BAB II

LANDASAN TEORI

Bab ini menguraikan teori-teori yang digunakan dalam pembuatan tugas akhir dengan judul penerapan representasi logika fuzzy untuk structured query language pada sistem basis data yang antara lain meliputi :

2.1. Sistem Basis Data (*Database*)

Sistem basis data atau yang lebih sering disebut *database* (Basis Data) terdiri dari dua kata, yaitu Basis dan Data. Basis dapat diartikan sebagai markas atau gudang, tempat bersarang / berkumpul. Sedangkan data adalah representasi fakta dunia nyata yang mewakili suatu objek. *Database* (Basis Data) dapat didefinisikan dari sejumlah sudut pandang , yaitu (1) Merupakan himpunan kelompok data (arsip) yang saling berhubungan yang diorganisir sedemikian rupa agar kelak dapat dimanfaatkan kembali dengan cepat dan mudah, (2) Merupakan kumpulan data yang saling berhubungan yang disimpan secara bersama sedemikian rupa dan tanpa pengulangan (redudansi) yang tidak perlu, untuk memenuhi berbagai kebutuhan, dan (3) Merupakan kumpulan *file* / tabel / arsip yang saling berhubungan yang disimpan dalam media penyimpanan elektronis (Fathansyah, 1999).

2.2. DBMS (Database Management System)

DBMS (*Database Management System*) merupakan sistem *software* yang multiguna, yang menyediakan fasilitas untuk :

1. Mendefinisikan yang artinya melibatkan spesifikasi : (a) tipe data, (b) struktur, dan (c) kendala / *constraint* dari data yang di simpan.
2. Membangun yang artinya berkaitan dengan proses penyimpanan data itu sendiri pada suatu media penyimpanan yang dikontrol oleh DBMS.
3. Memanipulasi yang artinya termasuk didalamnya fungsi-fungsi sebagai "*query*" terhadap basis data, misal : *retrieve*, *update*, *generate report*.

dimana fungsi-fungsi tersebut digunakan sistem basis data untuk aplikasi yang beraneka ragam.

2.3. Structured Query Language (SQL)

Pada awal tahun 1970, D.D Chamberlain, M. M. Astrahan dan lainnya mengembangkan metode formal dalam memanipulasi dan mendefinisikan database relasional yaitu bahasa SEQUEL (*Structured English Query Language*). Bahasa ini menyediakan metode yang luas dalam menggunakan database relasional sedemikian sehingga mengubah teori relasional menjadi sesuatu yang dapat diimplementasikan secara praktis. Contoh perintah SEQUEL untuk mengambil manager project dari departemen engineering akan tampak seperti berikut :

```
SELECT PROMNGR
FROM PROJECT
WHERE DEPARTMENT = "ENGINEERING"
```

Bahasa yang kemudian disingkat menjadi SQL (*Structured Query Language*) ini kemudian menjadi dasar (standart) dari sistem *database* relasional.

2.4. Fuzzy Logic

Untuk menghitung gradasi yang tidak terbatas jumlahnya antara benar dan salah, Zadeh mengembangkan ide penggolongan himpunan (*set*) yang ia namakan himpunan *fuzzy* (*fuzzy set*). Tidak seperti logika *boolean* yang menyatakan bahwa suatu pernyataan adalah benar atau salah, *fuzzy logic* dapat membaginya dalam derajat keanggotaan dan derajat kebenaran sehingga suatu pernyataan dapat menjadi sebagian benar dan sebagian salah pada waktu yang sama.

2.4.1. Konsep Utama Fuzzy

A. Prinsip ketidakpastian

Beberapa ilmu matematika terkadang sulit untuk dipastikan, seperti teori probabilitas. Hal ini bisa diklasifikasikan berdasar tipe ketidakpastian yang dilakukan. Ada beberapa tipe ketidakpastian, dua diantaranya adalah *Stochastic Uncertainty dan Lexical Uncertainty*.

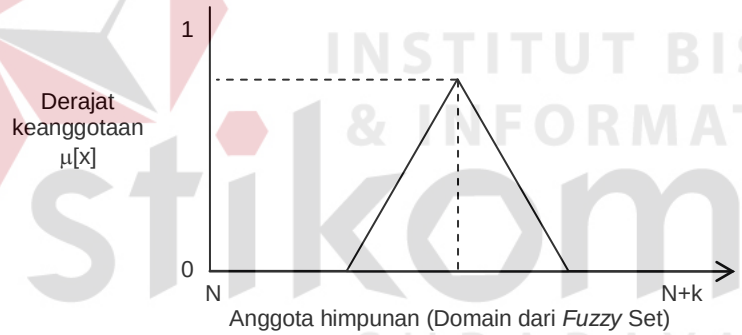
Stochastic Uncertainty berhubungan dengan arah ketidakpastian dari kejadian yang pasti. Sedangkan *Lexical Uncertainty* merupakan ketidakpastian yang diungkapkan oleh kata-kata manusia, seperti “orang yang tinggi”, “hari yang panas” dan sebagainya.

B. Variabel linguistik

Fuzzy pada dasarnya menitikberatkan pada pengukuran dan penalaran tentang kekaburan atau bentuk *fuzzy* yang nampak dalam bahasa alami. Dalam *fuzzy* bentuk *fuzzy* dinyatakan sebagai variabel linguistik (disebut juga variabel *fuzzy*).

Variabel linguistik adalah bentuk yang digunakan dalam bahasa alami untuk menggambarkan beberapa konsep yang biasanya mempunyai kekaburan atau nilai *fuzzy*. Sebagai contoh dalam pernyataan "Jack adalah muda" menyatakan bahwa variabel linguistik umur mempunyai nilai linguistik muda. Seperti halnya variabel aljabar yang berisi angka sebagai nilainya maka vaiabel linguistik menggunakan kata dan kalimat sebagai nilainya. Misalnya: jika T variabel linguistik yang berisi himpunan umur, maka isi T yang juga merupakan himpunan *fuzzy* adalah: *T = {sangat tua, tua , setengah baya, agak muda, muda, sangat muda}*.

C. Himpunan fuzzy (*Fuzzy Set*)



Gambar 2.1 Himpunan Fuzzy (fuzzy sets)

Gambar 2.1 menjelaskan tentang himpunan *fuzzy* yang terdiri atas 3 bagian, dimana sumbu horisontal menunjukkan himpunan anggota, sumbu vertikal menunjukkan derajat dari keanggotaan, dan garis yang menghubungkan masing-masing titik dari anggota dengan derajat keanggotaan yang tepat.

Teori himpunan tradisional menggambarkan dunia sebagai hitam dan putih. Ini berarti sebuah obyek berada didalam atau diluar himpunan yang diberikan. Dalam teori himpunan tradisional untuk anggota diberi nilai 1 dan

untuk bukan anggota diberi nilai 0; ini disebut himpunan *crisp*. Sebagai contoh anggota himpunan orang muda dapat berisi hanya orang yang berumur kurang dari 10. Penggunaan interpretasi ini pada seseorang yang berulang tahun ke-11, maka orang tersebut bukan anggota himpunan orang muda.

Himpunan *fuzzy* memberikan nilai keanggotaan antara 0 dan 1 yang menggambarkan secara lebih alami sebuah kumpulan anggota dengan himpunan, Sebagai contoh, jika seorang berumur 5 tahun dapat diberikan nilai keanggotaan 0.9 atau jika umurnya 13 tahun nilai keanggotaannya 0.1. Dalam contoh ini "umur" adalah variabel linguistik dan "muda" adalah salah satu himpunan *fuzzy*.

Himpunan *fuzzy* dapat didefinisikan sebagai berikut : misalkan X semesta pembicaraan, dengan elemen dari X dinotasikan x. Sebuah himpunan *fuzzy* A dari X dikarakteristikkan dengan fungsi keanggotaan $\mu \wedge (x) : X \rightarrow [0,1]$. Pada *fuzzy* , kejadian atau elemen x diberikan nilai keanggotaan dengan fungsi keanggotaan μ. Nilai ini mempresentasikan derajat keanggotaan elemen x pada himpunan *fuzzy* A. $\mu_{\wedge}$ (x) = Degree($x \in A$) nilai keanggotaan dari x berada pada interval : $0 \leq \mu_{\wedge}(x) \leq 1$

Himpunan *fuzzy* adalah perluasan dari teori himpunan tradisional. Himpunan *fuzzy* menyamakan konsep keanggotaan dengan menggunakan fungsi keanggotaan μ yang menghasilkan nilai antara 0 dan 1 yang mempresentasikan derajat keanggotaan obyek x pada himpunan A.

Untuk mempresentasikan himpunan *fuzzy* dalam komputer perlu didefinisikan fungsi keanggotaannya. Sebagai contoh : orang tinggi. Dapat dinyatakan pada setiap individu, pada tingkatan mana bahwa mereka yakin seseorang itu dikatakan tinggi. Setelah mengumpulkan jawaban untuk interval ukuran tinggi, dapat disajikan tingkat rata-rata untuk menghasilkan suatu

himpunan *fuzzy* dari orang-orang yang tinggi. Fungsi ini dapat digunakan sebagai suatu keyakinan (nilai keanggotaan). Bagi individu yang menjadi anggota himpunan *fuzzy* dari orang tinggi.

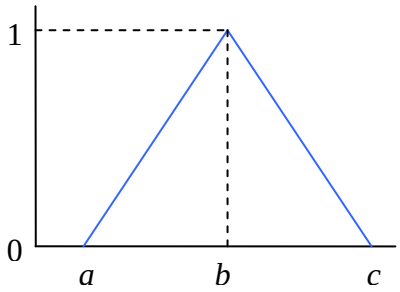
Dengan membentuk *fuzzy* subset untuk berbagai bentuk *fuzzy*, dianggap nilai keanggotaan dari obyek yang diberikan pada setiap himpunan. Pendekatan lain yang sering ditemukan pada praktek untuk membentuk himpunan *fuzzy* sangat berhubungan dengan interpretasi dari seorang ahli. Seperti teknik pengumpulan data, dapat ditanyakan pada pakar untuk kepercayaannya bahwa berbagai obyek merupakan bagian himpunan yang diberikan.

D. Fungsi keanggotaan (*Membership Function*)

Derajat dimana angka teknis bernilai sesuai konsep bahasa dari kondisi variabel bahasa (*linguistic*) dinamakan sebagai derajat keanggotaan. Untuk variabel berlanjut (*continous variable*) derajat ini disebut fungsi keanggotaan.

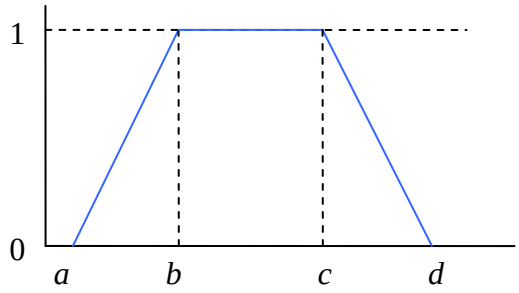
Fungsi keanggotaan segitiga dijelaskan sebagai berikut :

$$\text{Triangular}(x;a,b,c) = \begin{cases} 0, & x \le a. \\ \dfrac{x-a}{b-a}, & a \le x \le b. \\ \dfrac{c-x}{c-b}, & b \le x \le c. \\ 0, & c \le x. \end{cases}$$



Fungsi keanggotaan trapesium dijelaskan sebagai berikut :

$$\text{Trapesium}(x;a,b,c,d) \begin{cases} 0, & x \le a \text{ atau } x \ge d \\ \dfrac{x-a}{b-a} & a \le x \le b \\ 1, & b \le x \le c \\ \dfrac{d-x}{d-c} & c \le x \le d \end{cases}$$



Gambar 2.2 Fungsi keanggotaan segitiga dan trapesium

Gambar 2.2 menjelaskan tentang fungsi keanggotaan yang digunakan dalam mempresentasikan himpunan *fuzzy*. Dalam *fuzzy* fungsi keanggotaan yang biasa dipakai adalah fungsi keanggotaan segitiga, trapesium, Gaussian, fungsi keanggotaan S, fungsi keanggotaan lonceng dan sebagainya. Dalam sistem ini fungsi keanggotaan digunakan adalah fungsi keanggotaan segitiga da trapesium.

E. Aturan fuzzy

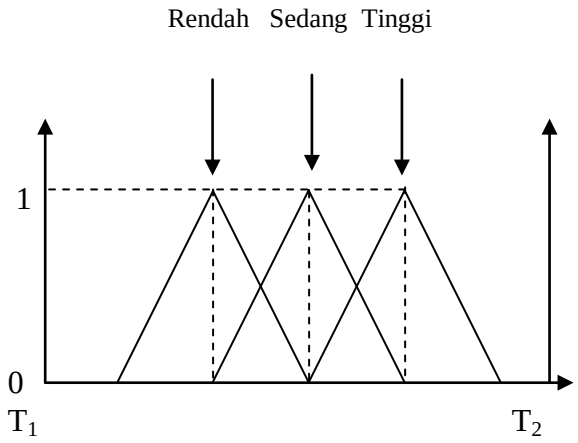
Aturan dari sistem *fuzzy* (*Fuzzy System*) menggambarkan pengetahuan dari sistem. Mereka menggunakan variabel linguistik sebagai bahasanya, sebagai contoh untuk mengekspresikan strategi pengendalian dari sebuah pengendali pengontrol *fuzzy logic*. Menjelaskan aturan *fuzzy logic* berarti menunjukkan, bagaimana menghitung dengan konsep linguistik.

2.4.2. Perhitungan fuzzy

A. Fuzzyfikasi

Proses fuzzyfikasi merupakan proses untuk mengubah variabel non *fuzzy* (variabel *numeric*) menjadi variabel *fuzzy* (variabel *linguistic*). Nilai masukan-masukan yang masih dalam bentuk variabel *numeric* yang telah dikwantisasi sebelum diolah oleh pengendali logika *fuzzy* harus diubah terlebih dahulu kedalam variabel *fuzzy*. Melalui fungsi keanggotaan yang telah disusun maka dari nilai-nilai masukan tersebut menjadi informasi *fuzzy* yang berguna nantinya untuk proses pengolahan secara *fuzzy* pula. Proses ini disebut *fuzzyfikasi*.

Proses fuzzyfikasi dalam menentukan nilai min, tengah, dan maximum pada aplikasi ditunjukkan pada gambar 2.3 berikut ini :



Gambar 2.3 Variabel Fuzzy

B. Inferensi fuzzy

Dalam inferensi *fuzzy* dilakukan proses yang dinamakan evaluasi *rule*. Tahap ini digunakan untuk mencari derajat kebenaran (*rule strength*) dari masukan *fuzzy* yang nilai keanggotaannya telah ditentukan sebelumnya pada proses fuzzyfikasi. Struktur dasar dari sistem inferensi *fuzzy* terdiri dari basis aturan yang berisi aturan *if-then*, basis data yang mendefinisikan fungsi keanggotaan dari himpunan *fuzzy*.

2.4.3. Operasi himpunan fuzzy

Terdapat 3 operasi dalam himpunan *fuzzy*, yaitu :

A. Irisan *(Intersection)*

Dalam teori himpunan klasik, irisan dari dua himpunan berisi elemen-elemen yang sama dari keduanya. Dalam himpunan *fuzzy*, sebuah elemen

mungkin sebagian dalam kedua himpunan. Oleh karena itu ketika mengingat irisan dari kedua himpunan, tidak dapat dikatakan bahwa sebuah elemen adalah lebih mungkin menjadi dalam irisan daripada dalam suatu himpunan asli.

B. Gabungan *(Union)*

Cara kedua dari penggabungan himpunan *fuzzy* adalah gabungannya. Penggabungan dari dua himpunan adalah terdiri dari dua himpunan adalah terdiri dari elemen-elemen yang menjadi satu atau dua himpunan. Dalam situasi ini anggota dari gabungan tidak dapat mempunyai nilai keanggotaan yang kurang dari nilai keanggotaan yang lain dari himpunan aslinya.

C. Komplemen (*Complement*)

Komplemen dari himpunan *fuzzy* A dinotasikan dengan (~A) dinyatakan dengan persamaan sebagai berikut : $\mu_{\sim A}(x) = 1 - \mu_A(x)$

2.4.4. Batasan (*Hedges*)

Dalam pembicaraan normal, manusia mungkin menambahkan kekaburan untuk memberikan pernyataan dengan menggunakan kata keterangan seperti sangat, agak. Kata keterangan adalah sebuah kata yang memodifikasi kata benda, kata sifat, kata keterangan lain, atau keseluruhan kalimat. Sebagai contoh, kata keterangan memodifikasi kata sifat, “orang itu sangat tinggi”. Sebuah *hedges* memodifikasi himpunan *fuzzy* yang sudah ada secara matematis untuk menghitung beberapa kata keterangan yang ditambahkan.

2.5. Fuzzy Query

Query dalam sebuah *database* digunakan untuk menjawab secara cepat beberapa pertanyaan yang diajukan oleh beberapa *user*. Beberapa tipe dari *query*

digunakan sebagai ukuran pembanding (waktu, wilayah, produk dan lainnya.) Sebuah *query* tidak dapat secara mudah direpresentasikan oleh SQL (*Structured Query Language)* sebab itu digunakan penerapan logika fuzzy guna mencari *membership function* (fungsi keanggotaan) dari fuzzy untuk menyediakan urutan bagi elemen-elemen yang ada pada sebuah *fuzzy set*.

Sebuah bentuk *membership function* mendefinisikan istilah sebuah variabel dari fuzzy. Sebuah definisi statemen merupakan pembanding sebuah bentuk statemen, sebuah istilah fuzzy dan sebuah *membership function* menspesifikasikan istilah-istilah dari logika fuzzy. Spesifikasi dari *membership function* mempunyai sebuah *function* yang mengekspresikan nama dan operasi aritmatika dalam suatu argumen.

Dalam sistem yang akan dibangun merupakan sistem basis data fuzzy (Fuzzy Database System). Relasi yang ada dalam database menekankan fuzzy pada beberapa field dalam tabel-tabel yang ada pada database tersebut. Pembuatan query menggunakan operator AND atau OR untuk menghubungkan antar variabel.

Berikut adalah contoh query beserta hasil yang diberikan:

Mencari tingkat kemiskinan berdasarkan gaji dan anak. Structure Query Language(SQL) yang dibentuk adalah :

```
Select id, gaji, anak
from penduduk
where (gaji="rendah") and (anak="banyak")
```

Tabel 2.1 Hasil Query

No	Gaji	Anak	Rendah	Banyak	Tinggi And Banyak
1	1818500	1	0	0	0
2	825200	2	0	0	0
3	743300	1	0.1134	0	0
4	667900	2	0.2642	0	0
5	733400	0	0.1332	0	0
6	1642900	2	0	0	0
7	615900	2	0.3682	0	0
8	638900	3	0.3222	1	0.3222
9	649500	2	0.301	0	0
10	658400	2	0.2832	0	0

Data yang mendekati 1 direcomendasikan mendekati miskin sehingga angka-angka yang mendekati itulah yang dapat diambil sebagai data yang dibutuhkan.

2.6. Penggunaan DFD (Data Flow Diagram)

Meskipun suatu analisa yang disebut dengan DFD mempunyai struktur tersendiri, namun sistem analisa dapat meletakkan secara bersamaan sebuah gambar yang merepresentasikan seluruh proses-proses data dalam sebuah organisasi. Pendekatan data *flow* menitikberatkan pada logika yang tersirat dari suatu sistem.

Dengan menggunakan kombinasi sistem, sistem analisa dapat membuat sebuah gambaran dari suatu proses yang sebenarnya dengan menggunakan dokumen sistem.

2.6.1. Pembuatan DFD (Data Flow Diagram)

DFD dapat dan harus digambarkan secara sistematis pertama, dibutuhkan sistem analis untuk mengkonsep data *flow*, dari atas ke bawah seperti dijelaskan sebagai berikut :

1. Membuat sebuah daftar dari kegiatan bisnis dengan menggunakan beberapa variasi, yaitu : (a) *Entity* luar (*external entity*), (b) Data *flow*, (c) Proses, dan (d) Data *store*.
2. Membuat sebuah context diagram dimana ditunjukkan *external entity* dan data *flow* ke dan dari sistem.
3. Menggambar diagram level 0, level selanjutnya.
4. Membuat sebuah *child* diagram untuk tiap-tiap proses pada level 0.
5. Pengecekan *error*.
6. Membangun sebuah DFD fisik dari DFD logika.
7. Melakukan pemisahan data.

Untuk memulai sebuah DFD dari suatu sistem biasanya dituangkan dalam sebuah daftar dengan empat kategori yaitu *entity* luar, arus data, proses, dan penyimpanan data. Daftar ini akan membantu menentukan batasan-batasan dari suatu sistem yang akan digambarkan. Pada dasarnya daftar tersebut berisi elemen-elemen data yang dikarang yang terdiri dari :

a. Pembuatan *context* diagram

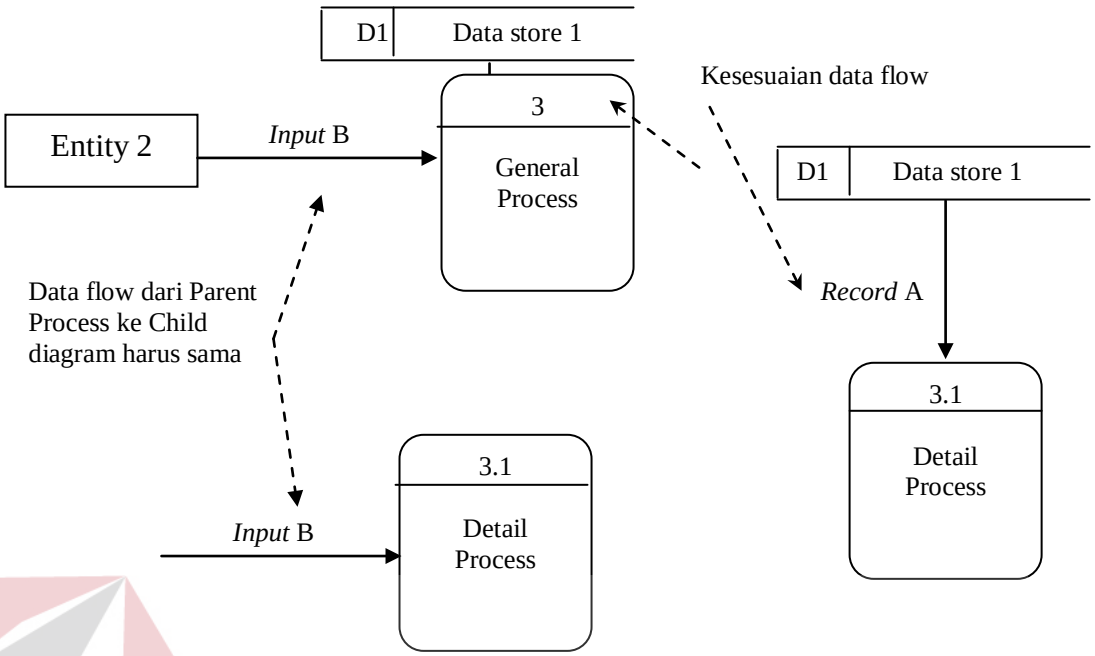
Context diagram adalah level yang tertinggi dalam sebuah DFD dan hanya berisi satu proses serta merupakan representasi dari sebuah sistem. Proses dimulai dengan penomeran ke-0 dan untuk seluruh *entity* luar akan ditunjukkan dalam *context* diagram yang sama seperti data awal yang dikirim dari *entity* luar. *Context* diagram tidak berisi penyimpanan data.

b. Pembuatan diagram level 0 serta level berikutnya

Diagram level 0 dihasilkan oleh *context* diagram dan berisi proses-proses. Pengisian proses-proses yang berlebihan pada level ini akan menghasilkan sebuah diagram yang salah, sehingga sulit untuk di mengerti . Masing-masing proses diberikan penomoran dengan sebuah bentuk *integer*. Umumnya dimulai dari kiri atas dan penyelesaiannya di kanan bawah dalam sebuah bentuk diagram.

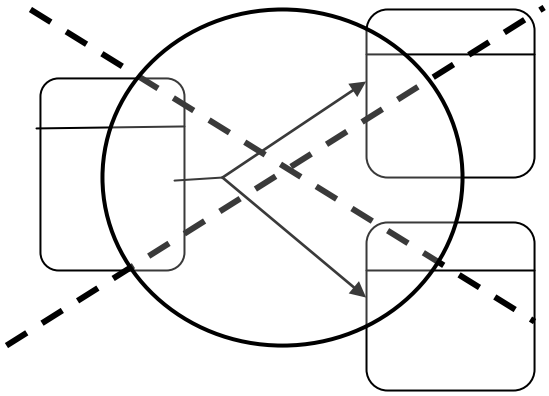
c. Pembuatan *child* diagram

Child diagram diberikan nomor yang sama seperti proses diatasnya (parent proses) dalam diagram level 0. Contohnya, proses 3 harus diturunkan ke diagram 3, proses pada *child* diagram menggunakan penomoran unik untuk masing-masing proses dengan mengikuti penomoran proses diatasnya. Contohnya, dalam diagram 3 proses-proses diberikan nomor 3.1, 3.2, 3.3 dan seterusnya. Konversi ini diikuti oleh analisi sistem untuk menelusuri seri-seri dari proses-proses yang dikeluarkan oleh beberapa level, jika pada proses diagram level 0 digambarkan sebagai 1, 2 dan 3 maka *child* diagram-diagramnya adalah 1, 2, dan 3 pada level yang sama. ilustrasi level detil dengan sebuah *child* DFD dapat ditunjukkan pada gambar 2.4 :

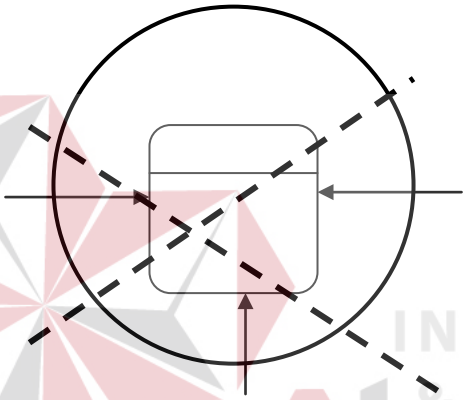


Gambar 2.4 Contoh ilustrasi detil child diagram

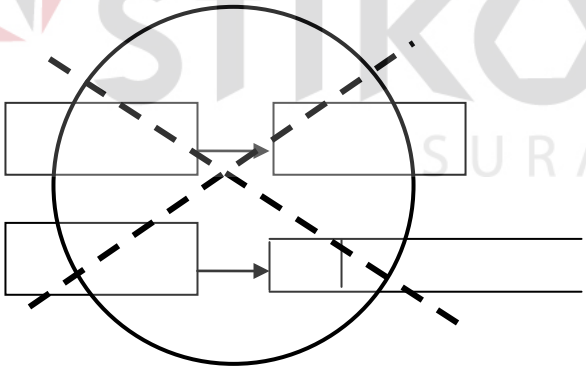
d. Pengecekan kesalahan-kesalahan pada diagram digunakan untuk melihat kesalahan-kesalahan yang terdapat pada sebuah DFD. Beberapa kesalahan-kesalahan yang umum terjadi ketika penggambaran / pembuatan DFD ditunjukkan pada gambar 2.5 berikut ini :



Sebuah data *flow* tidak diperbolehkan mempunyai percabangan / memisahkan diri (*flow*) ke dalam dua atau lebih data *flow* yang berbeda



Sebuah proses harus mempunyai minimal satu *input*an data *flow* dan satu *output* data *flow*



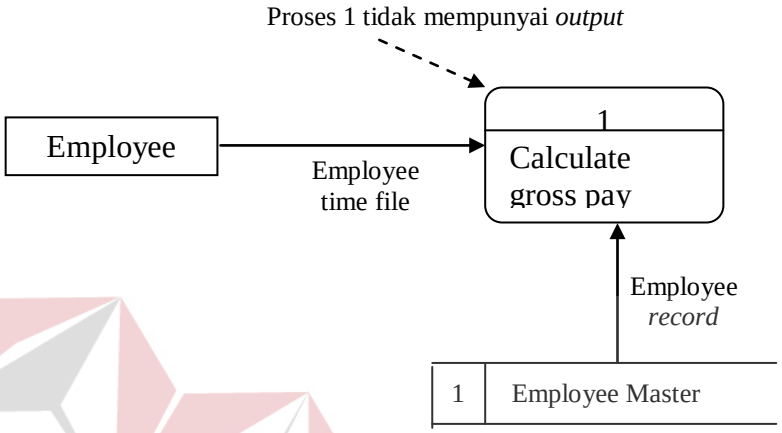
Semua data *flow* salah satunya harus berasal atau berakhir pada sebuah proses

Gambar 2.5 Salah satu contoh kesalahan penulisan proses dalam DFD

Beberapa kesalahan penulisan yang juga umum terjadi pada proses pembuatan / penggambaran DFD, antara lain diuraikan sebagai berikut :

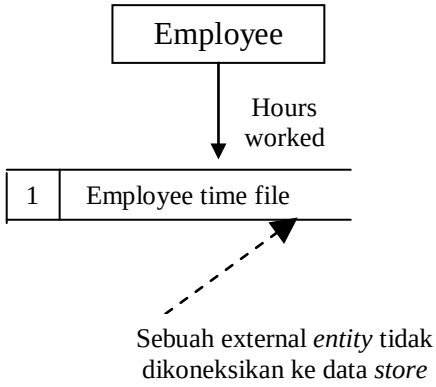
1. Tidak menginputkan sebuah arus data atau arah panah langsung. Sebagai contoh adalah penggambaran proses yang menunjukkan sebuah data *flow*

seperti *input* atau seperti *output*. Tiap-tiap proses pengubahan data harus menerima *input* dan *output*. Tipe kesalahan ini terjadi ketika sistem analis tidak memasukkan sebuah data *flow* atau meletakkan sebuah arah panah di tempat yang salah seperti ditunjukkan pada gambar 2.6 berikut ini :



Gambar 2.6 Kesalahan proses input dan output

2. Hubungan penyimpanan data dan *entity* luar secara langsung satu sama lain. Data *store* dan *entity* luar harus dikoneksikan melalui sebuah proses seperti ditunjukkan pada gambar 2.7 berikut ini :



Gambar 2.7 Kesalahan penulisan hubungan entity luar dengan data store

3. Kesalahan penamaan (label) pada proses-proses atau data *flow*. Pengecekan DFD untuk memastikan bahwa tiap-tiap obyek atau data *flow* telah diberikan label. Sebuah proses haruslah diindikasikan seperti nama dari sistem atau menggunakan format kata kerja. Tiap data *flow* haruslah dideskripsikan dengan sebuah kata benda.
4. Memasukkan lebih dari sembilan proses dalam sebuah DFD. Memiliki banyak proses akan mengakibatkan kekacauan pada diagram sehingga dapat menyebabkan kebingungan dalam pembacaan sebuah proses dan akan menghalangi tingkat komunikasi. Jika lebih dari sembilan proses dalam sebuah sistem, maka beberapa grup dalam proses dilakukan bersama-sama ke dalam sebuah sub sistem dan meletakkannya dalam sebuah *child* diagram.
5. Menghilangkan suatu arus data. Pengujian dari suatu diagram yang menunjukkan garis / arah (*flow*), dimana untuk setiap proses data *flow* hanya mempunyai *input* data, *output* kecuali dalam kasus dari detil (*child*). Setiap *child* data dari DFD, arah arus data seringkali digambarkan untuk mengidentifikasikan bahwa diagram tersebut kehilangan data *flow*.
6. Membuat ketidaksesuiaan komposisi dalam *child* diagram , dimana tiap *child* diagram harus mempunyai *input* dan *output* arus data yang sama seperti proses dilevel atasnya (*parent process*). Pengecualian untuk *rule* ini adalah kurangnya *output*, seperti kesalahan garis yang ada didalam *child* diagram.

2.6.2. Keuntungan pembuatan data flow

Data *flow* mempunyai lima keuntungan utama dari penjelasan-penjelasan jalannya data dalam sistem, yaitu :

1. Kebebasan yang berasal dari kepercayaan untuk mengimplementasikan secara benar teknik sistem dari suatu sistem yang baru.
2. Memberikan pengertian dari hubungan sistem-sistem dan subsistem yang ada.
3. Komunikasi mengenai pengetahuan sistem bagi *user* melalui DFD
4. Analisa dari sebuah usulan sistem untuk menentukan jika data dan proses-proses yang ada dapat didefinisikan secara mudah.
5. Penggunaan data *flow* merupakan keuntungan tambahan yang dapat digunakan sebagai latihan bagi sistem analis, kesempatan sistem analis menjadi lebih mengerti tentang hubungan sistem dan subsistem yang ada didalamnya.

2.6.3. Perbedaan DFD logika dan DFD fisik

Perbedaan antara DFD (Data Flow Diagram) secara fisik dan logika dijabarkan pada tabel 2.2 berikut ini :

Tabel 2.2 Perbedaan DFD logika dan fisik

Desain	Logika	Fisik
Gambaran model	Operasi-operasi bisnis	Bagaimana sistem akan diimplementasikan
Tampilan proses	Aktivitas-aktivitas	Program-program, modul, dan prosedur-prosedur manual
Tampilan data store	Koleksi-koleksi dari data yang dikesampingkan dari bagaimana data tersebut di simpan	*File-file* fisik dan *database-database* dari *file-file* manual
Kontrol sistem	Menunjukkan kontrol-kontrol bisnis	Menunjukkan kontrol-kontrol untuk validasi *input* data, untuk memperoleh sebuah *record* , untuk memastikan kesuksesan proses dan untuk keamanan sistem

2.7. Desain Output

Output merupakan sebuah informasi yang dikirimkan kepada *user* melalui suatu sistem informasi (seperti : Internet, Extranet, dan WWW). Beberapa data yang dibutuhkan diproses secara teliti sebelum dinyatakan layak sebagai sebuah *output*; penyimpanan data lain, dan ketika data-data tersebut dibutuhkan

kembali, data-data tersebut berupa *ouput* yang membutuhkan proses yang sedikit. *Output* dapat di ambil dari beberapa bentuk yaitu : bentuk laporan yang dihasilkan printer dan tampilan layar komputer, mikrofon (suara).

Oleh sebab itu, penggunaan *output* sangat penting guna menjamin pemakaian dan penerimaan dari suatu sistem informasi. Ada beberapa obyek dimana sistem analis mencoba untuk mencapai suatu desain *ouput* yang tepat. Obyek-obyek tersebut antara lain:

1. Desain *output* untuk melayani sebuah tujuan tertentu.
2. Pembuatan *output* yang disesuaikan bagi kebutuhan *user*.
3. Pengiriman sejumlah *output*.
4. Pengelolaan distribusi *output*.
5. Pengelolaan waktu *output*.
6. Penyesuaian metode *output* yang paling efektif.

2.8. Desain Input

Bentuk-bentuk desain secara khusus mungkin dapat digunakan jika suatu analisa sistem dapat disesuaikan dengan bentuk desain secara lengkap dan bermanfaat. Hal itu juga penting untuk mengenalkan secara dini desain yang digunakan, arus data atau bentuk-bentuk yang tidak dibutuhkan pada sumber-sumber suatu organisasi dan hal itu harus dihilangkan. Untuk mendesain *form-form* yang baik dan berguna, ada beberapa hal dari desain *form* yang harus diterapkan antara lain :

1. Membuat *form-form* tersebut mudah dalam pengisiannya.

2. Memastikan *form-form* tersebut sesuai dengan tujuan untuk masing-masing desain.
3. Desain *form* digunakan untuk menjamin kelengkapannya.
4. Mempertahankan *form-form* yang menarik.

2.9. Desain Database

Desain *database* merupakan gambaran atau deskripsi dari file-file yang digunakan serta merupakan sebuah pendefinisian normal dan merupakan gambaran dari pusat penyimpanan dari data tertentu yang digunakan dalam beberapa aplikasi yang berbeda. Desain *database* terdiri dari :

2.9.1. Database

Database bukan hanya merupakan sebuah koleksi dari suatu file-file. Meskipun, sebuah *database* merupakan sebuah pusat sumber data yang disimpan oleh beberapa *user* dari sebuah aplikasi-aplikai yang bervariasi. Inti dari sebuah *database* adalah DBMS (*Database Management* Sistem), dimana diikuti dengan kreasi, modifikasi, dan perubahan (*update*) dari *database*.

2.9.2. Bentuk database dan file

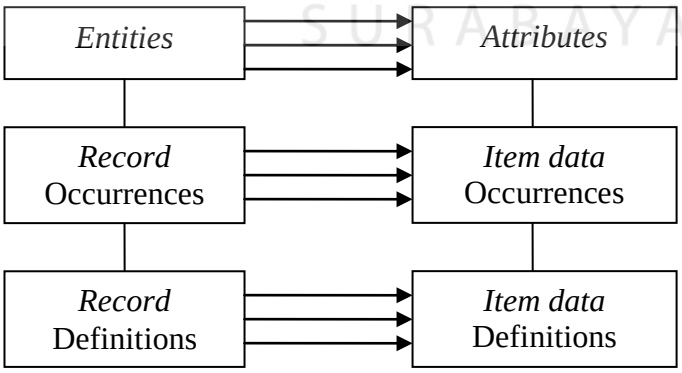
Ada dua bentuk pendekatan pada proses penyimpanan data dalam sebuah sistem komputer. Metode pertama adalah untuk menyimpan file-file tunggal, masing-masing dengan bentuk unik untuk berbagai macam aplikasi. Pendekatan yang kedua adalah untuk menyimpan data dalam sebuah sistem komputer dengan melibatkan pembuatan sebuah *database*.

2.9.3. Konsep-konsep data

Konsep-konsep data merupakan hal yang sangat penting untuk di mengerti bagaimana data dipresentasikan kembali sebelum memutuskan penggunaan *file* atau *database*. Konsep-konsep tersebut terdiri dari :

A. Realita, data dan metadata

Realita data merupakan gambaran attribut-attribut yang dimiliki oleh sekumpulan data. Data merupakan representasi fakta dunia nyata yang mewakili suatu objek. Metadata merupakan data dimana dijelaskan tentang data *file* atau *database* serta menjelaskan pemberian nama dan menunjukkan panjang dari masing-masing item data. Metadata juga menjelaskan panjang dan komposisi dari tiap-tiap *record*. Gambaran dari hubungan antara realita, data dan meta data yaitu didalamnya terdapat *entity* dan attribut (realita), *record* dan *item data* (data), definisi *record* dan definisi *item data* (meta data). Ditunjukkan pada gambar 2.8 berikut :



Gambar 2.8 Hubungan antara realita, data dan metadata

dimana :

- *Entity* → Merupakan beberapa obyek atau kejadian tentang dimana dapat mencocokkan koleksi data sebagai sebuah *entity*. *Entity* juga dapat berupa sebuah kejadian atau unit dari satu waktu.
- *Relationship*→ Merupakan hubungan antara *entity*, adapun tipe-tipe dari *relationship* antara lain (1) *one to one* (1:1), (2) *one to many* (1:M), dan (3) *many to many* (M:N).
- *Attribute* → Merupakan sebuah karakteristik dari sebuah *entity*.
- *Record* → Merupakan suatu kumpulan dari item-item data yang secara umum merupakan penjelasan umum dari *entity*.

B. Kelompok file

Sebuah *file* berisi grup-grup dari *record* yang digunakan untuk melengkapi informasi untuk suatu operasi-operasi, perencanaan, manajemen, dan pembuatan keputusan. Tipe-tipe dari *file* yang digunakan antara lain :

1. *File master* : berisi *record-record* dari sebuah kelompok *entity*.
2. *File* tabel : berisi data yang digunakan.
3. *File-file* transaksi : digunakan untuk mengisi perubahan (*update*) sebuah *file master* dan laporan-laporan.
4. *Work file* : digunakan untuk menjalankan program agar lebih efektif.
5. *File* laporan : Memudahkan untuk menjalankan program (ketika tidak ada printer).

2.10. Normalisasi

Normalisasi merupakan perubahan dari *user* yang ditampilkan secara lengkap dan simpanan data untuk ukuran dari yang terkecil, serta merupakan struktur-struktur data yang stabil. Normalisasi dibutuhkan untuk mengorganisir data dan menghindari redudansi data (data *double*). Ada tiga bentuk normalisasi, antara lain :

1. *First Normal Form* (1NF)

   Langkah pertama ini terdapat pada sebuah relasi normalisasi yang digunakan untuk menghilangkan (menghapus) grup-grup yang berulang.

2. *Second Normal Form* (2NF)

   Pada bentuk normal yang kedua, seluruh attribut yang ada akan difungsikan tergantung pada PK (*Primary Key*).

3. *Third Normal Form* (3NF)

   Sebuah relasi penormalisasian yang ketiga adalah jika seluruh yang bukan kunci PK (*Primary Key*) dari semua attribut seluruhnya difungsikan bergantung pada PK dan attribut tersebut bukan transitif ketergantungan (tanpa kunci).

2.11. Desain User Interface

Bagaimanapun baik atau buruknya (minimnya) suatu tampilan, hal itu berpengaruh pada keberadaan representasi sebuah sistem. Desain terdiri dari :

1. Tipe-tipe *user interface* (tampilan)
2. Tampilan bahasa
3. Tampilan tanya – jawab

4. Menu-menu
5. Tampilan bentuk isian / bentuk *input* dan *output*
6. Tampilan bahasa perintah
7. GUIs (*Graphical User Interface*)