasar relasi, pengelompokan *file*, dan indeks yang digunakan untuk mencapai akses data yang efisien, dan beberapa *associated integrity constrain* dan *security measures*.

Perancangan basis data fisikal merupakan tahap akhir dari proses perancangan basis data, di mana perancang menentukan bagaimana basis data diimplementasikan. Untuk model relasional, perancangan basis data fisikal meliputi:

1) Pembuatan sekumpulan tabel relasional dan aturannya (*constraint*-nya) dari informasi yang ada pada model data logikal;
2) Mengidentifikasi struktur penyimpanan khusus dan metoda akses data untuk memperoleh kinerja yang optimal dari sistem basis data;
3) Merancang perlindungan keamanan untuk sistem.

Secara ideal, perancangan basis data konseptual dan logikal untuk sistem yang lebih besar harus dipisahkan dari perancangan basis data fisikal karena 3 alasan utama,yaitu:

1) Bersangkutan dengan masalah subyek yang berbeda-*'the what', not 'the how'*;
2) Dikerjakan pada waktu yang berbeda-*'the what'* harus diketahui sebelum menanyakan bagaimana sebelum *'the how'* ditentukan;
3) Memerlukan kemampuan yang berbeda, yang biasanya ditemukan pada orang yang berbeda pula.

2.11 *Normalisasi*

Normalisasi adalah teknik pembuatan sekumpulan relasi dengan *properties* yang diinginkan, berdasarkan data persyaratan dari suatu perusahaan.

Normalisasi biasanya dilakukan secara bertahap pada pengujian relasi untuk menentukan apakah sudah sesuai atau melanggar syarat yang diberikan pada bentuk normal (*normal form*).

Ada 3 *normal form* pada proses normalisasi, yaitu:

a. *First Normal Form* (1NF)

Pada awal tahap normalisasi, terlebih dahulu dilakukan pemasukan data dari sumber ke dalam tabel yang berbentuk baris dan kolom. Tabel yang berisi data sumber ini disebut UNF (*Unnormalized Form*). UNF merupakan tabel yang berisi satu atau lebih kelompok yang berulang (redundan).

1NF adalah relasi di mana perpotongan setiap baris dan kolom berisi satu dan hanya satu nilai. Untuk mengubah UNF menjadi 1NF, dilakukan identifikasi dan membuang kelompok yang berulang pada tabel. Ada dua pendekatan dalam pembuangan kelompok dari UNF:

1) Pada pendekatan pertama, akan dibuang kelompok yang berulang dengan cara memasukkan data yang tepat pada kolom kosong dari baris yang berisi data yang berulang. Dengan kata lain, tempat kosong akan diisi dengan menduplikat data yang tidak berulang yang dibutuhkan. Hasilnya, sekarang mengacu pada sebuah relasi, berisi sebuah nilai tunggal pada perpotongan setiap baris dan kolom. Pada pendekatan ini redundansi dikenalkan ke dalam penghasilan relasi yang berurutan terbuang dalam proses normalisasi.

2) Pada pendekatan kedua, akan dibuang kelompok yang berulang dengan cara menempatkan data yang berulang bersama dengan tiruan dari *key attribute* yang asli dalam relasi yang terpisah. Sebuah *primary key* teridentifikasi untuk relasi baru. Kadang-kadang pada tabel UNF berisi lebih dari satu kelompok yang berulang, atau kelompong berulang di dalam kelompok yang berulang. Pada kasus seperti ini, pendekatan ini dilakukan berulang-ulang sampai tidak ada lagi kelompok yang berulang. Sekumpulan relasi akan terdapat pada 1NF jika tidak lagi kelompok yang berulang.

Kedua pendekatan ini sama-sama benar. Tetapi pada pendekatan yang kedua, akan dihasilkan relasi 1NF yang kurang redundan. Jika memilih pendekatan pertama, relasi 1NF memecah lebih jauh pada saat tahap berurutan normalisasi ke dalam relasi sama yang dihasilkan oleh pendekatan kedua.

Contoh 1NF:

Patient No, Drug No, Start Date, Full Name, Ward No, Ward Name, Bed No, Name, Description, Dosage, Method of Admin, Units per Day, Finish Date

b. *Second Normal Form* (2NF)

Second Normal Form adalah relasi yang di 1NF dan setiap atribut *non-primary-key* (bukan kunci utama) yang memiliki ketergantungan fungsional penuh pada *primary key* (kunci utama).

Normalisasi 1NF menjadi 2NF meliputi pembuangan bagian yang memiliki ketergantungan. Jika ada bagian yang memiliki ketergantungan, atribut yang memiliki ketergantungan fungsional itu akan dibuang dari relasi dengan menempatkannya ke relasi baru dengan tiruan (*copy*) dari determinannya.

Contoh 2NF:

Patient No, Drug No, Start Date, Ward No, Ward Name, Bed No, Units per Day, Finish Date Drug No, Name, Description, Dosage, Method of Admin Patient No, Full Name

c. *Third Normal Form* (3NF)

Third Normal Form adalah relasi yang ada pada 1NF dan 2NF di mana atribut bukan kunci utama memiliki ketergantungan transitif pada kunci utama. Ketergantungan transitif adalah suatu kondisi di mana A, B, dan C merupakan atribut dari relasi di mana A→C dan B→C sehingga C dikatakan memiliki ketergantungan transitif terhadap A melalui B. Ketergantungan transitif adalah tipe ketergantungan fungsional.

Relasi 2NF ke 3NF meliputi pembuangan ketergantungan transitif dengan cara menghapus atribut yang memiliki ketergantungan transitif tersebut dari relasi dengan menempatkan atribut tersebut dalam relasi baru bersamaan dengan tiruan (*copy*) determinan.

Contoh 3NF:

Patient No, Drug No, Start Date, Ward No, Bed No, Units per Day, Finish Date Drug No, Name, Description, Dosage, Method of Admin Patient No, Full Name Ward No, Ward Name

2.12 *Data Flow Diagram*

Menurut Whitten et al. (2004, p372), konteks data flow diagram adalah sebuah model proses yang digunakan untuk mendokumentasikan ruang lingkup untuk sebuah sistem.

Rung lingkup setiap proyek mendefinisikan apa aspek dari sistem atau aplikasi bisnis yang mendukung dan bagaimana cara sebuah sistem dibuat harus berinteraksi dengan sistem lainnya.

Berikut langkah-langkah untuk mendokumentasikan ruang lingkup:

a. Bayangkan sistem sebagai sebuah wadah untuk membedakan isinya dari luar. Hiraukan proses yang ada di dalam wadah. Ini disebut dengan pemikiran "*black box*".
b. Tanyakan kepada pengguna sistem transaksi bisnis yang harus direspon.
c. Tanyakan kepada pengguna respon yang harus dihasilkan sistem.
d. Identifikasikan *external data stores*.
e. Gambar diagram konteks berdasarkan informasi-informasi yang didapat.

2.13 UML (*Unified Modeling Language*)

UML menurut Priestley (2000, p7), seperti namanya, adalah suatu penggabungan dari beberapa bahasa model obyek orientasi sebelumnya. UML adalah usulan standar dalam pembuatan spesifikasi dari bermacam-macam komponen dari sistem piranti lunak, Silberschatz et al. (2002, p68). Menurut Burrows dan Langford (2003, p11), UML adalah bahasa untuk menspesifikasi, memperlihat, membangun, dan mendokumentasikan artifak dari sistem piranti lunak, baik sebagai model bisnis dan sistem *non-software* lainnya.

Ada beberapa tujuan dari pengembangan UML. Pertama dan yang paling penting, UML adalah bahasa *modeling* yang umum yang bisa digunakan oleh semua perancang.

UML tidak terikat dan berdasarkan pada persetujuan umum dari banyak komunitas komputer. Berarti meliputkan konsep dari metode utama sehingga bisa digunakan sebagai bahasa *modeling*-nya.

UML bukan dimaksudkan untuk menyelesaikan pengembangan metode, dan juga bukan termasuk tahapan dari proses perkembangan. Penting untuk menyadari bahwa UML dan sebuah proses menggunakan adalah dua hal yang terpisah. UML dimaksudkan untuk mendukung semua, setidaknya sebanyak mungkin proses pengembangan yang ada. UML meliputi semua konsep yang penting untuk mendukung proses dasar modern dalam pembuatan sebuah arsitektur kuat untuk menyelesaikan kebutuhan *use-case-driven*.

Tujuan akhir dari UML adalah menjadi sesederhana mungkin ketika masi bisa me-*modeling* jangkauan penuh dari sistem praktis yang perlu dibuat. UML perlu cukup ekspresif untuk mengatasi semua konsep yang dibangun di sistem modern, seperti *concurrency* dan distribusi, baik mekanisme rekayasa piranti lunak seperti enkapsulasi dan komponen-komponen.

2.13.1 Area Konsep UML

Konsep dan model UML dapat dikelompokkan dalam area konsep berikut:

a. *Static structure*

Beberapa model tepat pertama harus mendefinisikan tulisan semesta, yaitu, kunci konsep dari aplikasi, properti internalnya, dan hubungannya masing-masing. Kumpulan dari konstruksi ini adalah *static view*. Konsep aplikasi dibentuk sebagai *class*, setiap dari yang mendeskripsikan sekelompok obyek yang memegang informasi anda berkomunikasi untuk mengimplemen sifat.

Static view digambarkan dengan class diagram. *Static view* bisa digunakan untuk menghasilkan banyak deklarasi struktur data dalam sebuah program. Ada beberapa jenis elemen pada diagram UML, seperti *interface*, tipe data, *use case*, dan sinyal. Jika dikumpulkan, elemen-elemen tersebut disebut *classifiers* (kelompok atau golongan), dan bersifat sangat mirip dengan *class* dengan batasan tertentu pada setiap jenis *classifier*.

b. *Dynamic behavior*

Ada dua cara untuk membuat *behavior* (perilaku). Yang pertama adalah sejarah hidup dari satu obyek yang berinteraksi dengan dunia, satunya lagi adalah pola komunikasi dari sekumpulan obyek yang berhubungan yang berinteraksi untuk mengimplemen *behavior*.

c. *Implementation constructs*

Model UML berguna untuk analisis logikal dan implementasi fisikal. Konstruksi tertentu menghasilkan hasil implementasi berupa items (benda). Komponen merupakan yang fisikal, bagian dari sistem tergantikan yang menyesuaikan dan menyediakan realisasi dari sekumpulan *interface*.

d. *Model organization*

Komputer bisa menghadapi model besar yang datar, tetapi manusia tidak. Pada sistem yang besar, informasi modeling harus bisa dibagi menjadi bagian yang dimengerti sehingga team bisa bekerja dalam bagian-bagian yang berbeda. Bahkan pada sistem yang lebih kecil, pemahaman manusia membutuhkan isi model organisasi ke dalam paket yang berukuran kecil. Paket adalah unit hierarki organisasi yang umum dari model UML. Paket bisa digunakan untuk menyimpan, pengontrolan akses, pengaturan

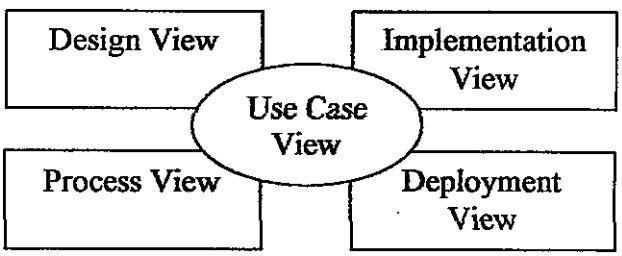
konfigurasi, dan pembuatan pustaka yang berisi pecahan model yang bisa digunakan ulang.

e. *Extensibility mechanisms*

Tidak peduli seberapa lengkap fasilitas dalam bahasa, orang masih akan membuat pengembangan sehingga harus disediakan kemampuan pengembangan yang terbatas pada UML yang dipercaya akan menampung sebagian besar hari-hari yang diperlukan untuk pengembangan, tanpa membutuhkan perubahan pada dasar bahasa.

2.13.2 Views

UML dituturkan dari tampilan dari struktur sistem piranti lunak yang dikenal dengan *4+1 view model.*



Gambar 2.5 "4+1 view model"

Use case view mendefinisikan perilaku sistem eksternal dan ketertarikan pada *end users*, analis, dan penguji. View ini mendefinisikan keperluan sistem, dan karena itu memaksa view yang lain, yang mendeskripsikan beberapa aspek dari rancangan sistem atau konstruksi.

Design view mendeskripsikan strutur logikal yang mendukung keperluan fungsional yg terdapat pada use case view. Berisi definisi dari komponen program, golongan-golongan penting, bersama dengan spesifikasi data yang dipegang, dan perlakunya dan interaksi.

Implementation view mendeskripsikan komponen fisikal di luar dari sistem yang akan dibangun.

Process view bertransaksi dengan persoalan yang beredar di dalam sistem, dan *deployment view* mendeskripsikan bagaimana komponen fisikal didistribusikan melalui lingkungan fisikal, seperti jaringan komputer, di mana sistem itu berjalan.

Beberapa tipe diagram UML adalah:

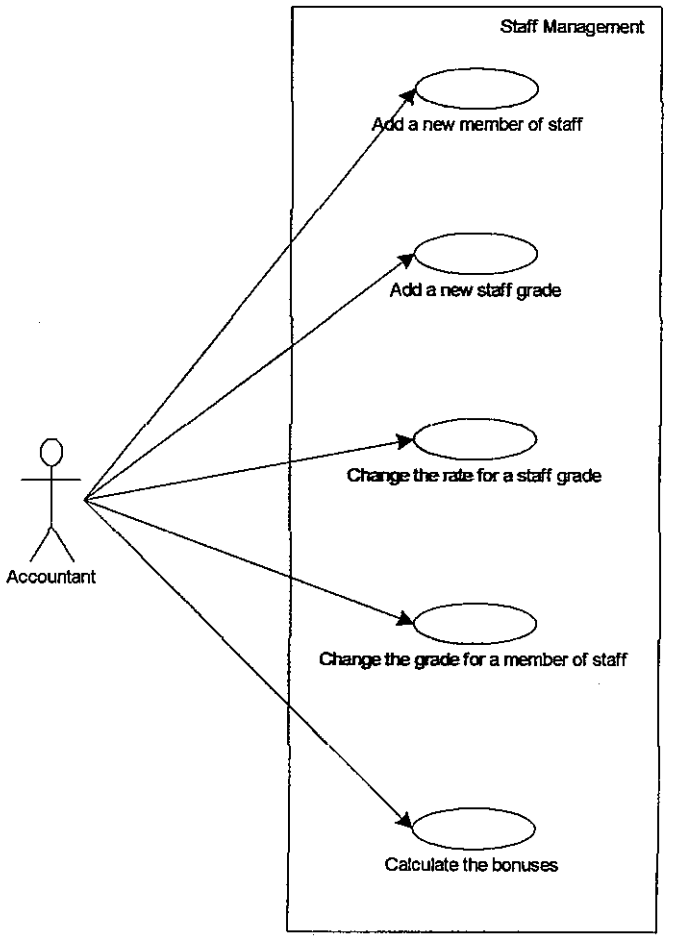
a. *Class diagram*. *Class diagram* ini mirip dengan ERD.
b. *Use case diagram*. *Use case diagram* menunjukkan interaksi antara pengguna dan sistem, biasanya adalah tahapan dari *task* yang dikerjakan oleh pengguna.
c. *Activity diagram*. *Activity diagram* menggambarkan aliran *task* antara bermacam-macam komponen di sistem.
d. *Implementation diagram*. *Implementation diagram* menunjukkan komponen sistem dan hubungannya, kedua-duanya pada tingkat komponen piranti lunak dan tingkat komponen perangkat keras.

2.13.3 Use Case Diagram

Use case adalah deskripsi dari fungsi sistem dari pandangan *user*. Use case menggambarkan interaksi antara *user* dan sistem. Diagram *use case* digunakan untuk menampilkan fungsi dari sistem yang akan menyediakan dan menunjukkan *user* mana yang akan berkomunikasi dengan sistem dalam beberapa cara dalam penggunaan fungsi tersebut.

Tujuan dari *use case* adalah untuk mendefinisikan bagian dari *behavior* yang bertahan tanpa memgambarkan struktur internal sistem.

Contoh diagram *use case* :



Gambar 2.6 Use Case Diagram

2.13.4 State Transition Diagram (STD)

State transition diagram adalah diagram yang digunakan untuk menggambarkan proses dari terjadi dalam sebuah sistem dalam hubungannya dengan representasi kondisi (*state*) dari sistem itu sendiri. Dalam STD minimal memiliki *start state* dan *finish state*.

2.14 *Microsoft Visual Basic.NET sebagai Front End*

Microsoft Visual Basic.Net yang merupakan perkembangan dari Microsoft Visual Basic, adalah sebuah alat yang digunakan untuk membangun aplikasi dengan menganut paradigma bahasa pemograman berorientasi obyek.

2.14.1 Pemecahan Masalah dengan Lingkungan Event-Driven

Banyak aplikasi komputer yang dipakai sekarang adalah *event driven*. Yang berarti bahwa aplikasi bekerja berdasarkan kejadian yang dideteksi oleh aplikasi dan atau hanya diam untuk menunggu sebuah kejadian. Lingkungan *event driven* adalah keadaan di mana sebuah aplikasi merespon terhadap kejadian yang dibuat oleh pengguna (seperti ketika mengklik tombol) atau oleh sistem (seperti ketika pesan *e-mail* tiba).

Pada lingkungan *event driven*, kode yang digunakan untuk memecah masalah disusun ,menjadi unit-unit kecil yang disebut *event procedure* yang didesain untuk merespon setiap kejadian. Contohnya, jika pengguna sedang mengisi nilai pada *field zip code*, akan ada kejadian yang didetek setiap kali pengguna mengetik karakter. Dengan meliputi pecahan kode kecil yang disebut *event handler* atau *event procedure*, program akan melihat apa yang pengguna masukkan dan jika nilainya tidak valid, akan menolak ketikan dan memberitahu masalah pada pengguna.

2.14.2 Komponen Visual dan NonVisual

Obyek pada komponen software dapat dikelompokkan menjadi dua, yaitu komponen visual dan nonvisual. Komponen visual adalah obyek pada sebuah program yang bisa tampak pada GUI (*graphical user interface*), sedangkan komponen nonvisual adalah obyek pada sebuah program yang tidak tampak pada GUI.

2.15 Lembur

Lembur adalah suatu kondisi di mana seseorang bekerja di luar jam kerja normal yang ditetapkan oleh suatu instansi.

2.16 Absensi

Absensi merupakan suatu cara pemantauan dan pencatatan kehadiran seseorang yang merupakan anggota suatu organisasi sebagai salah satu kedisiplinan kerja dari segi waktu yang dihitung berdasarkan jam masuk dan jam pulang.

2.17 Sistem Biometrik

Berikut ini adalah definisi dari sistem biometrik berserta cara kerja dan karakteristiknya.

2.17.1 Definisi Biometrik

Biometrik merupakan suatu pendekatan dari pengenalan sidik jari (*fingerprints*), wajah, geometri telapak tangan, retina dan iris yang kemudian dihubungkan dengan basis data komputer yang ada sehingga dapat dicocokkan.

2.17.2 Cara Kerja Sistem Biometrik

Pertama-tama, sistem biometrik akan mengambil sampel sesuai dengan jenis sistem biometrik yang dipakai (fitur), seperti merekam sinyal suara digital untuk pengenalan suara, atau mengambil gambar digital berwarna untuk pengenalan wajah, Sampel tersebut kemudian diubah menggunakan fungsi matematika ke dalam *template* biometrik. *Template*

biometrik akan menyediakan representasi normal, efisien dan akurat dari fitur, yang kemudian dapat dibandingkan secara objektif dengan *template* lain untuk pengenalan identitas. Kebanyakan sistem biometrik mengijinkan dua *mode* operasi. *Enrollment mode* untuk menambah *template* ke dalam basis data, dan *Mode* identifikasi dimana *template* dibuat secara individual dan kemudian dicocokkan dengan *pre-enrolled templates* yang telah ada di dalan basis data.

2.17.3 Karakteristik Sistem Biometrik

Sistem biometrik yang baik harus memiliki karakteristik:

a. Memiliki tingkat keunikan yang tinggi, sehingga kemungkinan yang sangat kecil adanya dua orang yang memiliki karakteristik yang sama;
b. Stabil, yaitu tidak berubah-ubah dari waktu ke waktu;
c. Mudah dalam pengambilan *template*-nya.

2.17 Sidik Jari Manusia

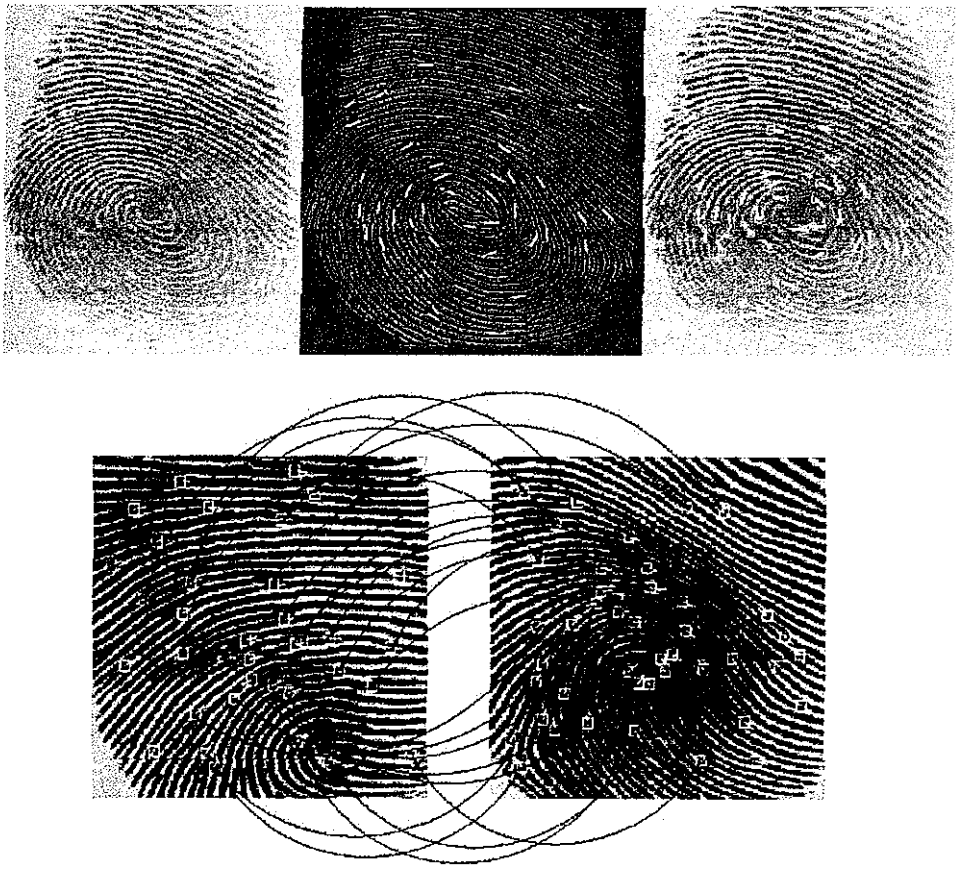
Berikut ini adalah beberapa penjelasan tentang sidik jari manusia yang terbagi menjadi seperti berikut.

2.17.1 Identifikasi Sidik Jari

Di antara semua teknik biometrik, identifikasi sidik jari merupakan metode yang paling tua dan berhasil digunakan pada banyak aplikasi. Setiap orang diketahui memiliki sidik jari yang tetap dan unik. Sebuah sidik jari tersusun dari sekelompok *ridge* (lengkungan) dan *furrow* (kerutan) pada permukaan jari. Keunikan sidik jari dapat ditentukan dari motif

lengkungan dan kerutan beserta titik-titik *minutiae*. Titik *minutiae* merupakan karakteristik lengkungan lokal yang muncul pada percabangan atau akhir lengkungan.

Teknik identifikasi sidik jari dapat dibedakan menjadi dua kategori; yaitu berbasis *minutiae* dan berbasis korelasi. Teknik berbasis *minutiae* mencari titik *minutiae* terlebih dahulu dan kemudian melakukan *mapping* posisi relatifnya pada jari. Akan tetapi, terdapat beberapa kesulitan ketika menggunakan pendekatan ini. Dalam pendekatan ini, sangatlah sulit dalam melakukan ekstraksi titik *minutiae* secara akurat ketika sidik jari dalam kualitas rendah. Selain itu, metode ini tidak melihat motif *ridge* dan *furrow* secara umum. Teknik kedua adalah teknik berbasis korelasi. Teknik ini mampu menangani masalah yang ditemukan pada pendekatan berbasis *minutiae*. Akan tetapi, teknik ini juga memiliki masalah sendiri. Teknik korelasi membutuhkan lokasi yang tepat dari titik registrasi dan dipengaruhi oleh translasi dan rotasi gambar.



Gambar 2.7 Pola Sidik Jari

Identifikasi sidik jari berbasis *minutiae* memiliki masalah dalam mengenal motif *minutiae* yang berbeda ukuran (tidak terdaftar). Struktur lokal *ridge* tidak dapat dikarakterisasi secara sempurna oleh *minutiae*.

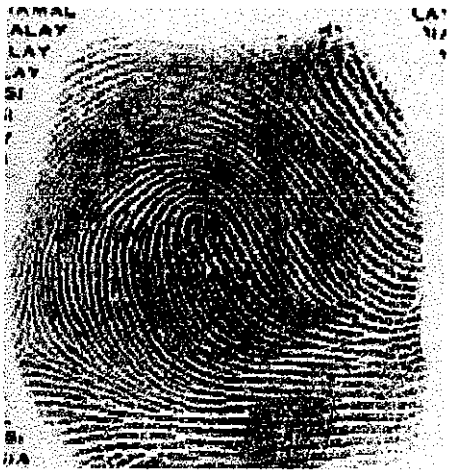
Sistem autentikasi komersial berbasis sidik jari memerlukan *False Reject Rate* yang sangat rendah untuk setiap *False Accept Rate*. Tentu saja, hal tersebut sangat sulit diperoleh. Dalam aplikasinya, sensor, sistem pengambil dan kecepatan sistem sangatlah kritikal.

2.17.2 Klasifikasi Sidik Jari

Klasifikasi sidik jari adalah teknik untuk menempatkan sidik jari ke dalam satu dari beberapa tipe yang ditentukan dalam literatur yang dapat menyediakan mekanisme *indexing*. Sidik jari yang dimasukkan terlebih dahulu dibandingkan dengan tipe yang ditentukan sebelumnya dan kemudian, barulah dibandingkan ke dalam subset dari basis data yang berisi tipe sidik jari itu saja.

2.17.3 Perbaikan Gambar Sidik Jari

Suatu langkah yang sangat kritikal dalam membandingkan sidik jari adalah secara otomatis dan akurat mengekstrak *minutiae* dari gambar sidik jari yang dimasukkan. Akan tetapi, kecepatan dari algoritma ekstraksi *minutiae* sangat bergantung kepada kualitas gambar sidik jari yang dimasukkan. Untuk memastikan kecepatan sistem identifikasi/verifikasi sidik jadi otomatis, sangatlah penting untuk membuat algoritma perbaikan gambar sidik jari dalam modul ekstraksi *minutiae*.



Gambar 2.8 Sidik Jari

Gambar 2.9 Sidik Jari dengan Perbaikan