BAB 2

LANDASAN TEORI

2.1 Teori-Teori Umum

Dalam menganalisis dan merancang sebuah sistem diperlukan teori-teori umum yang menjadi dasar pengetahuan. Berikut akan dijabarkan teori-teori umum yang berkaitan dengan sistem infomasi.

2.1.1 Pengertian Sistem

Menurut O'Brien (2010, p26) Sistem adalah sekelompok komponen yang saling bekerja sama menuju tujuan bersama dengan menerima input dan menghasilkan output dalam proses transformasi yang terorganisir.

2.1.2 Pengertian Informasi

Menurut O'Brien (2010, p34) Informasi adalah data yang telah dikonversi ke dalam konteks yang bermakna dan berguna bagi pengguna akhir tertentu.

Menurut Jogiyanti (2005, p36) Informasi adalah data yang diolah menjadi bentuk yang berguna bagi para pemakainya.

Dari definisi-definisi tersebut dapat ditarik kesimpulan bahwa informasi adalah data yang telah diolah dan menjadi memiliki kegunaan bagi pemakai.

2.1.3 Pengertian Sistem Informasi

Menurut O'Brien (2010, p4) Sistem Informasi dapat berupa kombinasi yang terorganisir antara orang, perangkat keras, perangkat lunak, jaringan komunikasi, dan sumber daya yang terkumpul, berubah, dan menyebarkan informasi dalam sebuah organisasi. Manusia bergantung pada sistem informasi untuk melakukan komunikasi dengan peralatan fisik (*hardware),* instruksi pemrosesan informasi atau prosedur (*software),* jaringan komunikasi (*network),* dan data *(data resources).*

Manusia, perangkat keras, perangkat lunak, data, dan jaringan merupakan 5 (lima) komponen utama yang diperlukan sebuah sistem informasi. Sumber daya manusia meliputi pengguna akhir (*end-user)* dan spesialis sistem informasi, sumber daya perangkat keras meliputi mesin dan medianya, sumber daya perangkat lunak meliputi program-program dan prosedur, sumber daya data meliputi data itu sendiri, sumber daya jaringan meliputi media komunikasi dan pendukung jaringan.

2.1.4 Komponen-Komponen Sistem Informasi

Menurut O'Brien (2010,p32) Semua sistem informasi menggunakan sumber daya manusia, *hardware*, *software*, data, dan jaringan untuk melakukan aktivitas input, pemrosesan, output, penyimpanan, dan pengendalian yang mengubah sumber daya data menjadi produk informasi.

2.1.4.1 Sumber daya manusia

Menurut O'Brien (2010, p32) manusia dibutuhkan untuk pengoperasian semua sistem informasi. Sumber daya manusia ini meliputi pemakai akhir dan pakai SI.

a. Pemakai Akhir

   Adalah orang-orang yang menggunakan sistem informasi atau informasi yang dihasilkan sistem tersebut.

b. Pakar SI

   Adalah orang-orang yang mengembangkan dan mengoperasikan sistem informasi.

2.1.4.2 Sumber Daya *Hardware*

Menurut O'Brien (2010, p32) konsep sumber daya *hardware* meliputi semua peralatan dan bahan fisik yang digunakan dalam pemrosesan informasi. Contoh-contoh *hardware* dalam sistem informasi berbasis komputer adalah :

a. Sistem Komputer, yang terdiri dari unit pemrosesan pusat yang berisi pemroses mikro, dan berbagai peralatan *peripheral* yang saling berhubungan.
b. Periferal Komputer, yang berupa peralatan seperti *keyboard* atau *mouse* elektronik untuk input data dan perintah, layar video, atau printer untuk output informasi, dan disk magnetis atau optikal untuk menyimpan sumber daya data.

2.1.4.3 Sumber Daya *Software*

Menurut O'Brien (2010, p33) Konsep sumber daya *software* meliputi semua rangkaian perintah pemrosesan informasi. Konsep umum *software* ini meliputi tidak

hanya rangkaian perintah operasi yang disebut program, dengan *hardware* komputer pengendalian dan langsung, tetapi juga rangkaian perintah pemrosesan informasi yang disebut prosedur. Berikut ini adalah contoh-contoh sumber daya *software* :

a. *Software* sistem, seperti program sistem operasi, yang mengendalikan serta mendukung operasi sistem komputer.
b. *Software* aplikasi, yang memprogram pemrosesan langsung bagi penggunaan tertentu komputer oleh pemakai akhir. Contohnya adalah program analisis penjualan, program penggajian, dan program pengolah kata (*word processing*).
c. Prosedur**,** yang mengoperasikan perintah bagi orang-orang yang akan menggunakan sistem informasi. Contohnya adalah perintah untuk mengisi formulir kertas atau menggunakan *software.*

2.1.4.4 Sumber Daya Data

Menurut O'Brien (2010, p33) data lebih daripada hanya bahan baku mentah sistem informasi. Data dapat berubah banyak bentuk, termasuk data alfanumerik tradisional, yang terdiri dari angka dan huruf serta karakter lainnya yang menjelaskan traksaksi bisnis dan kegiatan serta entitas lainnya. Data teks, terdiri dari kalimat dan paragraph yang digunakan dalam menulis komunikasi, data gambar, seperti bentuk grafik dan angka, serta gambar video grafis dan video; serta data audio, suara manusia dan suara-suara lainnya, juga merupakan bentuk data yang penting.

Sumber daya sistem informasi umumnya diatur, disimpan, dan diakses oleh berbagai teknologi pengelolaan sumber daya data ke dalam database yang menyimpan data yang telah diproses dan diatur.

Data pengetahuan yang menyimpan pengetahuan dalam berbagai bentuknya, seperti fakta, peraturan, dan contoh kasus mengenai praktik bisnis yang berhasil baik.

2.2 Teori Khusus yang Berhubungan dengan Topik yang Dibahas

2.2.1 Pengertian Data

Menurut O'Brien (2010. P34) data adalah jaman dari datum, meskipun umumnya merupakan bentuk tunggal dan jamak. Data adalah fakta atau pengamatan baku, biasanya sekitar fenomena atau transaksi-transaksi bisnis.

2.2.2 *Database*

Menurut Connolly dan Begg (2005, p15) *Database* merupakan Kumpulan yang terbagi dari data yang terkait dengan logika, dan sebuah deskripsi dari data, didesain untuk memenuhi kebutuhan informasi dari sebuah organisasi.

Menurut O'Brien (2010,p173), *Database* merupakan kumpulan terpadu dari elemen data logis yang saling berhubungan. *Database* mengonsolidasi banyak catatan yang sebelumnya disimpan dalam file terpisah agar kelompok data yang sama menyediakan banyak aplikasi.

Dari definisi-definisi tersebut, dapat disimpulkan bahwa *database* adalah kumpulan dari data yang terkait dengan logika yang saling berhubungan, yang didesain untuk menghasilkan informasi yang dibutuhkan oleh perusahaan.

2.2.3 Komponen Database

Menurut Connolly dan Begg (2005, pp18-21), komponen *database* terbagi menjadi lima komponen yaitu :

a. *Hardware*

DBMS memerlukan *hardware* untuk berfungsi. *Hardware* dapat berupa PC, *mainframe,* jaringan komputer. Sebagian *hardware* tergantung pada kebutuhan organisasi dan DBMS yang digunakan. Sebagian DBMS berfungsi hanya pada *hardware* tertentu atau sistem operasi yang khusus. Sebuah DBMS memerlukan sejumlah memori dan *disk space* untuk bekerja, tetapi konfigurasi yang minimum mungkin tidak dapat memberikan kinerja yang sesuai.

b. *Software*

Komponen *software* terdiri dari *software* DBMS itu sendiri, program aplikasi bersama dengan sistem operasi dan jaringan *software,* jika DBMS digunakan melalui jaringan. Program aplikasi dapat dibangun dengan menggunakan bahasa pemrograman seperti C++, Pascal, Delphi, atau Visual Basic.

c. Data

Bagi sisi pemakai, komponen terpenting dalam DBMS adalah data karena dari data inilah pemakai dapat memperoleh informasi yang sesuai dengan kebutuhan masing-masing. Data berfungsi sebagai penghubung antara komponen mesin dan komponen manusia. *Database* berisi data operasional dan metadata.

d. Prosedur

Prosedur mengacu pada instruksi dan aturan yang memerintahkan perancangan dan penggunaan *database*. Pengguna dan petugas yang mengatur *database* memerlukan

dokumentasi prosedur yang menjelaskan bagaimana untuk menggunakan atau menjalankan sistem. Instruksi ini mungkin berisi bagaimana untuk :

- Cara masuk ke dalam DBMS
- Menggunakan fasilitas DBMS tertentu atau aplikasi program
- Memulai dan menghentikan DBMS
- Membuat *back up* dari *database*
- Menangani kesalahan dari *hardware* atau *software*
- Merubah struktur dari tabel, mengorganisir ulang *database*, meningkatkan kinerja, atau mengarsipkan data ke dalam *secondary strorage*.

e. Manusia

Komponen manusia terdiri dari :

- Data Administrator : orang yang bertanggung jawab untuk mengatur sumber data termasuk perencanaan *database*, pengembangan dan pemeliharaan standar, kebijakan dan prosedur, dan *conceptual/logical* perancangan *database.*
- *Database* Administrator : orang yang menyediakan dukungan teknis untuk implementasi keputusan tersebut, dan bertanggung jawab pada realisasi secara fisik atas *database* pada tingkat teknis.
- *Database Designer* : ada dua tipe desainer, yang pertama *logical database designer* yang berfokus pada identifikasi data, *relationship,* antara data dan kendala data untuk disimpan dalam *database*. Lalu yang kedua adalah *physical database designer* yang memutuskan bagaimana *logical database design* dapat di realisasi secara fisik.
- *Application developers* : orang yang bertanggung jawab untuk mengimplementasikan aplikasi program yang membutuhkan

fungsionalitas untuk *end users.*

- *End User* : '*client'* untuk *database* yang telah dirancang dan di implementasi untuk menyediakan informasi yang dibutuhkan

2.2.4 Database Management System (DBMS)

Menurut Connoly dan Begg (2005, p16) Sebuah sistem perangkat lunak yang memungkinkan pengguna untuk menetapkan, membuat, memelihara, dan mengendalikan akses ke database.

Menurut Kadir (2003, p54) DBMS adalah perangkat lunak sistem yang memungkinkan para pemakai membuat, memelihara, mengontrol, dan mengakses basis data dengan cara yang praktis dan efisien.

Dalam bukunya, Kadir menerangkan bahwa umunya DBMS menyediakan fitur-fitur sebagai berikut :

a. Independensi data-program

Karena basis data ditangani oleh DBMS, program dapat ditulis sehingga tidak tergantung pada struktur data dalam basis data. Dengan kata lain, program tidak akan terpengaruh sekiranya bentuk fisik data diubah.

b. Keamanan

Keamanan dimaksudkan untuk mencegah pengaksesan data oleh orang yang tidak berwenang.

c. Integritas

Hal ini ditujukan untuk menjaga agar data selalu dalam keadaan yang valid dan konsisten.

d. Konkurensi

Konkurensi memungkinkan data dapat diakses oleh banyak pemakai tanpa menimbulkan masalah.

e. Pemulihan (recovery)

DBMS menyediakan mekanisme untuk mengembalikan basis data ke keadaan semula yang konsisten sekiranga terjadi gangguan perangkat keras atau kegagalan perangkat lunak.

f. Katalog system

Katalog sistem adalah deskripsi tentang data yang terkandung dalam basis data yang dapat diakses oleh pemakai.

g. Perangkat produktivitas

Untuk menyediakan kemudahan bagi pemakai dan meningkatkan produktivitas, DBMS menyediakan sejumlah perangkat produktivitas sebagai pembangkit query dan pembangkit laporan.

2.2.5 Keuntungan DBMS

Menurut Connolly dan Begg (2005, p26-30), DBMS memiliki banyak keuntungan antara lain :

2.2.5.1 Kontrol terhadap pengulangan data (*data redundancy*)

Database berusaha untuk menghilangkan pengulangan dengan mengintegrasikan *file* sehingga berbagai *copy* daru data yang sama tidak tersimpan. Bagaimanapun juga, pendekatan ini tidak menghilangkan pengulangan secara menyeluruh, tetapi mengendalikan jumlah pengulangan dalam *database*.

2.2.5.2 Konsisten Data

Dengan menghilangkan atau mengendalikan pengulangan, kita telah mengurangi resiko ketidak-konsistenan yang terjadi. Jika *item* data disimpan hanya sekali didalam *database*, maka berbagai *update* bagi nilai data tersebut harus dibuat hanya sekali dan nilai baru tersebut harus tersedia untuk semua pengguna. Jika item data disimpan lebih dari sekali, sistem dapat memastikan bahwa semua *copy* dari item tersebut tetap konsisten.

2.2.5.3 Mendapatkan Informasi Tambahan

Dengan data operasional yang terintegrasi, hal ini memungkinkan bagi organisasi untuk mendapatkan infotmasi tambahan dari data yang sama.

2.2.5.4 Pembagian Data (*sharing of data)*

File biasanya dimiliki oleh orang atau departemen yang menggunakannya. Di sisi lain, basis data adalah milik keseluruhaan organisasi dan dapat dibagikan kepada setiap user yang memiliki otoritas untuk mengaksesnya. Dalam hal ini, banyak pengguna membagikan lebih banyak data.

2.2.5.5 Meningkatkan Integritas Data

Integritas *database* mengacu pada validitas dan konsistensi data yang disimpan. Integritas biasanya diekspresikan dalam istilah batasan, yang berupa aturan konsisten yang tidak boleh dilanggar oleh *database*. Integrasi memungkinkan DBA untuk menjelaskan, dan memungkinkan DBMS untuk membuat batasan integritas.

2.2.5.6 Meningkatkan Keamanan

Keamanan *database* melindungi data dari pengguna yang tak berotoritas. Hal ini dapat dilakukan dengan menggunakan *username* dan *password* untuk mengindetifikasikan orang yang berotoritas untuk menggunakan *database*. Akses pengguna yang berotoritas pada *database* mungkin dibatasi oleh jenis operasi seperti pengambilan, *insert*, *update*, dan *delete*.

2.2.5.7 Penetapan standarisasi

Integrasi memungkinkan DBA untuk mendifinisikan dan membuat standar yang diperlukan. Standar ini termasuk standar departemen, organisasi, nasional, atau internasional untuk memfasilitasi pertukaran data antar sistem, ketetapan penamaan, standar dokumentasi, prosedur *update*, dan aturan akses.

2.2.5.8 Pengurangan biaya

Dengan menyatukan semua data operasional organisasi kedalam satu *database* dan pembuatan sekelompok aplikasi yang bekerja pada satu sumber data dapat menghasilkan pengurangan biaya. Penyatuan biaya pengembangan dan pemeliharaan sistem pada setiap departemen akan menghasilkan total biaya yang lebih rendah. Sehingga biaya lainnya dapat digunakan untuk membeli konfigurasi sistem yang sesuai bagi kebutuhan organisasi.

2.2.5.9 Menyeimbangkan konflik kebutuhan

Setiap pengguna mempunyai kebutuhan yang mungkin bertentangan dengan kebutuhan pengguna lain. *Database* dikendalikan oleh DBA, DBA dapat membuat

keputusan berkaitan dengan perancangan dan penggunaan operasional *database* yang menyediakan penggunaan terbaik dari sumber daya bagi keseluruhan organisasi.

2.2.5.10 Meningkatkan kemampuan akses dan respon pada data

Dengan pengintegrasian data yang melintasi batasan departemen dapat dilihat secara langsung diakses oleh pengguna akhir. Hal ini dapat menyediakan sebuah sistem yang lebih banyak fungsinya seperti fungsi untuk menyediakan layanan yang lebih baik pada pengguna akhir atau klien organisasi. Banyak DBMS menyediakan *query languange* yang memungkinkan pengguna untuk menanyakan pertanyaan *ad-hoc* dan memperoleh informasi yang diperlukan dengan segera pada terminal mereka, tanpa memerlukan *programmer* menuliskan beberapa *software* untuk mengubah informasi tersebut dari *database*.

2.2.5.11 Meningkatkan produktivitas

DBMS menawarkan banyak fasilitas yang memudahkan dalam menyusun aplikasi sehingga waktu pengembangan aplikasi dapat diperpendek.

2.2.5.12 Meningkatkan pemeliharaan melalui independensi data

DBMS menyediakan pendekatan yang membuat perubahan dalam data tidak membuat program harus dirubah.

2.2.5.13 Meningkatkan konkurensi

Dalam beberapa sistem berbasis *file*, jika dua atau lebih pengguna mengakses *file* yang sama secara bersamaan, maka akses akan bertentangan satu sama lain, kehilangan

informasi atau bahkan kehilangan integritas. DBMS menyediakan mekanisme sehingga data yang sama dapat diakses oleh sejumlah orang dalam waktu yang bersamaan.

2.2.5.14 Peningkatan fasilitas *backup* dan pelayanan *recovery*

DBMS melindungi pengguna dari efek kegagalan sistem. Jika terjadi kegagalan, DBMS dapat mengembalikan data sebagaimana kondisi saat sebelum terjadi kegagalan melalui fasilitas *backup* dan *recovery database*.

2.2.6 Kerugian DBMS

a. Kompleksitas

Ketentuan dari fungsi yang kita harapkan dari DBMS yang baik membuat DBMS menjadi sebuah software yang sangat kompleks. Perancang dan pengembang database, DA, dan DBA, serta pengguna akhir harus memahami fungsi tersebut untuk mendapatkan banyak keuntungan dari DBMS ini.

b. Ukuran

Fungsi yang kompleks dan luar membuat DBMS menjadi software yang sangat besar, memerlukan banyak ruang hardisk dan jumlah memory yang sangat besar untuk beralan dengan efesien.

c. Biaya dari DBMS

Biaya DBMS bervariasi, tergantung pada lingkungan dan fungsi yang disediakan. Disitu juga terdapat buaya pemeliharaan tahunan yang juga dimasukan dalam daftar harga DBMS.

d. Biaya penambahan perangkat keras

Kebutuhan tempat penyimpanan bagi DBMS dan database amat memerlukan pembelian tempat penyimpanan tambahan. Lebih lanjut, untuk mencapai performa yang diperlukan, mungkin diperlukan untuk membeli mesin yang lebih besar lagi. Hal ini tentu memerlukan tambahan biaya yang tidak sedikikt. Tergantung pada spesifikasi perangkat keras yang diperlukan.

e. Biaya Perangkat Keras Tambahan

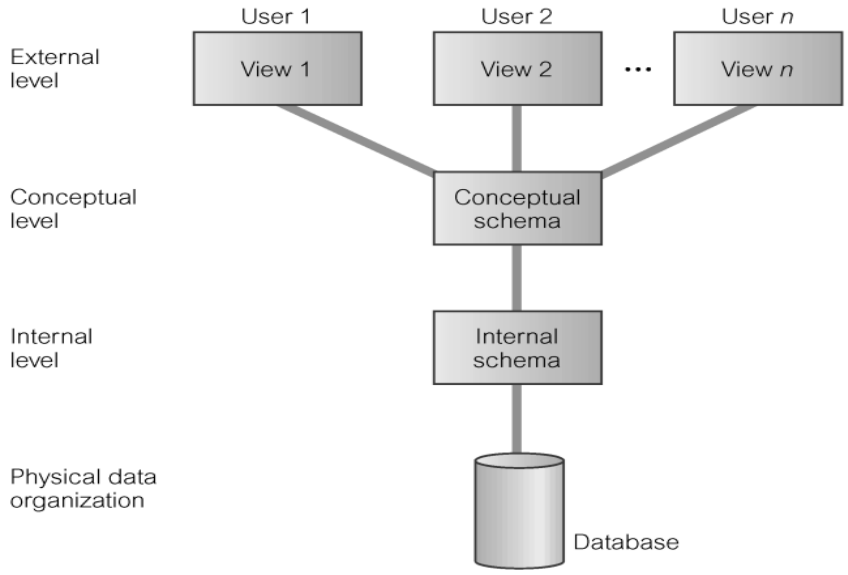
Biaya Tambahan untuk melakukan konversi aplikasi yang telah ada agar dapat berjalan pada DBMS dan perangkat keras yang baru. Selain itu biaya konversi ini meliputi biaya tambahan untuk pelatihan staff agar dapat menggunakan sistem baru dan bila memungkinkan bahwa bisa juga membutuhkan staff ahli untuk membantu dalam melakukan konversi dan menjalankan sistem baru.

f. Performa

DBMS digunakan untuk memenuhi banyak permintaan aplikasi sehingga beberapa aplikasi tidak berjalan sesuai dengan yang seharusnya.

2.2.7 Arsitektur ANSI-SPARC Three Level

Menurut Connolly dan Begg (2005, pp34-37), arsitektur ANSI-SPARC *three-level* dibagi menjadi tiga bagian yaitu :



Gambar 2.1 : Arsitektur ANSI-SPARC *three-level*

a. Level Eksternal

External level terdiri dari sejumlah *view database* untuk *user*, dimana masing-masing *user* hanya akan menangani satu bagian *view* saja. Setiap *user* memiliki *view* yang ditunjukan dalam bentuk yang mudah dimengerti oleh *user*. Eksternal *view* hanya terbatas pada entitas, atribut, dan hubungan antar entitas yang diperlukan. Entitas, atribut, dan hubungan antar entitas lain yang tidak tampil pada *view* tetap berada dalam *database*, tapi *user* mungkin tidak menyadarinya.

b. Level Konseptual

Conceptual view menjelaskan data apa saja yang disimpan dalam *database* dan hubungan antar data. Level ini berisi struktur *logical* yang

ada di dalam *database* seperti yang terlihat oleh DBA. *Conceptual level* menunjukan:

• Seluruh entitas, atribut, dan hubungannya

• Data *constraint*

• Informasi semantik tentang data

• Keamanan dan integritas informasi

c. Level Internal

Level ini menjelaskan bagaimana data disimpan dalam *database*. Internal level merepresentasikan keseluruhan *database* secara fisik. Internal level meliputi implementasi *database* untuk mengoptimalkan kinerja dan penyimpanan dalam *database*. Ini meliputi struktur data dan *file* organisasi yang digunakan untuk menyimpan data dalam *storage device*. Internal level berfokus pada:

• Ruang penyimpanan untuk data dan indeks

• Catatan deskripsi untuk penyimpanan

• Catatan penempatan data

• Teknik enkripsi data

Dengan kata lain level ini berkaitan dengan struktur penyimpanan/*database* yang tersimpan yang menerangkan tempat penyimpanan data pada *internal view*, dan definisi struktur penyimpanan pada skema internal yang menerangkan hubungannnya dengan cara pengaksesan data yang disimpan. Tujuan dari arsitektur *three-level* adalah untuk menyediakan data *independence* yang berarti bahwa level yang

lebih tinggi tidak terpengaruh oleh perubahan pada level yang lebih rendah.

2.2.8 Database Languange

Menurut Connolly dan Begg (2005, p39) sebuah data *sublanguage* terdiri dari dua bagian yaitu *Data Definition Language* (DDL) dan *Data Manipulation Language* (DML). DDL digunakan untuk menentukan skema *database* dan DML digunakan baik untuk membaca dan meng-*update database*.

2.2.8.1 Data Definition Languange (DDL)

Data Definition Language menurut Connolly dan Begg (2005, p40) adalah suatu bahasa yang mengijinkan DBA atau pengguna untuk mendeskripsikan dan memberi nama entitas, atribut, dan *relationship* yang diperlukan untuk aplikasi. DDL berfungsi untuk mengubah suatu data menjadi data yang berguna bagi pengguna. Beberapa statement DDL menurut Connolly dan Begg (2005, pp168-167) :

a. *Create Table*

Untuk membuat table dengan mengidentifikasikan tipe data untuk tiap kolom.

b. *Alter Table*

Untuk menambah atau membuang kolom dan konstrain.

c. *Drop Table*

Untuk membuang atau menghapus tabel beserta semua data yang terkait di dalamnya.

d. *Create Index*

Untuk membuat *index* pada suatu tabel.

e. *Drop Index*

Untuk membuang atau menghapus *index* atau menghaous *index* yang sudah dibuat sebelumnya.

2.2.8.2 Data Manipulation Languange (DML)

Menurut Connolly dan Begg (2005, p40) DML adalah suatu bahasa yang menyediakan sekumpulan operasi yang diinginkan untuk mendukung operasi manipulasi data utama pada data yang diperoleh dalam *database*. DML menyediakan operasi dasar manipulasi data pada data yang ada dalam *database*, yaitu :

- Penyisipan data baru kedalam *database (insertion)*
- Mengubah atau memodifikasi data yang disimpan dalam *database (modify)*
- Pemanggilan data yang ada dalam *database (retrieve)*
- Menghapus data dari *database (delete)*

2.2.9 Fungsi DBMS

Menurut Connolly dan Begg (2005, pp48-52), funsi DBMS adalah sebagai berikut :

- Penyimpanan, Pengambilan, dan Peng-*update*-an data

  Sebuah DBMS harus menyediakan sebuah kemampuan untuk menyimpan, mengambil, dan meng-*update* data dalam DBMS. Ini merupakan fungsi yang mendasar dari DBMS. Dalam menyediakan fungsi ini DBMS harus menyembunyikan detil implementasi fisikal internal seperti organisasi *file* dan struktur penyimpanan dari pengguna.

- Katalog *User-Accesible*

  Sebuah DBMS harus menyediakan sebuah katalog yang menyimpan deskripsi tentang item data dan mudah diakses oleh pengguna. Menurut Connolly dan Begg (2005, pp48-49), katalog sistem merupakan tempat penyimpanan informasi yang menjelaskan data dalam *database*, yaitu metadata atau data tentang data. Katalog sistem menyimpan informasi seperti berikut :

  - Nama, jenis, dan ukuran data
  - Nama *relationship*
  - Batasan integritas pada data
  - Nama pengguna yang berotoritas yang mempunyai akses pada data.
  - Skema eksternal, konseptual, dan internal dan pemetaan antara skema.
  - Penggunaan statistik, seperti frekuensi transaksi dan perhitungan sejumlah akses yang dibuat pada objek dalam *database.*

- Mendukung Transaksi

  Sebuah DBMS harus menyediakan mekanisme yang akan memastikan bahwa semua kegiatan *update* yang dilakukan sesuai dengan transaksi yang diberikan atau tidak ada kegiatan *update* yang dibuat bagi transaksi tersebut. Transaksi merupakan sederetan tindakan, yang dilakukan oleh pengguna tunggal atau program aplikasi yang mengakses atau mengubah isi *database*.

- Layanan Kendali Konkurensi

  Sebuah DBMS harus menyediakan sebuah mekanisme untuk memastikan bahwa *databse* di-*update* dengan benar ketika banyak pengguna meng-

update database secara bersama-sama. Akses bersama relatif mudah jika semua pengguna hanya membaca data. Namun, ketika dua atau lebih pengguna mengakses *database* secara serentak dan paling sedikit satu dari mereka meng-*update* data, di sana dapat terjadi gangguan yang menghasilkan ketidak-konsistenan.

- Layanan Perbaikan

  Sebuah DBMS harus menyediakan sebuah mekanisme untuk memperbaiki *database* disaat *database* mengalami kerusakan dalam berbagai cara. Kerusakan *database* dapat diakibatkan karena kerusakan sistem, kesalahan media, dan kesalahan *software* atau *hardware*. Atau disebabkan karena adanya kesalahan selama proses transaksi dan penyelesaian transaksi yang tidak lengkap.

- Layanan Authorisasi

  Sebuah DBMS harus menyediakan sebuah mekanisme untuk memastikan bahwa pengguna yang berotoritas dapat mengakses *database*. Hal ini untuk mencegah data yang tersimpan tak terlihat oleh semua pengguna dan melindungi *database* dari akses yang tak berotoritas.

- Mendukung Komunikasi Data

  Sebuah DBMS harus mampu mengintegrasikan dengan *software* komunikasi. Kebanyakan pengguna mengkases *database* dari *workstation*. Kadang *workstation* tersebut terhubung secara langsung ke komputer DBMS. Dalam kasus yang lain, *workstation* berada pada lokasi yang jauh dan berkomunikasi dengan komputer DBMS melalui jaringan. Dalam hal ini, DBMS menerima permintaan sebagai pesan komunikasi dan

menanggapi dengan cara yang sama. Semua pengiriman ini ditangani oleh *Data Communication Manager*.

- Layanan Integritas

  Sebuah DBMS harus menyediakan sebuah arti untuk memastikan bahwa data di dalam *database* dan perubahan pada data mengikuti aturan tertentu. Integritas *database* dapat mengacu pada kebenaran dan konsistensi data yang disimpan. Integritas berhubungan dengan kualitas data yang disimpan. Integritas biasanya diekspresikan dengan istilah batasan, yaitu berupa aturan konsistensi yang tidak boleh dilanggar oleh *database*.

- Layanan Peningkatan Keterbebasan Data

  Sebuah DBMS harus memasukkan sebuah fasilitas untuk mendukung keterbebasan program dari struktur *database* yang sebenarnya. *Data Independence* biasanya dicapai melalui sebuah view atau mekanisme subskema. *Physical data independence* lebih mudah untuk dicapai karena terdapat beberapa jenis perubahan yang dapat dibuat untuk karakteristik fisikal dari *database* tanpa mempengaruhi *view*. Bagaimanapun *data independence* logikal yang lengkap lebih susah untuk dicapai.

- Layanan Utilitas

  Sebuah DBMS harus menyediakan seperangkat layanan utilitas. Program utilitas membantu DBA mengelola *database* secara efektif. Beberapa utilitas bekerja pada tingkat eksternal, dan konsekuensinya dapat dibuat oleh DBA, yang lainnya bekerja pada tingkat internal dan dapat disediakan hanya dengan vendor DBMS. Contoh dari utilitas tersebut antara lain :

  • Fasilitas import, untuk meng-*load database* dari *flat file*, dan fasilitas

eksport, untuk meng-*unload database* pada *flat file*.

- Fasilitas pemantauan, untuk memantau penggunaan dan operasi *database*.
- Program analisa statistik, untuk memeriksa performa dan penggunaan statistik.
- Fasilitas penyusunan *index*, untuk menyusun kembali *index* dan *overflow* mereka.
- Penempatan dan pengumpulan sampah, untuk menghilangkan *record* yang dihapus secara fisik dari alat penyimpanan, untuk menggabungkan ruang yang terlepas, dan untuk menempatkan kembali *record* tersebut dimana ia dibutuhkan.

2.2.10 Komponen DBMS

Menurut Connolly dan Begg (2005, pp53-55), komponen dari sebuah DBMS adalah sebagai berikut :

a. *Querry Processor* : Merupakan komponen DBMS yang utama yang mengubah *query* ke dalam seperangkat instruksi tingkat rendah langsung ke *database manager*

b. *Database Manager* : DM menghubungkan program aplikasi *user-submitted* dan *query*. DM menerima *query* dan memeriksa skema eksternal dan konseptual untuk menentukan *record* konseptual apa yang diperlukan untuk memuaskan permintaan.

c. *File Manager* : *File Manager* memanipulasi penyimpanan *file* dan mengatur penempatan ruang penyimpanan dalam *disk*. Komponen ini

mendirikan dan memelihara daftar struktur dan *index* yang didefinisikan dalam skema internal.

d. *DML Preprocessor* : Modul ini mengubah pernyataan DML yang tertanam dalam program aplikasi ke dalam panggilan fungsi standar dalam *host languange*. Komponen ini harus berinteraksi dengan *query processor* untuk membuat kode yang sesuai.

e. *DDL Compiler* : Modul ini mengubah pernyataan DDL ke dalam seperangkat tabel berisi metadata. Tabel ini kemudian disimpan dalam katalog sistem sementara itu informasi kendali disimpan dalam *header file* data.

f. *Catalog Manager* : Mengatur akses ke sistem dan memelihara katalog sistem. Katalog sistem diakses oleh sebagian besar komponen DBMS.

2.2.11 Struktur Data Relasional

Menurut Connolly dan Begg (2005, pp72-74), struktur data relasional terbagi menjadi beberapa bagian yaitu :

1. Relasi : Merupakan sebuah tabel dengan baris dan kolom. Digunakan untuk menyimpan informasi tentang objek yang digambarkan dalam *database*.

2. Atribut : Merupakan nama kolom relasi. Atribut dapat ditampilkan dalam berbagai perintah dan dalam relasi yang sama sehingga menyampaikan arti yang sama.

3. *Domain :* Merupakan sekelompok nilai yang diijinkan bagi satu atau lebih atribut. Setiap atribut dalam relasi didefinisikan pada sebuah *domain*.

Domain dapat berbeda bagi setiap atribut, atau dua/lebih atribut dapat didefinisikan pada *domain* yang sama. Konsep *domain* sangat penting karena memungkinkan pengguna menjelaskan arti dan sumber nilai yang ada pada atribut.

4. *Tuple :* Merupakan baris dari sebuah relasi. *Tuple* dapat disebut *intention* jika struktur relasi, *domain* serta batasan-batasan yang lainnya pada nilai yang mungkin bersifat tetap, namun sebaliknya jika relasi berubah setiap waktu ini disebut *extension*.

5. *Degree :* Merupakan jumlah atribut yang terdapat dalam realsi. Jika relasi mempunyai satu atribut akan mempunyai derajat satu yang disebut relasi *unary/*satu *tuple*. Jika relasi mempunyai dua atribut akan mempunyai derajat dua yang disebut *binary*. Dan begitu seterusnya dengan menggunakan istilah *n-ary*.

6. *Cardinality :* Merupakan jumlah *tuple* yang terdapat dalam relasi. Merupakan properti dari *extension* relasi dan ditentukan dari *instance* tertentu.

7. *Database Relational :* Merupakan kumpulan dari relasi yang ternormalisasi dengan nama relasi yang berbeda.

2.2.12 *Relational Keys*

Menurut Connolly dan Begg (2005, pp78-79), relasional *key* dibagi menjadi beberapa jenis yaitu :

1. *Superkey :* Merupakan sebuah atribut atau sekelompok atribut yang mengidentifikasi secara unik *tuple* dalam relasi. *Superkey* yang mudah diidentifikasi adalah yang hanya berisi jumlah minimum atribut yang diperlukan.

2. *Candidate Key :* Merupakan *superkey* dalam relasi. *Candidate Key (K)*, bagi sebuah relasi (R) mempunyai dua sifat yaitu :

    - Keunikan : Dalam setiap *tuple* dari R, nilai dari K secara unik mengidentifikasi *tuple* tersebut.

    - *Irreducibility :* Tidak ada *subset* yang sesuai dari K yang mempunyai keunikan sifat. Ketika sebuah *key* terdiri dari lebih dari satu atribut kita sebut ini sebagai *composite key.*

3. *Primary Key :* Merupakan *candidate key* yang terpilih untuk identifikasi *tuple* secara unik dalam satu relasi. Sementara *candidate key* yang tak terpilih sebagai *primary key* disebut *alternate key*.

4. *Foreign Key :* Merupakan sebuah atribut atau sekelompok atribut dalam relasi yang dibandingkan dengan *candidate key* pada beberapa relasi.

2.2.13 *Entity Relationship Model*

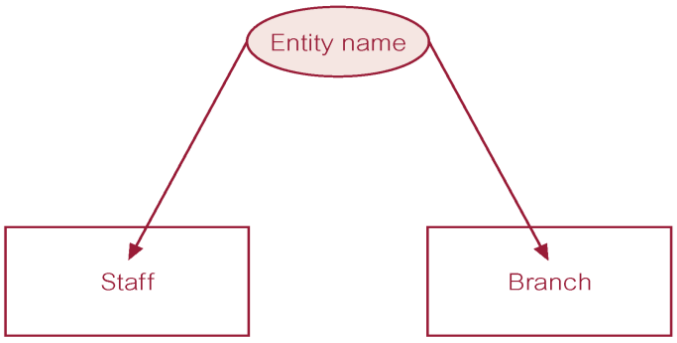
Menurut Connolly and Begg (2005), salah satu aspek terpenting dalam perancangan basis data adalah suatu kenyataan bahwa seorang perangcang, *programmer*, dan *end-user* cenderung dalam melihat data memiliki cara yang berbeda. Oleh karena itu, untuk memastikan pemahaman yang tepat tentang sifat data bagaimana data tersebut

digunakan oleh organisasi/perusahaan, maka dibutuhkan satu model untuk berkomunikasi yang bersifat non-teknis dan bebas dari ambiguitas. Salah satu contohnya adalah: *Entity Relationship Model*

Pemodelan ER merupakan pendekaran *top-down* dalam perancangan basis data yang dimulai dengan mengidentifikasi data penting yang dinamakan entitas dan relasi antara data yang harus direpresentasikan dalam model. Kemudian ditambahkan rincian lebih lanjut seperti informasi yang terdapat pada entity dan relasi yang dinamakan atribut dan batasan pada entitas, relasi dan atribut

2.2.13.1 Entitas

Menurut Connoly dan Begg (2005,p343) Entitas adalah sekumpulan objek yang memiliki sifat-sifat yang sama dan diidentifikasi oleh perusahaan sebagai objek yang mempunyai keberadaan yang bebas. Sebuah entitas dapat menjadi objek secara fisik (contohnya: karyawan, barang, pembeli) atau menjadi objek dengan keberadaan konseptual (contohnya: reservasi, pembayaran). Representasi diagram dari entitas ditunjukan sebagai segi empat berlabel nama entitas. Setiap objek dari suatu tipe entitas dapat diidentifikasikan secara unik. Yang disebut sebagai *entity occurrance*. Setiap entitas digambarkan dalam bentuk segi empat yang dijelaskan melalui nama dari entity tersebut, yang biasanya kata benda tunggal. Dalam UML huruf pertama setiap kata dalam nama entity berupa huruf capital.



Gambar 2.2 : Contoh Entitas

2.2.13.2 Atribut

Connoly dan Begg (2005, p350) Atribut adalah sifat dari entitas atau tipe *relationship*. Atribut menyimpan nilai yang menjelaskan setiap kejadian entitas dan menunjukan bagian utama dari data disimpan di dalam *database*.

Menurut Connoly dan Begg (2002, p351), setiap atribut dihubungkan dengan sekumpulan nilai yang disebut *domain*. Atribut *domain* adalah sekumpulan nilai yang diperbolehkan untuk satu atau lebih atribut. *Domain* sendiri diartikan sebagai nilai potensial yang attibut mungkin miliki.

Atribut dapat diklasifikasikan sebagai berikut :

- Atribut Sederhana (*Simple*)

  Simple Attribute merupakan atribut yang terdiri dari satu komponen tunggal dengan keberadaan-nya sendiri sehingga tidak dapat dipecah menjadi komponen-komponen yang lebih kecil lagi.
- Komposit (*Composite*)

  Composite Attribute merupakan atribut yang dibentuk dari banyak komponen.
- *Single Valued Attribute*

Single Valued Attribute merupakan atribut yang menyimpan nilai tunggal untuk setiap kemunculan tipe entitas.

- *Multi Valued Attribute*

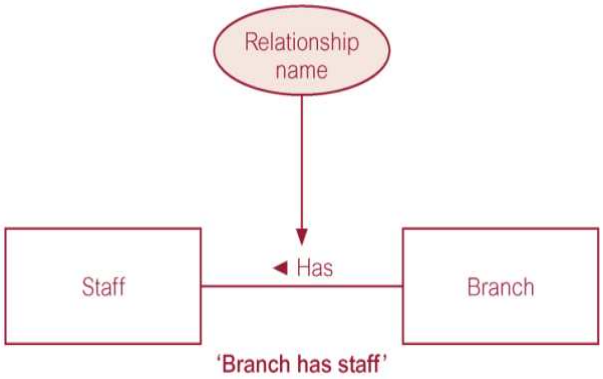
  Atribut *Multi Valued* merupakan atribut yang menyimpan banyak nilai untuk setiap kemunculan entitas.

- Atribut Turunan

  Merupakan atribut yang mereprensentasikan nilai hasil penurunan dari nilai satu atau beberapa atribut yang berhubungan dan tidak harus dari tipe entitas yang sama.

2.2.13.3 *Relationship*

Menurut Connoly dan Begg (2005, p346) *Relationship* adalah sekumpulan hubungan yang berarti antara satu atau lebih entitas. Setiap tipe hubungan diberi nama yang menggambarkan fungsi tersebut.

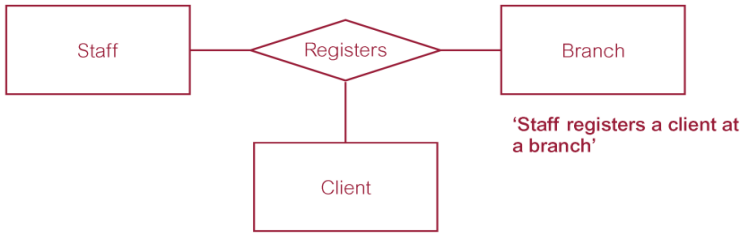


Gambar 2.3 : Contoh *Relationship*

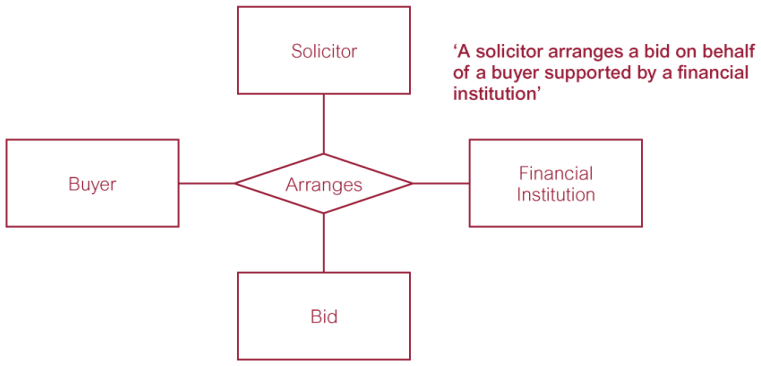
Menurut Connolly dan Begg (2005, p346) entitas yang dilibatkan dalam *relationship types* tertentu dinyatakan sebagai partisipan pada relasi tersebut. Jumlah partisipan pada tipe relasi tersebut dinamakan derajat tipe relasi. Relasi dengan derajat

dua disebut *binary*. Relasi dengan derajat tiga disebut *ternary*. Relasi dengan derajat empat disebut dengan *quarternary*.

Gambar 2.4 : Contoh tipe relasi *binary*

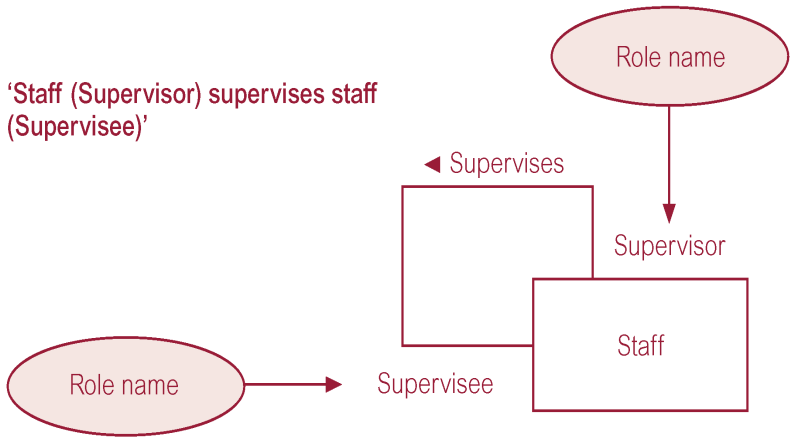


Gambar 2.5 : Contoh tipe relasi *ternary*



Gambar 2.6 : Contoh relasi *quarternary*

Menurut Connolly dan Begg (2005,p346) relasi *rekursive* merupakan tipe relasi dimana tipe entitas yang sama berpartisipasi lebih dari satu peranan yang berbeda.



Gambar 2.7 : Contoh tipe relasi *recursive*

2.2.13.4 Kunci

Menurut Connolly dan Begg (2005, p352) terdapat beberapa kunci yang dapat dibagi menjadi :

1. Kunci Kandidat (*Candidate key*)

   Kumpulan kecil dari atribut unik yang mengidentifikasi tiap-tiap kejadian dari sebuah tipe entitas. *Candidate key* bias lebih dari satu untuk setiap entitas.

2. Kunci Primer (*Primary Key*)

   Calon kunci yang terpilih untuk mengidentifikasi secara unik setiap kejadian pada tipe entitas. *Primary key* dari sebuah entitas hanya ada satu dan tidak dimiliki oleh entitas lain.

3. Kunci Gabungan (*Composite Key*)

Kunci kandidat yang terdiri dari dua atau lebih atribut. Dalam beberapa kasus, *key* dari entitas terdiri dari beberapa atribut yang bernilai unik untuk setiap *entity occurrence* tetapi tidak terpisah.

2.2.13.5 Entitas Kuat dan Lemah

Menurut Connolly dan Begg (2005, p342), Entitas kuat adalah entitas yang keberadaannya tidak bergantung pada beberapa entitas yang lain. Karakter dari entitas ini adalah bahwa setiap kejadia entitas teridentifikasi secara unik menggunakan atribut *primary key*. Sebagai contoh, kita dapat mengidentifikasi secara unik setiap anggota staf dengan menggunakan atribut StaffId.

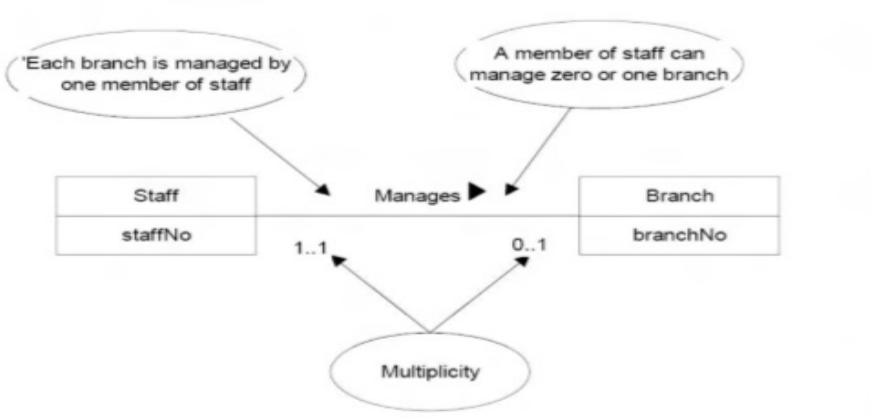
Sedangkan entitas lemah adalah entitas yang keberadaannya tergantung pada beberapa entitas yang lain. Karakteristik dari entitas ini bahwa setiap kejadian entitas tidak dapat teridentifikasi secara unik hanya dengan menggunakan atribut entitas tersebut, Kita hanya dapat mengidentifkasi secara unik entitas *Preference* melalui *relationship* yang dimiliki dengan entitas *Client* yang secara unik teridentifikasi menggunakan *primary key* bagi entitas *Client*.

Gambar 2.8 : Diagram Entitas Kuat dan Lemah

2.2.13.6 *Multiplicity*

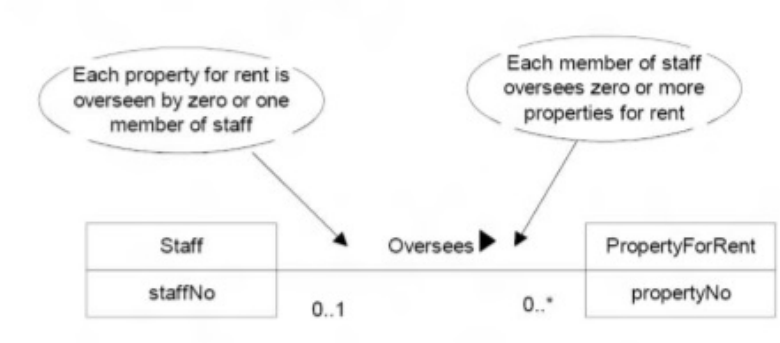
Menurut Connolly dan Begg (2005, p356), *multiplicity* adalah sejumlah kejadian yang mungkin terjadi dari entitas yang berhubungan dengan kejadian tunggal melalui *relationship* tertentu. *Multiplicity* menggambarkan bagaimana entitas saling dihubungkan. Ada tiga jenis *multiplicity* menurut Connolly-Begg (2005,pp357-359) yaitu :

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



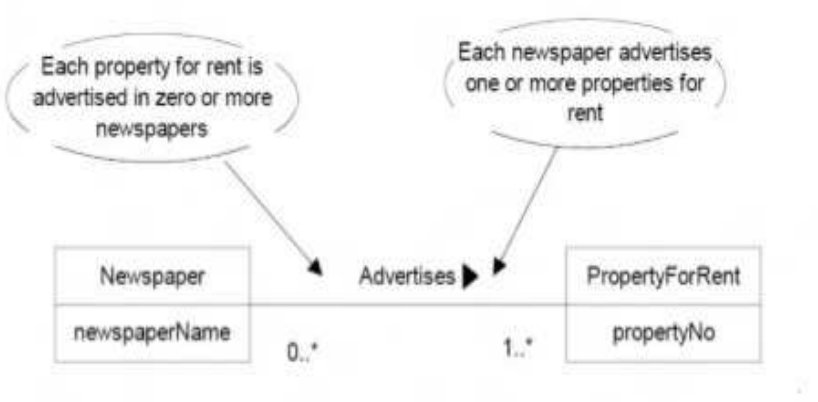
Gambar 2.9 : *Relationship One To One* (1:1)

b. *One To Many* (1:*)



Gambar 2.10 : *Relationship One To Many* (1:*)

c. *Many To Many* (*:*)



Gambar 2.11 :*Relationship Many To Many* (*:*)

2.2.14 Normalisasi

Menurut Connolly dan Begg (2005, p388) normalisasi adalah suatu teknik untuk memproduksi satu set set hubungan dengan kebutuhan yang diinginkan, memberi kebutuhan data dari suatu perusahaan. Tujuan lain dari normalisasi adalah membuat *database* menjadi lebih mudah diakses oleh pengguna dan memberikan kemudahan dalam pemeliharaan data serta mencegah adanya redudansi atau pengulangan data yang nantinya akan menghemat tempat pada penyimpanan Connolly dan Begg (2005, p388). Proses dalam normalisasi menurut Connolly dan Begg terbagi menjadi beberapa tahap, yaitu :

a. Unnormalized Form (UNF)

Suatu tabel dikatakan sebagai bentuk yang *unnormalized* bila didalamnya terdapat kelompok yang berulang atau yang biasa dikenal *repeating group*.

b. Normalisasi Data Pertama (*First Normal Form/1NF*)

Untuk mengubah bentuk *Unnormalized Form* menjadi 1NF, yang harus dilakukan adalah mengidentifikasi dan menghilangkan kelompok berulang agar setiap pertemuan antara baris dan kolom berisi satu dan hanya satu nilai. Langkah-langkah membuat normal pertama (1NF) dari bentuk *unnormal* (UNF) :

1. Menentukan 1 atau lebih atribut sebagai atribut kunci dari tabel *unnormal*.
2. Mengidentifikasi *repeating group* dari suatu tabel *unnormal* yang mengulang atribut kunci di atas.
3. Menghilangkan *repeating group*, dapat dilakukan dengan 2 cara.
4. Mengisi kolom dari baris yang kosong dengan data yang sesuai, atau
5. Menempatkan data yang berulang beserta *key*-nya ke dalam tabel baru.

c. Normalisasi Data Kedua (*Second Normal Form/2NF*)

Pada tahap normalisasi 2NF dihilangkan setiap *Partial Dependence* yang ada pada bentuk 1NF. Yang dimaksud dengan *Partial Dependence* adalah atribut non *primary key* yang merupakan sebagian fungsi dari *primary key*, atau dapat dijelaskan demikian apabila terdapat atribut-atribut dalam suatu relasi yang memiliki ketergantungan fungsional misalnya atribut A dan atribut B, dikatakan *partial dependence* apabila ada sebagian atribut dari A yang dihilangkan namun ketergantungan masih ada. Langkah membuat 2NF dari 1NF :

1. Mengidentifikasi *primary key* dari bentuk 1NF

2. Mengidentifikasi *functional dependency* pada bentuk 1NF

3. Bila terdapat *partial dependency* dalam *primary key* maka tempatkan *primary key* tersebut dalam suatu tabel baru beserta *field-field* yang berkaitan dengannya.

d. Normalisasi Data Ketiga (*Third Normal Form/3NF*)

Pengujian terhadap bentuk normal ketiga dilakukan dengan cara melihat apakah terdapat atribut bukan *key* tergantung fungsional terhadap atribut bukan *key* yang lain (disebut ketergantungan *transitive*). Pada tahap normalisasi 3NF dihilangkan setiap *Transitive Dependence* yang terdapat pada bentuk 2NF. Kondisi *Transitive Dependence* dapat diterangkan sebagai berikut : Misalkan terdapat atribut A,B, dan C yang mempunyai relasi A-B dan B-C. C adalah *Transitive Dependence* terhadap A melalui B.

Transitive Dependence terjadi ketika ada atribut yang bukan merupakan *primary key*, yang memiliki ketergantungan pada atribut lain yang juga bukan merupakan *primary key*. Langkah-langkah membuat 3NF dari 2NF :

1. Mengidentifikasi *primary key* pada bentuk 2NF

2. Mengidentifikasi *functional dependency* dalam tabel tersebut

3. Bila terdapat *transitive dependency* pada *primary key* maka tempatkan *primary key* beserta *field-field* yang berkaitan pada tabel baru.

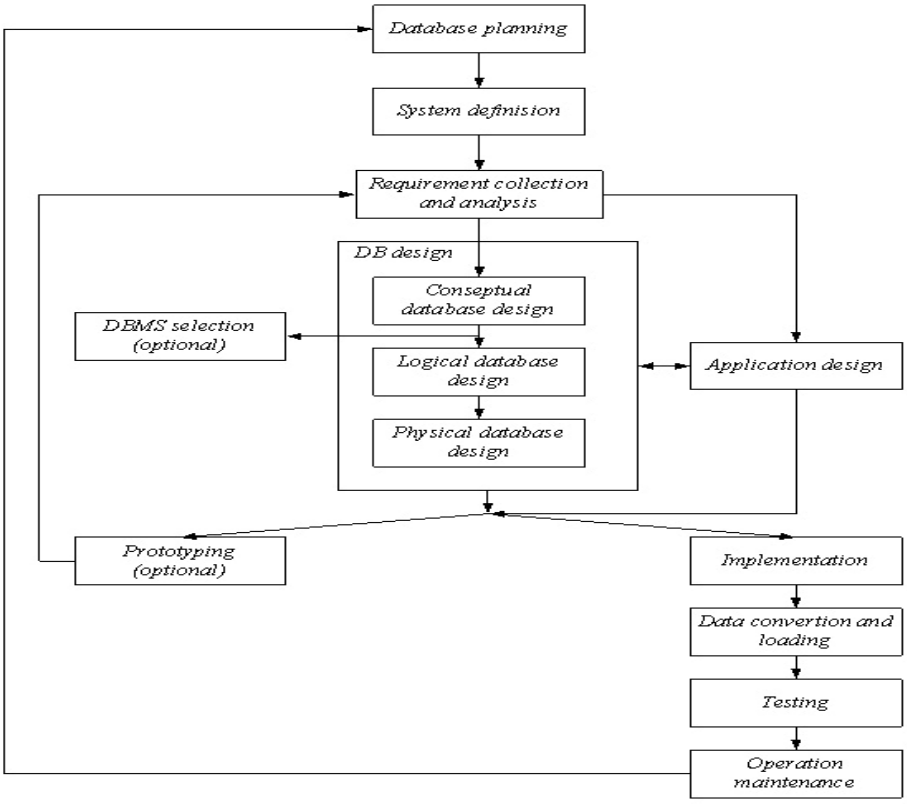
Proses Normalisasi secara umum dilakukan sampai ke tahap 3NF karena sudah tidak terdapat data pengulangan, dan anomali yang ada sudah sangat sedikit.

e. Boyce-Codd Normal Form (BCNF)

Menurut Connolly (2005) *Boyce-Codd Normal Form* adalah sebuah relasi dimana setiap penentu atau determinan adalah *candidate key*. BNCF meupakan perbaikan dari 3NF karena bentuk 3NF masih memiliki kemungkinan mengandung *anomaly*, sehingga masih perlu dinormalisasi lebih lanjut lagi.

2.2.15 Siklus Hidup Aplikasi *Database*

Menurut Connolly dan Begg (2005, pp283-306), siklus hidup aplikasi *database* terdiri dari sepuluh tahapan, yaitu :



Gambar 2.12 : Siklus Hidup Aplikasi *Database*

2.2.15.1 Perencanaan *Database*

Menurut Connolly dan Begg (2005,p285), perencanaan *database* adalah sebuah aktifitas pengaturan yang memungkinkan langkah-langkah dari aplikasi *database* direalisasikan seefektif dan seefesien mungkin.

Hal pertama yang dilakukan adalah mendefinisikan pernyataan sistem bagi proyek *database*. Pernyataan misi mendefinisikan tujuan utama dari aplikasi *database*. Pernyataan misi membantu mengklarifikasi tujuan dari proyek *database* dan menyediakan jalur yang lebih jelas terhadap pembuatan aplikasi *database* yang efisien dan efektif.

Aktifitas selanjutnya mendefinisikan tujuan misi. Setiap tujuan misi harus mengidentifikasikan tugas tertentu yang harus didukung oleh aplikasi *database*. Jika *database* mendukung tujuan misi, pernyataan misi harus disesuaikan. Pernyataan dan tujuan misi harus diselesaikan dengan informasi tambahan yang ditentukan seperti : kegiatan yang harus dilakukan, sumber daya apa yang digunakan, dan biaya yang harus dibayarkan.

2.2.15.2 Definisi Sistem

Menurut Connolly dan Begg (2005, p286), definisi sistem adalah suatu kegiatan yang menjelaskan bidang dan batasan dari aplikasi *database* dan *user view* utama. Batasan dan bidang sistem yang kita buat harus ditentukan tidak hanya untuk pengguna dan area aplikasi saat ini, namun juga untuk pengguna dan area aplikasi di masa depan.

2.2.15.3 Pengumpulan dan Analisa Kebutuhan

Menurut Connolly dan Begg (2005, p288), pengumpulan dan analisa kebutuhan merupakan sebuah proses pengumpulan dan penganalisaan informasi tentang bagian organisasi yang harus didukung oleh aplikasi *database*, dan menggunakan informasi ini untuk mengidentifikasi kebutuhan pengguna bagi sistem yang baru. Informasi yang didapatkan dari setiap *user view* utama (peran kerja atau area aplikasi perusahaan) adalah deskripsi data yang digunakan atau dibuat. Penjelasan bagaimana data digunakan atau dibuat. Berbagai kebutuhan tambahan bagi aplikasi *database* yang baru. Dalam tahapan ini, jumlah data yang diperoleh tergantung masalah dan kebijakan perusahaan. Informasi yang terkumpul pada tahapan ini mungkin tidak terstuktur dan melibatkan beberapa permintaan informal yang harus diubah ke dalam kebutuhan yang lebih terstruktur. Terdapat dua pendekatan dalam tahapan ini yaitu pendekatan terpusat dan pendekatan *view integration*.

2.2.15.4 Perancangan *Database*

Menurut Connolly dan Begg (2005, p291), perancangan *database* yang akan mendukung operasi dan tujuan perusahaan.

Menurut Connolly dan Begg (2005, pp293-295), tahap-tahap perancangan *database* dibagi menjadi tiga bagian, yang mana masing-masing bagian terdapat beberapa tahap, yaitu :

2.2.15.4.1 Perancangan *Database* Konseptual

Menurut Connolly dan Begg (2005, p293), perancangan *database* konseptual adalah sebuah proses pembuatan model dari informasi yang digunakan dalam pembuatan model dari informasi yang digunakan dalam perusahaan, yang terbebas dari semua pertimbangan fisikal seperti DBMS target, program aplikasi, bahasa pemrograman, *hardware*, dan sebagainya. Setiap model data konseptual lokal terdiri dari :

- Entitas
- *Relationship*
- Atribut dan *domain* atribut
- *Primary Key*
- *Candidate Key*
- Batasan Integral

Didalam langkah ini melibatkan beberapa aktifitas, yaitu :

a. Mengidentifikasi Entitas

   Bertujuan untuk mengidentifikasi entitas utama yang dibutuhkan oleh *view*. Sebuah metode untuk mengidentifikasi entitas adalah dengan memeriksa spesifikasi kebutuhan pengguna. Dari spesifikasi ini, kita mengidentifikasi kata benda ata frase benda. Kita juga mencari objek utama.

b. Mengidentifikasi *Relationship*

Bertujuan untuk mengidentifikasi *relationship* penting yang terdapat antara entitas yang telah diidentifikasi. Secara khusus, *relationship* ditandai dengan kata kerja atau verbal.

c. Mengidentifikasi dan Menghubungkan Atribut Dengan Entitas atau *Relationship*

Bertujuan untuk menghubungkan atribut dengan entitas atau *relationship* yang sesuai. Atribut dapat diidentifikasi dimana kata benda merupakan sebuah sifat, kualitas, *identifier*, atau karakteristik dari entitas atau *relationship*.

d. Menentukan *Domain* Atribut

Bertujuan untuk menentukan *domain* bagi atribut didalam model data konseptual lokal. *Domain* merupakan sebuah kolom nilai dari satu atau lebih atribut yang menggambarkan nilai mereka. Pengembangan model data yang utuh menentukan *domain* bagi setiap atribut termasuk :

- Sekelompok nilai yang diperbolehkan bagi atribut
- Ukuran dan format atribut

e. Menentukan Atribut *Primary Key* dan *Candidate Key*

Bertujuan untuk mengidentifikasi *candidate key* bagi setiap entitas dan jika terdapat lebih dari satu *candidate key*, pilihlah satu untuk dijadikan *primary key*.

f. Mempertimbangkan Penggunaan Konsep Pemodelan Lebih Lanjut

- Memeriksa Model Untuk Pengulangan

Bertujuan untuk memeriksa keberadaan berbagai pengulangan didalam model. Terdapat dua aktifitas didalam tahap ini, yaitu :

- Memeriksa kembali *relationship one-to-one*
- Menghilangkan *relationship* yang berulang
- Memvalidasi Model Konseptual Lokal Terhadap Transaksi *User*

Bertujuan untuk memastikan bahwa model konseptual lokal mendukung transaksi yang diperlukan oleh *view*. Dengan menggunakan model tersebut, kita berusaha untuk untuk membentuk operasi manual. Jika kita dapat memecahkan semua transaksi dalam cara ini, kita telah memastikan bahwa model data konseptual mendukung transaksi yang diperlukan. Ada dua pendekatan untuk memastikan bahwa model data konseptual lokal mendukung transaksi yang diperlukan, yaitu :

- Menjelaskan transaksi

- Menggunakan *transaction pathway*

- Me*review* Model Data Konseptual Lokal Dengan *User*

Bertujuan untuk me*review* model data konseptual lokal dengan pengguna untuk memastikan bahwa model tersebut merupakan gambaran yang sebenernya dari *view*.

2.2.15.4.2 Perancangan *Database* Logikal

Menurut Connolly dan Begg (2005, p294), perancangan *database* logikal merupakan sebuah proses pembuatan model dari informasi yang digunakan dalam

sebuah perusahaan berdasarkan pada model data tertentu, tetapi terbebas dari DBMS tertentu dan pertimbangan fisikal yang lainnya.

a. Menghilangkan fitur-fitur yang tidak sesuai dengan model *relational* (*optional*).

   Langkah-langkah yang dilakukan antara lain :

- Hilangkan tipe relasi *binary many-to-many*
- Hilangkan tipe relasi yang kompleks
- Hilangkan atribut *multi-valued*

b. Membuat relasi untuk model data logikal lokal

   Tujuan dari langkah ini adalah untuk membuat suatu relasi untuk model data lokal logikal yang merepresentasikan suatu *entity*, relasi, dan juga atribut yang telah diidentifikasi.

c. Memvalidasi relasi menggunakan normalisasi

   Proses normalisasi melibatkan beberapa langkah, yaitu :

- *First normal form* (1NF), menghilangkan *repeating group*
- *Second Normal Form* (2NF), menghilangkan *partial dependencies* pada *primary key*
- *Third Normal Form* (3NF), menghilangkan *transitive dependencies* pada *primary key*

d. Memvalidasi relasi pada transaksi-transaksi *user*

Langkah ini dilakukan dengan tujuan untuk memastikan bahwa relasi dalam model data logikal lokal mendukung transaksi-transaksi yang diperlukan dalam penggambaran (*view*).

e. Mendefinisikan *integrity constraints*

Ada lima tipe *integrity constraints*, antara lain :

- *Required Data*

  Beberapa atribut harus selalu berisi data yang sah sehingga atribut tersebut tidak diperbolehkan menerima *null*.

- *Attribute Domain Constraints*

  Setiap atribut mempunyai *domain* yang merupakan sekumpulan nilai yang sah.

- *Entity Integrity*

  Primary key dari sebuah *entity* tidak dapat menerima *null*.

- *Referential Integrity*

  Jika *foreign key* berisi nilai maka nilai tersebut harus menunjuk pada *tuple* yang ada pada relasi induk. Untuk meyakinkan *referential integrity* perlu dispesifikasikan *existence constraints* yang mendefinisikan kondisi dimana *candidate key* atau *foreign key* ditambahkan, diubah, atau dihapus. Jika sebuah *tuple* dari relasi induk dihapus, *referential integrity* hilang jika ada *tuple* anak menunjuk ke *tuple* induk yang dihapus.

f. Meninjau ulang model data logikal lokal dengan *user*

Langkah ini dilakukan dengan tujuan untuk memastikan model data logikal lokal dan dokumentasi pendukung menggambarkan model yang merupakan representasi nyata, dari *view* tersebut.

2.2.15.4.3 Perancangan *Database* Fisikal

Menurut Connolly dan Begg (2005, p294), perancangan *database* fisikal adalah sebuah proses pembuatan deskripsi implementasi *database* pada tempat penyimpanan kedua. Perancangan ini menjelaskan relasi dasar, organisasi *file*, dan *index* yang digunakan untuk mencapai akses yang efisien ke data dan berbagai batasan integritas yang berhubungan serta penilaian keamanan.

a. Merancang relasi dasar

Tujuan dari langkah ini adalah untuk memutuskan bagaimana merepresentasikan relasi dasar yang diidentifikasi dalam model data logikal global pada DBMS yang dipakai.

b. Merancang batasan perusahaan

Tujuan dari langkah ini adalah untuk merancang batasan perusahaan untuk DBMS yang dipakai.

c. Analisis transaksi

Tujuan dari langkah ini adalah untuk mengerti fungsi dari suatu transaksi yang mana akan dijalankan pada basis data dan untuk menganalisa transaksi yang penting

untuk membuat perancangan sistem basis data fisikal efektif maka perlu dimiliki pengetahuan mengenai transaksi *Query* yang akan dijalankan didalam basis data. Hal ini termasuk kuantitatif atau kualitatif informasi.

d. Memilih indeks

Tujuan dari langkah ini adalah untuk menentukan apakah penambahan indeks akan meningkatkan kinerja dari suatu sistem. Biasanya pemilihan atribut untuk indeks dilakukan dengan cara :

- Suatu atribut yang digunakan paling sering untuk operasi penggabungan, yang akan membuat penggabungan tersebut lebih efisien
- Suatu atribut yang digunakan paling banyak untuk mengakses suatu *record* didalam relasi yang ada

e. Memperkirakan kapasitas *disk* yang dibutuhkan untuk menyimpan basis data

Tujuan dari langkah ini adalah untuk mengestimasi ukutan kapasitas *disk* yang diperlukan untuk sistem *database*.

f. Merancang *views*

Tujuan dari langkah ini adalah untuk merancang tampilan *user* yang diidentifikasi selama pengumpulan informasi san analisa dari siklus hidup aplikasi sistem *database*.

g. Merancang mekanisme keamanan

Tujuan dari langkah ini adalah untuk merancang ukuran keamanan untuk basis data yang telah dispesifikasikan oleh *user*.

2.2.15.5 Pemilihan DBMS

Menurut Connolly dan Begg (2005, p295), pemilihan DBMS merupakan pemilihan sebuah DBMS yang sesuai untuk mendukung aplikasi *database*. Pemilihan DBMS sangat diperlukan ketika sebuah perusahaan ingin mengembangkan atau mengganti sistem yang sudah ada, proses ini digunakan untuk mengevaluasi produk DBMS. Tujuan dari pemilihan DBMS adalah untuk memilih sebuah sistem yang sesuai dengan kebutuhan saat ini dan yang akan datang pada perusahaan, menyeimbangkan biaya pembelian produk DBMS, berbagai *software* atau *hardware* tambahan yang diperlukan untuk mendukung sistem *database*, dan biaya yang berhubungan dengan perubahan dan pelatihan staf. Berikut ini adalah langkah-langkah untuk memilih DBMS yang baik :

- Menjelaskan istilah referensi pembelajaran

- Mendaftarkan dua atau tiga produk

- Mengevaluasi produk

- Merekomendasikan pilihan dan membuat laporan

2.2.15.6 Perancangan Aplikasi

Menurut Connolly dan Begg (2005, p299), rancangan aplikasi yaitu perancangan antarmuka pengguna dan program aplikasi yang digunakan dan memproses *database*. Aktifitas antara *database design* dengan *application design* terjadi secara paralel. Dalam banyak hal, tidak mungkin untuk menyelesaikan perancangan aplikasi sampai perancangan dari *database* itu sendiri. Dalam hal ini, *database* hadir untuk mendukung aplikasi dan harus ada aliran informasi antara *application design* dan *database design*. Seluruh fungsi-fungsi yang tercantum dalam spesifikasi kebutuhan pengguna harus ada dalam perancangan aplikasi, untuk aplikasi *database* yang meliputi perancangan program aplikasi yang mengakses *database* dan melakukan transaksi. Perancangan *user interface* yang tepat ke dalam aplikasi *database* menjadi kebutuhan tambahan agar fungsi yang dibutuhkan tercapai. Perancangan *user interface* terkadan tidak diperhatikan atau ditinggalkan selama tahapan perancagan. Program aplikasi yang mudah dipelajari, sederhana dalam penggunaan, maka pengguna cenderung untuk dapat memanfaatkan dengan baik. Hal ini menunjukkan bahwa *user interface* merupakan salah satu komponen penting dari sistem. Terdapat dua aspek penting dalam perancangan aplikasi, yaitu :

2.2.15.6.1 Perancangan Transaksi

Menurut Connolly dan Begg (2005, p288), transaksi merupakan tindakan yang dilakukan oleh pengguna tunggal atau program aplikasi yang mengakses atau mengubah isi *database*. Transaksi menggambarkan kejadian dunia nyata seperti pendaftaran, penambahan anggota baru, penambahan pembeli baru, dsb. Transaksi digunakan untuk

database untuk memastikan bahwa data yang terdapat dalam *database* sesuai dengan situasi dunia nyata dan mendukung kebutuhan informasi pengguna. Tujuan dari perancangan transaksi untuk mendefinisikan dan medokumentasikan karakteristik transaksi tingkat tinggi yang diperlukan pada *database*, yaitu :

- Data yang digunakan oleh transaksi

- Karakteristik fungsional dari transaksi

- Hasil transaksi

- Kepentingan terhadap pengguna

- Nilai harapan pengguna

Terdapat tiga jenis utama transaksi, yaitu :

- *Retrieval Transaction :* Digunakan untuk mengambil data untuk ditampilkan pada layar atau didalam hasil laporan

- *Update Transaction :* Digunakan untuk memasukkan *record* baru, menghapus *record* lama, atau memodifikasi *record* yang ada didalam *database*.

- *Mixed Transaction :* Melibatkan pengambilan dan peng-*update*-an data

2.2.15.6.2 Perancangan *User Interface*

Menurut Connolly dan Begg (2005, p289), terdapat beberapa langkah dalam membuat rancangan antarmuka yang baik bagi aplikasi *database*, yaitu :

- Judul yang berarti

- Rancangan permintaan secara visual dari laporan

- Label *field* yang dikenal

- Singkatan dan istilah yang konsisten

- Penggunaan warna yang konsisten

- Batasan dan ruang yang terlihat bagi *field data entry*

- Pergerakan kursor yang baik

- Pesan kesalahan bagi nilai yang tak sesuai

- Penandaan *field* opsional yang jelas

- Pesan penjelasan bagi *field*

- Sinyal penyelesaian

2.2.15.7 *Prototyping*

Menurut Connolly dan Begg (2005, p304), *prototyping* merupakan aktifitas membangun suatu model kerja dari sebuah sistem *database*. *Prototype* adalah sebuah model kerja yang tidak secara normal memiliki semua fitur yang dibutuhkan atau menyediakan semua fungsi akhir dari sistem. Tujuan utama dari pembangunan sebuah *prototype database system* adalah untuk memungkinkan pengguna menggunakan *prototype* untuk mengidentifikasi fitur sistem yang bekerja dengan baik atau tidak, dan

jika mungkin untuk memberikan saran perbaikan atau bahkan fitur baru untuk sistem *database*.

2.2.15.8 Implementasi

Menurut Connolly dan Begg (2005, p304), implementasi adalah realisasi fisik dari rancangan aplikasi dan *database*. Implementasi *database* dicapai dengan menggunakan DDL dari DBMS yang dipilih atau GUI yang menyediakan fungsi yang sama selama penyembunyian *statement* DDL tingkat rendah. DDL digunakan untuk membuat struktur *database* dan *file database* kosong. Program aplikasi diterapkan menggunakan bahasa tingkat tiga atau empat. Bagian dari program aplikasi ini adalah transaksi *database*, yang diterapkan menggunakan DML yang dapat dijalankan pada sekumpulan bahasa pemrograman, seperti *Visual Basic, Delphi, C, C++, Java,* atau *Pascal*. Digunakan juga komponen lain dari perancangan aplikasi seperti menu layar, *form* masukkan data dan laporan-laporan. Keamanan dan integritas dalam aplikasi juga diterapkan.

2.2.15.9 Pengubahan dan *Load* Data

Menurut Connolly dan Begg (2005, p305), pengubahan dan *load* data adalah suatu proses transfer berbagai data yang ada kedalam *database* yang baru dan pengubahan berbagai aplikasi yang ada untuk berjalan pada *database* yang baru. Tahapan ini dibutuhkan hanya ketika sistem *database* yang baru menggantikan sistem yang lama. Sekarang ini, sudah menjadi hal yang biasa bagi sebuah DBMS untuk mempunyai utilitas yang dapat memuat keseluruhan *file* yang ada kedalam *database*

yang baru. Utilitas biasanya membutuhkan spesifikasi dari sumber dan tujuannya, sehingga mengubah data sesuai dengan format *database* yang baru.

2.2.15.10 Pengujian

Menurut Connolly dan Begg (2005, p305), pengujian adalah sebuah proses pengeksekusian program aplikasi dengan maksud menemukan kesalahan. Hal ini dicapai dengan strategi pengujian terencana yang hati-hati dan data yang nyata sehingga proses pengujian seluruhnya ditangani dengan metodologis dan benar. Jika proses pengujian berjalan dengan baik, hal ini akan menemukan banyak kesalahan dalam program aplikasi dan struktur *database*. Pengujian juga menunjukkan agar aplikasi *database* bekerja sesuai dengan spesifikasi dan kebutuhan performa yang diinginkan. Hasil dari pengujian dapat memberikan penilaian terhadap kehandalan dan kualitas *software*.

2.2.15.11 Pemeliharaan Operasional

Menurut Connolly dan Begg (2005, p293), pemeliharaan operasional adalah sebuah proses pemantauan dan pemeliharaan sistem berikut instalasi. Dalam tahap ini melibatkan dua aktifitas, yaitu :

- Pemantauan kinerja sistem. Jika kinerja berada jauh dibawah level yang diharapkan, diperlukan perbaikan atau penyusunan kembali *database*.
- Pemeliharaan dan *upgrade* aplikasi *database*. Kebutuhan baru dimasukkan ke dalam aplikasi *database* melalui tahap-tahap siklus hidup aplikasi *database* yang sebelumnya.

2.2.16 Data Flow Diagram

Menurut Hall (2001 , p69), DFD yang juga disebut dengan Diagram Arus Data, adalah diagram yang menyajikan rangkaian simbol - simbol untuk mencerminkan proses, sumber-sumber data, arus data, dan entitas dalam sebuah sistem pada tingkatan rinci yang berbeda.

DFD pertama kali diperkenalkan sebagai *modeling tools* Tom Demarco (1978), Gane dan Sarson (1979) dengan menggunakan pendekatan metode analisis sistem terstrukstur (*structured system analysis method*). DFD juga dapat digunakan untuk merepresentasikan suatu sistem yang otomatis maupun manual dengan menggunakan gambar yang berbentuk jaringan grafik.

Tingkatan Diagram Pada DFD :

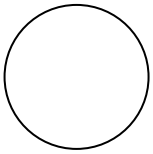
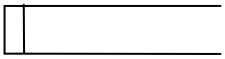
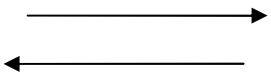
1. Diagram Konteks

- Merupakan level tertinggi pada DFD yang menggambarkan seluruh output atau input ke sistem
- Memberikan gambaran tentang keseluruhan sistem
- Terminal yang memberikan masukan kepada sistem disebut *source*, sedangkan yang menerima keluaran dari sistem disebut *sink*
- Hanya ada satu proses didalam diagram konteks
- Tidak boleh ada proses penyimpanan data (*data store*)

2. Diagram Nol

- Memperlihatkan proses penyimpanan data (data store) yang digunakan
- Untuk proses yang tidak rinci lagi pada level selanjutnya, tambahkan tanda * pada akhir nomor proses
- Keseimbangan antara input dan output pada diagram nol dan diagram konteks harus terpelihara

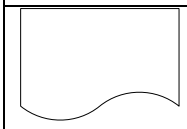
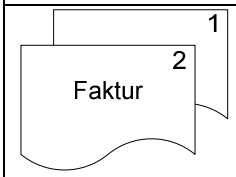
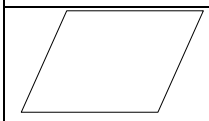
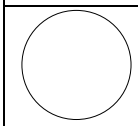
Simbol-simbol dalam DFD adalah :

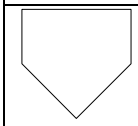
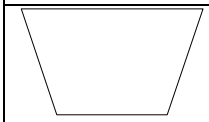
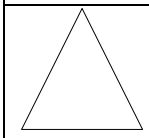
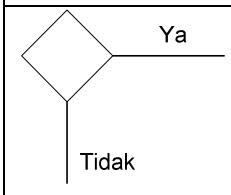
	External entity atau terminal
	Proses
	Penyimpanan data atau data storage
	Aliran data atau Data Flow

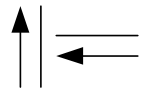
Tabel 2.13 Tabel Simbol DFD

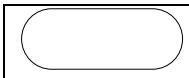
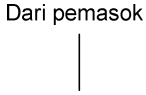
2.2.17 Bagan Alir Dokumen (Document FlowChart)

Document FlowChart digunakan untuk menggambarkan bagan alir dokumen suatu sistem. Berikut adalah simbol – simbol standar beserta keterangannya untuk mengambarka n bagan alir dokumen :

Lambang	Keterangan
	**Dokumen**. Simbol ini digunakan untuk menggambarkan semua jenis dokumen, yang merupakan formulir yang diunakan untuk merekam data terjadinya suatu transaksi
	**Dokumen dan tembusannya**. Simbol ini digunakan untuk menggambarkan dokumen asli dan tembusannya. Nomor lembar dokumen dicantumkan di sudut kanan atas
	**Catatan**. Simbol ini digunakan untuk menggambarkan catatan akuntansi yang digunakan untuk mencatat data yang direkam sebelumnya di dalam dokumen atau formulir
	**Penghubung pada halaman yang sama (*on-page connector*)**. Dalam menggambarkan bagan alir, arus dokumen dibuat mengalir dari atas ke bawah dan dari kiri ke kanan. Karena keterbatasan ruang halaman kertas untuk

	menggambar, maka diperlukan simbol penghubung ini.
	**Penghubung pada halaman yang berbeda (*off-page connector*).** Jika untuk menggambarkan bagan alir suatu sistem akuntansi diperlukan lebih dari satu halaman, simbol ini harus digunakan untuk menunjukkan kemana dan bagaimana bagan alir terkait satu dengan lainnya.
	**Kegiatan *manual***. Simbol ini digunakan untuk menggambarkan kegiatan manual seperti : menerima order dari pembeli, mengisi formulir dan kegiatan lainnya
	**Keterangan, komentar.** Simbol ini memungkinkan ahli sistem menambahkan keterangan untuk memperjelas pesan yang disampaikan dalam bagan alir
	**Arsip permanen**. Simbol ini digunakan untuk menggambarkan arsip permanen yang merupakan tempat penyimpanan dokumen yang tidak akan diproses lagi dalam sistem akuntansi yang bersangkutan
	**Keputusan**. Simbol ini menggambarkan keputusan yang harus dibuat dalam proses pengolahan data. Keputusan yang dibuat ditulis di dalam simbol

	**Garis alir (*flowline*)**. Simbol ini menggambarkan arah proses pengolahan data. Anak panah tidak digambarkan jika arus dokumen mengarah ke bawah dan ke kanan. Jika arus dokumen mengalir ke atas atau ke kiri, anak panah perlu dicantumkan
	**Persimpangan garis alir**. Jika dua garis alir bersimpangan, untuk menunjukkan arah masing – masing garis, salah satu garis dibuat sedikit melengkung tepat pada persimpangan ke dua garis tersebut
	**Pertemuan garis alir**. Simbol ini digunakan jika dua garis alir bertemu dan salah satu garis mengikuti arus garis lainnya

	**Mulai/berakhir** (***terminal***)**.** Simbol ini untuk menggambarkan awal dan akhir suatu sistem akuntansi
	**Masuk ke sistem**. Karena kegiatan di luar sistem tidak perlu digambarkan dalam bagan alir, maka diperlukan simbol untuk menggambarkan masuk ke sistem yang digambarkan dalam bagan alir
	**Keluar ke sistem lain**. Karena kegiatan di luar sistem tidak perlu digambarkan dalam bagan alir, maka diperlukan simbol untuk menggambarkan keluar ke sistem lain

Tabel 2.14 Tabel Simbol Document FlowChart

2.3 Teori Pendukung

2.3.1 Reservasi/Pemesanan

Kegiatan yang dilakukan sebelum tamu datang ke hotel, hal ini dilakukan supaya tamu mendapat jaminan akan memperoleh tempat/kamar yang diinginkan saat tiba ke hotel

2.3.2 Bill Tamu

Bill yang harus dibayar untuk sewa kamar & hidangan di hotel yang diselenggarakan dalam front office sampai dari tamu *check in* sampai tamu tersebut melakukan *check out*.

2.3.3 Hotel

Tempat penginapan yang ditangani secara komersial, dengan menyediakan fasilitas akomodasi yang memadai beserta hidangan dan pelayanan yang baik.

2.3.4 Pengenalan *Netbeans*

NetBeans mengacu pada dua hal, yakni platform untuk pengembangan aplikasi desktop java, dan sebuah Integrated Develoment Environtment (IDE) yang dibangun menggunakan platform NetBeans. Platform NetBeans meungkinkan aplikasi dibangun dari sekumpulan komponen perangkat lunak moduler yang disebut ‘modul’. Sebuah modul adalah suatu arsip Java (Java archive) yang memuat kelas-kelas java untuk berinteraksi dengan NetBeans Open API dan file manifestasi yang mengidentifikasinya sebagai modul.

Aplikasi yang dibangun dengan modul-modul dapat dikembangkan dengan menambahkan modul-modul baru. Karena modul dapat dikembangkan secara independen, aplikasi berbasis platform NetBeans dapat dengan mudah dikembangkan oleh pihak ketiga secara mudah dan powerful.

2.3.5 *Microsoft SQL Server 2005*

Menurut Widodo Budiharto (2006), *Microsft SQL Server 2005* adalah perangkat lunak *Relational Database Management System* (RDBMS) yang handal. Didesain untuk mendukung proses transaksi yang besar (seperti order entry yang online, inventory, akuntansi atau manufaktur). SQL Server 2005dapat dijalankan pada Windows 2000 Profesional Service Pack 4, Windows 2000 Server Service Pack 4, Windows XP Profesional Service Pack 2 atau Microsoft 2003 Service Pack 1. SQL server 2005 membutuhkan installer Windows 3.1 yang dapat diperoleh pada saat instalasi Visual Studio 2005. SQL Server 2005 memiliki fasilitas tambahan yang menyebabkannya memiliki kemampuan penuh dalam e-commerce.

2.3.5.1 Komponen *Microsoft SQL Server 2005*

Komponen dari Microsoft SQL Server 2005 adalah sebagai berikut:

1. *SQL Server Database Engine*

   Database engine adalah layanan inti untuk menyimpan, memproses, serta mengamankan data. Microsoft SQL Server 2005 mendukung sampai 50 database engine dalam satu computer.

2. *SQL Server Analysis Service*

   SQL Server analysis service menyediakan layanan yang diperlukan untuk menggunakan data warehouse. Layanan ini juga disebut OLAP

3. *SQL Server Reporting Service*

   Merupakan sebuah platform untuk membangun dan men-deploy aplikasi yang mengenerate dan mengirim notifikasi

4. *SQL Server Integration Service*

   Merupakan platform untuk membangun suatu solusi data terintegrasi dengan performa yang tinggi, termasuk ekstraksi, transformasi dan loading untuk paket data warehouse.

Management *Tool* dari *SQL Server 2005* salah satunya adalah *SQL Server Management Studio*, yang merupakan suatu aplikasi lengkap untuk melakukan pekerjaan manajemen terhadapseluruh basis data yang ada dalam *SQL Server database engine. SQL Server Management Studio* menyediakan fitur untuk melakukan query , melihat status, melihat dan menambah constraint, trigger, index dan sebagainya.