

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/340599533>

Basis Data Dasar

Book · April 2020

CITATIONS

0

READS

854

1 author:



[Adyanata Lubis](#)

STKIP Rokania

10 PUBLICATIONS 12 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



Basis Data Dasar [View project](#)

BASIS DATA DASAR

UU No 19 Tahun 2002 Tentang Hak Cipta

Fungsi dan Sifat hak Cipta Pasal 2

1. Hak Cipta merupakan hak eksklusif bagi pencipta atau pemegang Hak Cipta untuk mengumumkan atau memperbanyak ciptaannya, yang timbul secara otomatis setelah suatu ciptaan dilahirkan tanpa mengurangi pembatasan menurut peraturan perundang-undangan yang berlaku.

Hak Terkait Pasal 49

1. Pelaku memiliki hak eksklusif untuk memberikan izin atau melarang pihak lain yang tanpa persetujuannya membuat, memperbanyak, atau menyiarkan rekaman suara dan/atau gambar pertunjukannya.

Sanksi Pelanggaran Pasal 72

1. Barangsiapa dengan sengaja dan tanpa hak melakukan perbuatan sebagaimana dimaksud dalam pasal 2 ayat (1) atau pasal 49 ayat (2) dipidana dengan pidana penjara masing-masing paling singkat 1 (satu) bulan dan/atau denda paling sedikit Rp 1.000.000,00 (satu juta rupiah), atau pidana penjara paling lama 7 (tujuh) tahun dan/atau denda paling banyak Rp 5.000.000.000,00 (lima miliar rupiah).
2. Barangsiapa dengan sengaja menyiarkan, memamerkan, mengedarkan, atau menjual kepada umum suatu ciptaan atau barang hasil pelanggaran Hak Cipta sebagaimana dimaksud dalam ayat (1), dipidana dengan pidana penjara paling lama 5 (lima) tahun dan/atau denda paling banyak Rp 500.000.000,00 (lima ratus juta rupiah)

BASIS DATA DASAR

Adyanata Lubis, S.Kom., M.Kom.





deepublish | publisher

Jl.Rajawali, G. Elang 6, No 3, Drono, Sardonoharjo, Ngaglik, Sleman
Jl.Kaliurang Km.9,3 – Yogyakarta 55581
Telp/Faks: (0274) 4533427
Website: www.deepublish.co.id
www.penerbitdeepublish.com
E-mail: deepublish@ymail.com

Katalog Dalam Terbitan (KDT)

LUBIS, Adyanata

Basis Data Dasar/oleh Adyanata Lubis.--Ed.1, Cet. 1--
Yogyakarta: Deepublish, Maret 2016.

viii, 124 hlm.; Uk:17.5x25 cm

ISBN 978-Nomor ISBN

1. Basisdata

I. Judul

005.74

Hak Cipta 2016, Pada Penulis

Desain cover : Elyandri Prasiwiningrum

Penata letak : Invalidiant Candrawinata

PENERBIT DEEPUBLISH
(Grup Penerbitan CV BUDI UTAMA)

Anggota IKAPI (076/DIY/2012)

Copyright © 2016 by Deepublish Publisher
All Right Reserved

Isi diluar tanggung jawab percetakan

Hak cipta dilindungi undang-undang
Dilarang keras menerjemahkan, memfotokopi, atau
memperbanyak sebagian atau seluruh isi buku ini
tanpa izin tertulis dari Penerbit.

KATA PENGANTAR

Dalam pembuatan sebuah sistem aplikasi, diperlukan sebuah basis data yang baik dan benar. Untuk menghasilkan basis data yang baik dan benar haruslah dilakukan perancangan dengan baik dan benar pula.

Perancangan basis data dapat dilakukan dengan beberapa cara, salah satunya adalah dengan membuat *Entity Relationship Diagram* (ER-D). Pembacaan ER-D memang mudah dilakukan, namun bagaimana merancang ER-D yang baik dan benar?

Dalam buku ini dibahas tentang ER-D dari simbol hingga langkah-langkah pembuatan ER-D yang baik dan benar. Selain itu buku ini juga membahas kasus yang ada dan cara penyelesaiannya. Sehingga pembaca, khususnya mahasiswa Fakultas Ilmu Komputer dapat membuat ER-D setahap demi setahap untuk menghasilkan basis data yang baik dan benar.

Pembuatan buku ini memang masih jauh dari sempurna, oleh karena itu penulis membutuhkan masukan kritik atau saran ini yang bersifat membangun, demi kesempurnaan buku ini.

Pasir Pengairan, Januari 2016

Adyanata Lubis, S.kom,.M.Kom.

DAFTAR ISI

| | |
|--|-----------|
| KATA PENGANTAR..... | v |
| DAFTAR ISI..... | vii |
| BAB I KONSEP BASIS DATA | 1 |
| 1.1. Pengertian Data..... | 1 |
| 1.2. Komponen Dasar Sistem Basis Data | 6 |
| 1.3. Data pada Basis Data..... | 8 |
| 1.4. Keuntungan dan Kerugian Penggunaan Basis Data..... | 8 |
| 1.5. Perbedaan Traditional File Management (TFM) dengan Database Management System (DBMS) | 9 |
| 1.6. Istilah-istilah yang dipergunakan pada Sistem Basis Data | 9 |
| BAB II ARSITEKTUR BASIS DATA (DATABASE) | 11 |
| 2.1. Hierarki <i>Database</i> | 11 |
| 2.2. Tingkatan Arsitektur <i>Database</i> | 11 |
| 2.3. Abstraksi Hubungan antara <i>User</i> pada DBMS dengan <i>Physical Database</i> | 12 |
| 2.4. Arsitektur DBMS <i>Multi User</i> | 12 |
| 2.5. Karakteristik <i>Client-Server</i> | 16 |
| 2.6. Keuntungan dan Kelemahan Sistem <i>Client- Server</i> | 16 |
| 2.7. Arsitektur Dua dan Tiga-Tier | 17 |
| 2.8. Pendekatan Lain <i>Client-Server</i> | 18 |
| 2.9. Contoh Proses dari Sistem <i>Client-Server</i> | 18 |
| 2.10. Pengguna (<i>User</i>) Basis Data..... | 19 |
| 2.11. Bahasa Pemrograman Basis Data | 19 |
| BAB III DATABASE MANAGEMENT SYSTEM (DBMS) | 25 |
| 3.1. Definisi DBMS..... | 25 |
| 3.2. Komponen DBMS..... | 25 |

| | |
|--|------------|
| 3.3. Data Independence | 26 |
| BAB IV MODEL DATA..... | 27 |
| 4.1. Pengertian Data model | 27 |
| 4.2. Jenis-jenis Data model | 27 |
| 4.3. Perbedaan dengan Objek Based Data Model..... | 37 |
| BAB V ENTITY RELATIONSHIP DIAGRAM (ER-D)..... | 38 |
| 5.1. Simbol yang Digunakan pada ER-D | 39 |
| 5.2. Hal yang Dilarang dalam Pembuatan ER-D..... | 41 |
| 5.3. Pembatasan Pemetaan..... | 41 |
| 5.4. Tahapan Pembuatan ER-D | 44 |
| 5.5. Varian Entitas..... | 50 |
| 5.6. Spesialisasi dan Generalisasi..... | 54 |
| 5.7. Agregasi (<i>Aggregation</i>) | 59 |
| 5.8. Varian relasi..... | 63 |
| BAB VI NORMALISASI DAN DENORMALISASI..... | 97 |
| 6.1. Normalisasi Data | 97 |
| 6.2. Denormalisasi Data..... | 115 |
| DAFTAR PUSTAKA..... | 122 |
| TENTANG PENULIS..... | 123 |

BAB I

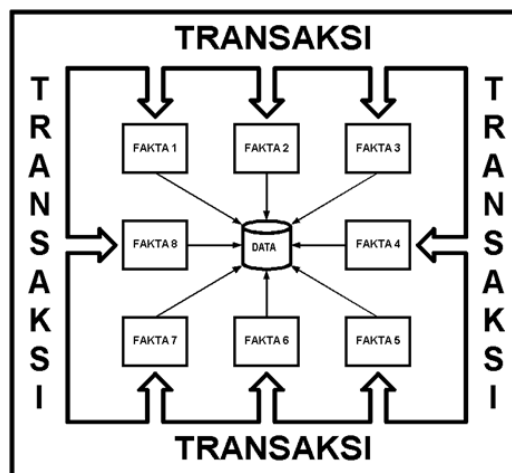
KONSEP BASIS DATA

1.1. Pengertian Data

Sebelum mengetahui lebih jauh tentang basis data, ada baiknya kita mengenal apa yang dimaksud dengan data. Data adalah :

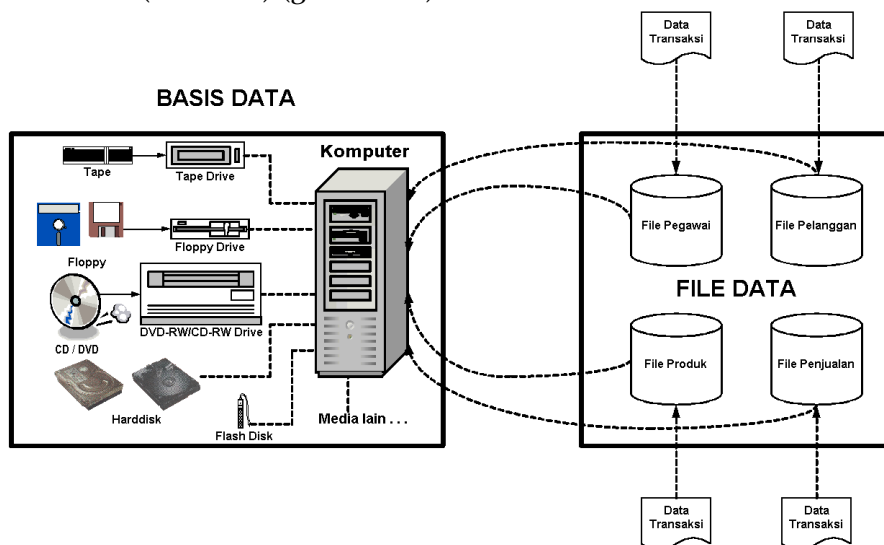
”Fakta-fakta yang menggambarkan suatu kejadian yang sebenarnya pada waktu tertentu”

Jadi data didapat dari suatu kejadian yang benar-benar terjadi, misalnya data penjualan didapat dari data hasil penjualan, data pembelian didapat dari kejadian pembelian, dan sebagainya. Data dalam suatu perusahaan yang menjual produk identik dengan bukti transaksi, misalnya kuitansi pembayaran dan sebagainya. Sedangkan dalam suatu perusahaan, data dapat diidentikkan dengan laporan tertulis baik antar departemen maupun dari luar perusahaan, misalnya bukti pajak, bukti bank dan sebagainya. Namun yang terpenting adalah bahwa data tersebut harus mempunyai bukti tertulis agar dapat ditelusuri dari mana data tersebut berasal, sebagai bukti adanya suatu transaksi, dan seterusnya. Untuk lebih jelasnya dapat dilihat pada gambar 1.1 dibawah ini :



Gambar 1.1. Data yang didapat dari fakta pada transaksi

Setelah semua fakta-fakta diterima, kemudian fakta-fakta tersebut akan dimasukan dan dikelompokkan kedalam data. Misalnya fakta penjualan barang, fakta dana promosi dan sebagainya, dimasukan kedalam data penjualan atau *marketing*. Demikian dengan data yang lain, misalnya data pembelian, data gudang, data pegawai dan seterusnya. Kemudian data tersebut akan dikelompokkan dan dimasukan kedalam *file-file* yang disebut dengan *file data*. Dari *file data* tersebut barulah akan dimasukan dan di hubungan antara *file data* yang satu dengan *file data* yang lain dalam sebuah tempat atau wadah penampungan data yang dikenal dengan basis data (*database*) (gambar 1.2).



Gambar 1.2. : Hubungan data dengan basis data

Basis data (*database*) dalam dunia komputer, terutama oleh pemrogram (*programmer*) sudah tidak asing lagi karena seringkali disinggung dan berhubungan langsung. Namun untuk memudahkan memahami apa yang dimaksud dengan basis data, ada baiknya dibahas terlebih dahulu apa yang dimaksud dengan basis data.

Basis data merupakan gabungan *file data* yang dibentuk dengan hubungan/relasi yang logis dan dapat diungkapkan dengan catatan serta bersifat independen. Adapun basis data adalah :

"Tempat berkumpulnya data yang saling berhubungan dalam suatu wadah (organisasi/perusahaan) bertujuan agar dapat mempermudah dan mempercepat untuk pemanggilan atau pemanfaatan kembali data tersebut."

Arti lain dari sistem basis data adalah :

"Suatu sistem penyusunan dan pengelolaan record-record dengan menggunakan komputer, dengan tujuan untuk menyimpan atau merekam serta memelihara data secara lengkap pada sebuah organisasi / perusahaan, sehingga mampu menyediakan informasi yang optimal yang diperlukan pemakai untuk kepentingan proses pengambilan keputusan."

Pada kehidupan sehari-hari di dunia komputer, basis data akan menggunakan media penyimpanan (*storage*), yaitu berkaitan dengan setiap alat yang dapat menerima data yang dapat disimpan, dan dapat dipanggil kembali data itu pada waktu berikutnya atau setiap alat yang dapat digunakan untuk menyimpan data. Adapun media penyimpanan yang digunakan terdiri dari *harddisk*, *diskette* atau *floppy disk*, *tape* maupun dengan *compact disk* (CD) atau DVD dan kini juga dapat digunakan *flashdisk*. Penggunaan CD memang masih baru digunakan akhir-akhir, hal ini dimungkinkan dengan dibuatnya suatu alat yang disebut dengan CD-RW (*read-write*). Bahkan dengan perkembangan teknologi, kini telah dibuat alat DVD-RW. Penggunaan DVD ini sangat baik sekali, karena DVD dapat menyimpan data yang jauh lebih besar dibandingkan dengan CD. (gambar 1.2)

Dengan bantuan basis data ini diharapkan bahwa sistem informasi yang dibuat dapat terintegrasi antara bagian / departemen yang satu dengan yang lainnya, sehingga pada akhirnya tidak ada pembatas area dalam perusahaan. Walaupun dalam pelaksanaannya tiap data akan dibatasi oleh penggunaanya, namun semua hanya ditujukan untuk membatasi pengaksesan data saja agar tidak terjadi pembuatan manipulasi data oleh orang yang tidak berkepentingan terhadap data tersebut.

Dalam pembuatan dan penggunaan basis data, terdapat 4 (empat) komponen dasar sistem basis data, yaitu:

1. Data

Data yang digunakan dalam sebuah basis data, haruslah mempunyai ciri sebagai berikut:

- Data disimpan secara terintegrasi (*integrated*), yaitu *Database* merupakan kumpulan dari berbagai macam *file* dari aplikasi-aplikasi yang berbeda yang disusun dengan cara menghilangkan bagian-bagian yang rangkap (*redundant*).
- Data dapat dipakai secara bersama-sama (*shared*), yaitu masing-masing bagian dari *database* dapat diakses oleh pemakai dalam waktu yang bersamaan, untuk aplikasi yang berbeda.

2. Hardware

Terdiri dari semua peralatan perangkat keras komputer yang digunakan untuk pengelolaan sistem *database*, seperti:

- Peralatan untuk penyimpanan, disk, drum, dll
- Peralatan *input* dan *output*
- Peralatan komunikasi data, dll

3. Software

Berfungsi sebagai perantara (*interface*) antara pemakai dengan data fisik pada *database*, dapat berupa :

- *Database Management System* (DBMS).
- Program-program aplikasi & prosedur-prosedur yang lain, seperti Oracle, SQL Server, MySQL, dll.

4. User (Pengguna)

Terbagi menjadi 3 klasifikasi :

- *Database Administrator* (DBA), yaitu orang/*team* yang bertugas mengelola sistem *database* secara keseluruhan
- *Programmer*, yaitu orang/*team* membuat program aplikasi yang mengakses *database* dengan menggunakan bahasa pemrograman.
- *End user*, orang yang mengakses *database* melalui terminal dengan menggunakan *query language* atau program aplikasi yang dibuat oleh *programmer*.

Penggunaan basis data pada komputer memang tidak terlepas dari hubungan atau relasi antar data dalam bentuk *file*. Adapun *file* menurut Noor (1990) adalah :

”Suatu pengumpulan yang terorganisir dari catatan yang saling berhubungan. Misalnya satu garis (*line*) dari faktur dapat berbentuk item, suatu faktur dapat membentuk catatan, suatu serangkaian catatan yang sedemikian ini dapat membentuk suatu *file*, pengumpulan dari *file* kontrol keuangan dapat berbentuk perpustakaan (*library*), dan keseluruhan perpustakaan yang digunakan oleh suatu organisasi dalam membentuk bank data.”

Dalam kehidupan sehari-hari *file* data identik sekali dengan tabel-tabel yang terdiri dari *field* dan *record*. Pada beberapa tipe konvensional yang termasuk *file* dan tabel menurut Jeffrey L. Whitten et. al. (2001) antara lain :

1. *File master*

File master merupakan tabel yang berisi *record* yang bersifat tetap. Sekali *record* ditambahkan kedalam *file master*, akan tetap dan dapat digunakan selama dioperasikan pada sebuah sistem. Sedangkan perubahan untuk *record* biasanya dilakukan untuk waktu yang tidak terbatas, akan tetapi tiap *record* akan dapat bertahan terhadap sekumpulan *record* dalam batas waktu yang tidak terbatas. Biasanya *file master* ini digunakan sebagai pedoman dalam penggabungan beberapa *file* yang ada. Contoh *file master* adalah : *file* karyawan, *file* pelanggan, *file* produk, *file* pemasok dan lain-lain.

2. *File transaksi*

Yang termasuk *file transaksi* pada suatu perusahaan adalah tabel yang berisi gambaran tiap peristiwa perusahaan. Penggambaran yang dimaksud adalah penggambaran dalam skala yang normal atau tidak dibuat-buat dan dalam waktu yang terbatas. Pembuatan tabel transaksi tidak dapat diwakili atau kolektif, melainkan harus dapat menggambarkan semua transaksi yang terjadi. Yang termasuk *file transaksi* adalah *file* pendaftaran/registrasi, *file* pembelian, *file* pengiriman dan lain-lain.

3. *File dokumen*

Pembuatan *file* dokumen dalam suatu perusahaan memang penting, ini menyangkut pada pembuatan dokumen yang bersifat sejarah atau perjalanan waktu. Adapun yang dimaksud dengan *file* dokumen adalah tabel yang berisi penyimpanan salinan *record* yang berisi data yang bersejarah untuk memudahkan pencarian kembali dan dapat dilihat dengan menggunakan *overhead*.

4. *File arsip*

File arsip adalah file atau tabel yang berisi file master atau transaksi yang telah dihapus dari penyimpanan secara langsung. Penghapusan bertujuan untuk memindahkan data dari penyimpanan data yang *on-line* ke penyimpanan data yang *off-line*. Pembuatan *file* arsip ini digunakan untuk kebutuhan dalam kebijakan pemerintah dan kebutuhan saat audit atau analisis.

5. *File table look-up*

Dalam pembuatan sistem informasi, relasi dalam tabel mungkin banyak digunakan. Oleh karena itu yang dimaksud dengan *file table look-up* adalah tabel yang berisi hubungan statis data yang digunakan bersama-sama oleh aplikasi untuk perawatan secara tetap dan dapat memperbaiki kinerja dari sistem, contohnya tabel pajak penjualan, tabel pendapatan pajak dan-lain-lain.

6. *File audit*

Update File audit untuk file-file yang lain, khususnya *file* master dan transaksi. Semuanya digunakan untuk bersama dengan file arsip untuk menelusuri data yang hilang (*lost*). Penelusuran jejak dengan audit dapat digunakan membangun teknologi basis data yang lebih baik.

1.2. **Komponen Dasar Sistem Basis Data**

Komponen dasar sistem basis data digunakan untuk membantu kelancaran dari pembuatan dan manajemen basis data.

Adapun komponen dasar basis data terdiri dari 4 komponen pokok, yaitu :

a. Data

Data pada sistem basis data mempunyai ciri-ciri sebagai berikut :

- Data disimpan secara terintegrasi (*integrated*).
Ter-*integrated* yaitu *Database* merupakan kumpulan dari berbagai macam file dari aplikasi-aplikasi yang berbeda yang disusun dengan cara menghilangkan bagian-bagian yang rangkap (*redundant*).
- Data dapat dipakai bersama-sama (*shared*).
Shared yaitu Masing-masing bagian dari *database* dapat diakses oleh pemakai dalam waktu yang bersamaan, untuk aplikasi yang berbeda.

b. *Hardware* (perangkat keras)

Terdiri dari semua peralatan perangkat keras komputer yang digunakan untuk pengelolaan sistem database antara lain :

- Peralatan untuk penyimpanan, disk, drum, dll.
- Peralatan *input* dan *output*.
- Peralatan komunikasi data, dll.

c. *Software* (perangkat lunak)

Berfungsi sebagai perantara (*interface*) antara pemakai dengan data fisik pada *database*, dapat berupa :

- *Database Management System* (DBMS).
- Program-program aplikasi & prosedur-prosedur.

d. *User* (pemakai)

Terbagi menjadi 2 bagian, yaitu :

- *Programmer*, orang/team membuat program aplikasi yang mengakses database dengan menggunakan bahasa pemrograman.
- *End user*, orang yang mengakses *database* melalui terminal dengan menggunakan *query language* atau program aplikasi yang dibuat oleh *programmer*.

1.3. Data pada Basis Data

Penggunaan data pada basis data mempunyai peran yang kuat pada masing-masing bagian, yang dikelompokkan dalam 3 jenis data pada sistem basis data, yaitu:

- Data operasional dari suatu organisasi, berupa data yang disimpan di dalam basis data
- Data masukan (*input data*), data dari luar sistem yang dimasukan melalui peralatan input (*keyboard*) yang dapat mengubah data operasional
- Data keluaran (*output data*), berupa laporan melalui peralatan output (*screen, printer*) sebagai hasil dari dalam sistem yang mengakses data operasional

1.4. Keuntungan dan Kerugian Penggunaan Basis Data

Penggunaan basis data pada sebuah perusahaan mempunyai keuntungan, antara lain :

1. Terkontrolnya kerangkapan data dan inkonsistensi,
2. Terpeliharanya keselarasan data,
3. Data dapat dipakai secara bersama-sama,
4. Memudahkan penerapan standarisasi,
5. Memudahkan penerapan batasan - batasan pengamanan,
6. Terpeliharanya integritas data,
7. Terpeliharanya keseimbangan atas perbedaan kebutuhan data dari setiap aplikasi,
8. Program / data *independent*.

Adapun kerugiannya penggunaan basis data pada sebuah perusahaan, antara lain :

1. Mahal dalam implementasinya,
2. Rumit / kompleks,
3. Penanganan proses *recovery & backup* sulit,
4. Kerusakan pada sistem basis data dapat mempengaruhi departemen yang terkait,
5. dan lain – lain.

1.5. Perbedaan Traditional File Management (TFM) dengan Database Management System (DBMS)

Pada awal penggunaan data, perusahaan masih menggunakan data yang terpisah-pisah penggunaannya (*traditional file management*/TFM), tergantung pemrogram yang membuatnya. Sehingga terdapatnya perbedaan data yang antara pemrogram yang satu dengan yang lain, dan berdampak pada kerangkapan data yang dibuat. Dengan dibuatnya basis data (*database management system*/DBMS), perbedaan dan kelemahan tersebut dapat diatasi. Adapun perbedaan antara *traditional file management* (TFM) dengan *database management system* (DBMS), yaitu :

a. *Traditional File Management*

- Bersifat *program oriented*
- Bersifat kaku
- Terjadi kerangkapan data dan tidak terjaminnya keselarasan data (data inkonsistensi)

Keterangan :

Program oriented “Susunan data di dalam *file*, distribusi data pada peralatan *storage*, dan organisasi *file*-nya dipilih sedemikian rupa, sehingga program aplikasi dapat menggunakan secara optimal”.

b. *Database management System*

- Bersifat *data oriented*
- Bersifat luwes/fleksibel
- Kerangkapan data serta keselarasan data dapat terkontrol

Keterangan :

Data oriented “Susunan data, organisasi *file* pada *database* dapat dirubah, begitu pula strategi aksesnya tanpa mengganggu program aplikasi yang sudah ada”.

1.6. Istilah-istilah yang dipergunakan pada Sistem Basis Data

Untuk mempermudah pembuatan dan penggunaan basis data, digunakan beberapa istilah, antara lain :

a. *Enterprise*, suatu bentuk organisasi.

Contoh : data sekolah -----> data mhs
rumah sakit -----> pasien

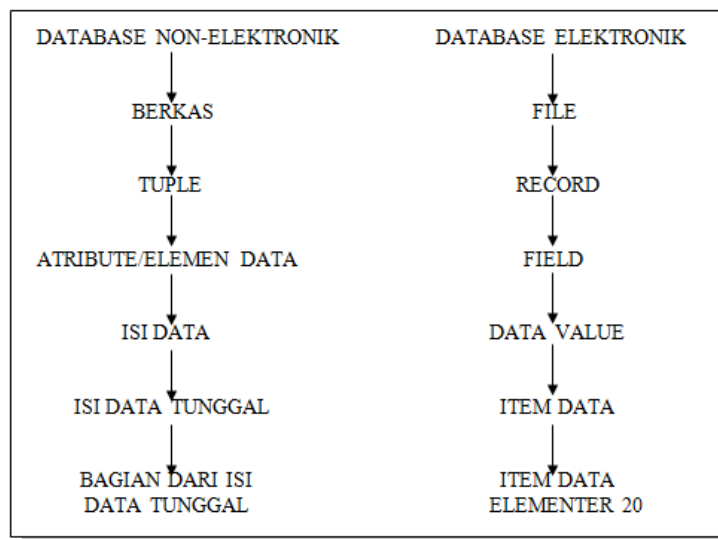
- b. *Entity* (entitas), suatu objek yang dapat dibedakan dengan objek lainnya.
Contoh :
Bidang administrasi siswa ---> entitas mahasiswa, buku, pembayaran
Bidang kesehatan ---> entitas pasien, dokter, Obat
- c. *Attribute / field* , setiap entitas mempunyai atribut atau suatu sebutan untuk mewakili suatu entitas.
Contoh :
Entity siswa --- > *field* Nim, nama_siswa, alamat
Entity nasabah --- > *field* Kode_nasabah, nama_nsh
- d. *Data value* (nilai atau isi data), *data actual* atau informasi yang disimpan pada tiap data elemen atau *attribute*. Isi dari atribut disebut nilai data.
Contoh :
Atribut nama karyawan --- > Sutrisno, budiman
- e. *Record/tuple*, kumpulan elemen-elemen yang saling berkaitan menginformasikan tentang suatu *entity* secara lengkap.
Contoh :
Satu *record* mewakili -- > nim, nm_mhs, alamat.
satu data/informasi
- f. *File*, kumpulan *record* sejenis yang mempunyai panjang elemen sama, *attribute* yang sama, namun berbeda-beda *data value*-nya.
- g. Kunci elemen data, tanda pengenal yang secara unik mengidentifikasikan entitas dari suatu kumpulan entitas.
- h. *Database Management System* (DBMS), kumpulan *file* yang saling berkaitan bersama dengan program untuk pengelolaannya.

BAB II

ARSITEKTUR BASIS DATA (*DATABASE*)

2.1. Hierarki *Database*

Penggunaan *database* dapat menggunakan 2 (dua) cara, yaitu *database* non elektronik dan *database* elektronik. Adapun urutan keduanya dapat dilihat pada gambar dibawah ini.



Gambar 2.1. Struktur Hierarki Basis Data

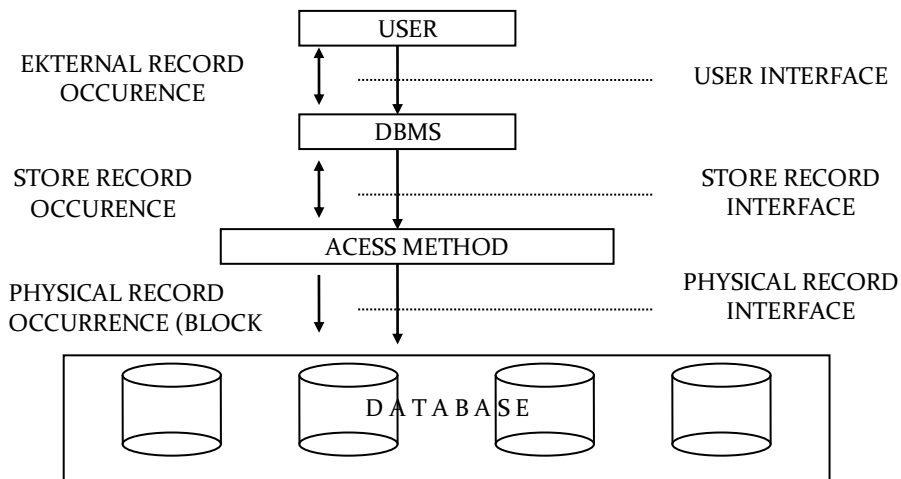
2.2. Tingkatan Arsitektur *Database*

Tingkatan arsitektur *database* yang digunakan dapat dibedakan menjadi 3 tingkatan, antara lain :

1. Internal level : “Menerangkan struktur penyimpanan basisdata secara fisik dan organisasi *file* yang digunakan”.
2. Konseptual level “Menerangkan secara menyeluruh dari basis data dengan menyembunyikan penyimpanan data secara fisik”.
3. Eksternal level: “Menerangkan *View* basis data dari sekelompok pemakai”.

2.3. Abstraksi Hubungan antara User pada DBMS dengan *Physical Database*

Penggunaan DBMS oleh *user*, dapat diilustrasikan dengan gambar dibawah ini.



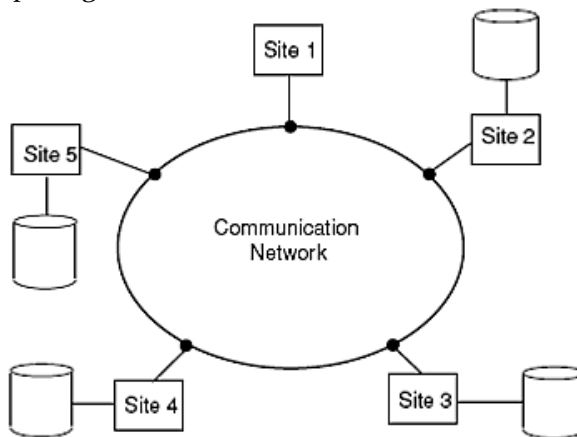
Gambar 2.2. Abstraksi Hubungan antara User pada DBMS dengan *Physical Database*

Pada gambar diatas, seorang *user* sebelum menggunakan basis data harus melalui tiga tahapan. Pertama kali seorang *user* akan menggunakan *user interface* untuk memanggil *external record* dari DBMS yang tersimpan di *storage* dengan menggunakan *store record interface*. Di dalam penyimpanan selanjutnya akan dihubungkan dengan basis data melalui *physical record interface*. Setelah terhubung dengan basis data, selanjutnya *record* dalam basis data tersebut dapat dieksekusi.

2.4. Arsitektur DBMS *Multi User*

Secara arsitektural, sebuah sistem basis data terdistribusi terdiri atas sebuah set *query sites* (kemungkinan besar kosong) dan sebuah set *data sites* yang tidak kosong. *Data sites* memiliki kemampuan untuk menyimpan data ketika set *query* tidak melakukannya. Yang kemudian hanya menjalankan *user interface* (sebagai tambahan dalam aplikasi) dengan tujuan untuk

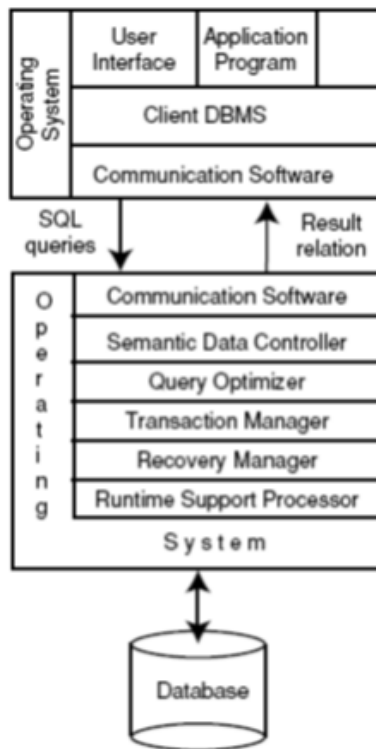
memberikan fasilitas akses pada *data sites*. Untuk lebih mudahnya, dapat dilihat pada gambar dibawah ini.



Gambar 2.2. Kebutuhan *database* terdistribusi

Terdapat beberapa macam model arsitektural untuk pengembangan sistem manajemen basis data terdistribusi, mulai dari sistem *client/server*, dimana *query sites* bersesuaian dengan *client* ketika *data sites* bersesuaian dengan *server*, untuk sistem *peer to peer* dimana tidak terdapat perbedaan antara mesin *client* dan mesin *server*, beberapa arsitektur ini berbeda berkenaan dengan dimana ketersediaan tempat untuk masing-masing fungsi DBMS.

Dalam kasus DBMS *client/server*, *server* akan melaksanakan lebih banyak pekerjaan mengenai manajemen data. Yang berarti bahwa seluruh proses terhadap *query* dan proses optimalisasi, manajemen transaksi dan manajemen penyimpanan diselesaikan pada *server*. Sedangkan *client*, merupakan tambahan untuk aplikasi dan antar muka untuk *user*. Terdapat sebuah modul DBMS *client* yang bertanggung jawab untuk mengelola data yang telah di *cached* pada *client* dan (kadang-kadang) mengelola kunci transaksi yang mungkin telah di *cached* juga. Sebuah standar fungsional dari distribusi *client/server* dapat dilihat pada gambar dibawah.



Gambar 2.3. Arsitektur Client/Server

Arsitektur *client/server* yang paling sederhana adalah sistem *multiple-client/single-server*. Dari sebuah perspektif manajemen data, hal ini tidak begitu berbeda dengan basis data terpusat sejak basis data disimpan hanya dalam satu mesin saja (*server*) dimana juga merupakan tempat untuk *software* yang digunakan untuk melakukan manajemen basis data tersebut. Bagaimanapun juga terdapat beberapa perbedaan penting dari sistem terpusat dalam bagaimana transaksi dieksekusi dan bagaimana *chaced* di kelola.

Sebuah arsitektur yang lebih canggih adalah dimana terdapat beberapa *server* di dalam sistem (yang kemudian disebut pendekatan *multiple-client/multiple-server*). Dalam kasus ini, terdapat dua alternatif strategi manajemen yang mungkin dilaksanakan: yaitu dimana masing-masing *client* DBMS mengelola koneksinya sendiri kepada *server* atau tiap *client* hanya mengenal *home server*-nya saja, dimana kemudian dibutuhkan komunikasi dengan *server* lain.

Pendekatan terdahulu adalah dengan melaksanakan penyederhanaan *code server*, namun membebani mesin *client* dengan beberapa tanggung jawab tambahan (*heavy client*) sedangkan pendekatan yang lain adalah dengan mengonsentrasikan kemampuan manajemen data secara fungsional di *server* dan kemudian menyediakan transparansi akses data pada *user interface* (*light client*).

Dalam kasus sistem *peer-to-peer*, tidak terdapat perbedaan antara *client* dan *server* dan masing-masing *site* dalam sistem dapat melaksanakan fungsi yang sama. Namun masing-masing dimungkinkan untuk memisahkan modul yang digunakan untuk melayani permintaan *user* dari yang lain untuk memanajemen data, namun ini hanya merupakan pemisahan secara logika dan sama sekali tidak menyiratkan distribusi fungsional.

Dalam proses eksekusi *query* (transaksi), ini memungkinkan bagi *query global optimizer* (monitor eksekusi global) untuk berkomunikasi secara langsung dengan prosesor *query* lokal (*local recovery managers*), dimana bagian-bagian dari *query* akan dieksekusi. Sehingga mekanisme komunikasi semakin dilibatkan, yang mendorong ke arah struktur *software* yang lebih rumit. Berikut karakteristik arsitektur sistem *Client/Server*, antara lain :

1. Arsitektur *client/server* terus dikembangkan dan terus dimasukkan dalam paket DBMS komersial – seperti halnya mereka bergerak terus untuk mendukung distribusi.
2. *Software* DBMS kemudian dibagi menjadi dua level yaitu *client* dan *server*.
3. Untuk menurunkan kompleksitasnya, dapat dilakukan langkah sebagai berikut:
 - a. Beberapa *site* dapat menjalankan hanya *software client* saja.
 - b. Sites yang lain dapat digunakan sebagai mesin *server* yang hanya akan menjalankan *software server* saja, dimana *site* yang lain dapat mendukung kedua modul *server* dan *client*.

Sedangkan fungsi sistem *Client/Server* terdiri dari :

1. *Software client* dan *server* saling berkomunikasi dengan menggunakan *Structure Query Language* (SQL).

2. Server SQL bertanggung jawab atas manajemen data lokal dalam sebuah *site*, seperti halnya DBMS terpusat
3. Fungsi *client* SQL juga diperluas sesuai dengan kebutuhan

2.5. Karakteristik *Client-Server*

Karakteristik basis data dengan *client* dan *server* dapat dibedakan menjadi :

1. *Client*
 - a. Menyediakan antar muka untuk *user*,
 - b. Menyediakan format *query* atau perintah dalam bahasa yang telah dikenal.
 - c. Mengomunikasikan format *query* dan perintah dengan *server* yang disesuaikan dengan metode komunikasi antar proses yang diterima.
 - d. Melaksanakan analisis terhadap data yang merupakan hasil yang dikembalikan oleh *server*.
 - e. Menampilkan hasil *query* dan perintah kepada *user*.
2. *Server*
 - a. Menyediakan pelayanan pada *client* (bisa lebih dari satu *client*).
 - b. Hanya merespons *query* atau perintah yang dikirimkan oleh *client*, tidak memulai komunikasi dengan *client*.
 - c. Secara ideal akan menyembunyikan keberadaan dari sistem *client-server* dari *client*

2.6. Keuntungan dan Kelemahan Sistem *Client-Server*

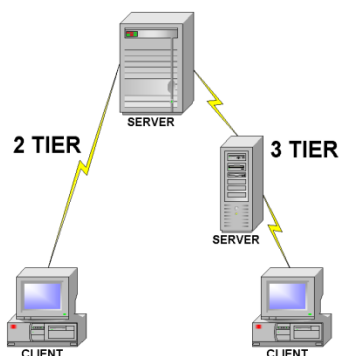
Penggunaan basis data dengan sistem *client* dan *server* mempunyai keuntungan dan kelemahan, yang terdiri dari :

1. Keuntungan
 - a. Efisiensi jumlah pekerjaan.
 - b. *Client* mengakses pada *remote* data (melalui standar),
 - c. Menyediakan fungsi DBMS secara penuh pada mesin *client*.
 - d. Pengukuran *resource* secara horizontal dan vertikal.
 - e. Harga dan performa yang lebih baik pada mesin *client*.

- f. Kemampuan untuk menggunakan *tool* yang lebih familiar dengan *user* di *client*.
 - g. Secara keseluruhan menyediakan performa dan harga yang lebih baik.
2. Kelemahan
- a. *Server* membentuk sebuah titik kegagalan tunggal terhadap semua akses.
 - b. Kesulitan dalam pengukuran basis data.

2.7. Arsitektur Dua dan Tiga-Tier

Arsitektur basis data dengan *client-server*, bisa dibuat dengan menggunakan dua atau tiga tier (gambar 2.4). Pembuatan dengan dua, tiga atau lebih tier disesuaikan dengan kebutuhan dan kemampuan dari setiap *user*.



Gambar 2.4. Arsitektur *Client-Server* model 2 (dua) tingkat dan 3 (tiga) tingkat

Adapun ciri arsitektur dua atau tiga tier adalah sebagai berikut :

1. Arsitektur ini memiliki sebuah *data management layer*, sebuah *application layer* dan sebuah *user interface layer*.
2. *Data management layer* memegang kendali atas skema basis data dan data.
3. Sebuah *application layer* memegang program yang mewujudkan *application logic*.
4. Sebuah *user interface layer* memanajemen *form* dan laporan yang akan dipresentasikan pada *user*.

5. CORBA/DCOM dapat digunakan untuk mendukung arsitektur tiga-tier.

2.8. Pendekatan Lain *Client-Server*

Diatas telah dibahas arsitektur *client-server*, namun pendekatan lain dari arsitektur *client-server* dapat dilakukan antara lain :

1. Program yang belum di-*compile* tersimpan dalam *site server*, kemudian akan dilibatkan dalam proses setelah adanya *remote procedure call* yang dijalankan oleh *client*
2. Beberapa keuntungan dari pendekatan ini, antara lain :
 - a. Tingkat independensi data yang lebih tinggi, sehingga dapat digunakan untuk menyembunyikan banyak detail spesifikasi dari sistem.
 - b. Sebuah prosedur yang telah tersimpan dapat digunakan secara bersama-sama oleh banyak *client*.
 - c. Optimalisasi dinyatakan selesai pada saat tingkatan kompilasi telah menyediakan keamanan yang lebih baik.

2.9. Contoh Proses dari Sistem *Client-Server*

Banyak poses yang dapat dilakukan pada sistem *client-server*, contoh proses dari sistem *client-server*, sebagai berikut :

1. Interaksi antara *client* dan *server* akan terjadi seperti halnya pada saat pemrosesan sebuah *query* pada SQL.
2. *Client* menguraikan *query* dari *user* dan kemudian memilah-milah *query* tersebut menjadi beberapa *query* lokal pada *site independent*.
3. Setiap *query* akan dikirimkan pada *site server* yang sesuai.
4. Setiap *server* memproses *query* lokal dan kemudian mengirimkan hasilnya ke *site client*.
5. *Site client* kemudian akan mengombinasikan hasil dari *subqueries-subqueries* untuk membentuk sebuah hasil atau laporan dari *query* yang dimaksudkan oleh *user*.

2.10. Pengguna (User) Basis Data

Penggunaan basis data dibatasi oleh penggunanya. Untuk menggunakan basis data akan terdiri dari beberapa tingkatan, yaitu :

1. *Database Manager*, orang yang bertugas mengelola *database*, dengan memberikan batasan tiap pengguna *database*. Adapun tugasnya antara lain :
 - a. Berinteraksi dengan *file manager*.
 - b. Menjaga integritas dan keamanan data.
 - c. *Backup* dan *recovery*.
 - d. Mengendalikan konsistensi dan konkurensi data.
2. *Database Administrator*, orang yang bertugas untuk mengelola isi dari *database*, dengan mengubah isi, lokasi dan organisasi *database* tanpa mengganggu program aplikasi yang ada. Sedangkan tugasnya antara lain :
 - a. Mendefinisikan skema *database*.
 - b. Mendefinisikan metode untuk mengakses data dan
 - c. Struktur penyimpanan.
 - d. Modifikasi fisik penyimpanan dan skema data.
 - e. Memberikan otorisasi hak akses data (*granting*).
 - f. Menentukan *integritas constraints*.
3. *Database User*, orang yang menggunakan *database* untuk mengisi atau hanya melihat saja, terdiri dari :
 - a. *Applications Programmer*.
 - b. *Specialized Users*.
 - c. Dan lain-lain

2.11. Bahasa Pemrograman Basis Data

Penggunaan bahasa dalam DBMS dibagi menjadi beberapa definisi yang masing-masing mempunyai spesifikasi sendiri-sendiri, yaitu:

1. *Data Definition Language* (DDL)

Paket bahasa yang merumuskan tentang apa dan bagaimana suatu *database* dibentuk. Hasil kompilasi dari perintah DDL adalah

satu set dari tabel yang disimpan dalam file khusus disebut *data dictionary/directory*.

DDL dalam bahasa SQL Server dapat menentukan tata letak baris, definisi kolom, pembuatan kolom kunci, pembuatan lokasi *file* dan strategi penyimpanan. Dengan DDL anda juga dapat mendefinisikan basis data, tabel dan *view*. Untuk setiap objek, biasanya ada pernyataan-pernyataan *CREATE*, *ALTER*, dan *DROP*. Bentuk umum pernyataan-pernyataan adalah :

```
CREATE nama_objek  
ALTER nama_objek  
DROP nama_objek
```

Contoh:

```
CREATE DATABASE Karyawan  
ON  
( NAME = Person_dat,  
FILENAME = 'F:\Data Kantor\Person.mdf',  
SIZE = 5,  
MAXSIZE = 50,  
FILEGROWTH = 1)
```

```
USE Karyawan  
CREATE TABLE Tunjangan  
( NIP Char(10) NOT NULL,  
Tunj_Istri Money NOT NULL DEFAULT o,  
Tunj_Anak Money NOT NULL DEFAULT o,  
Tunj_Sehat Money NOT NULL DEFAULT o)  
(Buat database dan tabel baru)  
USE Karyawan  
ALTER TABLE Tunjangan  
ADD Tunj_Transp Money NOT NULL DEFAULT o  
(Sisipkan field baru)
```

```
USE Karyawan  
DROP TABLE Tunjangan  
(Hapus tabel)
```

2. *Data Control Language (DCL)*

DCL adalah sebuah skema basis data yang digunakan untuk mengatur hak-hak pada objek basis data. Dengan DCL inilah kita

dapat membuat perintah-perintah yang akan digunakan untuk pengaturan hak, seperti GRANT dan REVOKE. Pada T-SQL dapat ditambahkan pernyataannya dengan DENY.

Perintah GRANT digunakan untuk memberikan hak kepada *user* untuk mengakses sebuah basis data. Misalnya perintah untuk memberikan izin menjalankan perintah SELECT pada tabel Tunjangan dalam basis data Karyawan pada role PUBLIC.

Contoh :

```
USE Karyawan
GRANT SELECT
ON Tunjangan
TO PUBLIC
```

```
USE Karyawan
REVOKE SELECT
ON Tunjangan
TO PUBLIC
```

Perintah REVOKE digunakan untuk membuang hak yang telah diberikan (dengan perintah GRANT) atau hak yang dilarang (dengan perintah DENY)

Contoh lain dari beberapa DCL :

a. INSERT

Sintaks : INSERT INTO

Nama_table [(nama_kolom₁,...)]

Contoh :

Masukan data matakuliah berkas akses dengan kode KK222, "Berkas Akses" dan besarnya 2

```
INSERT INTO MKUL VALUES("KK222","Berkas Akses", 2);
```

b. UPDATE

Sintaks : UPDATE nama_table

SET nama_kolom = ekspresi

WHERE kondisi ;

Contoh :

Ubah alamat menjadi "Depok" untuk mahasiswa yang memiliki NPM "50096487"

```
UPDATE MHS
SET ALAMAT="Depok"
WHERE NPM="50096487";
```

c. DELETE

Sintaks : DELETE FROM nama_table
WHERE kondisi

Contoh :

Hapus data nilai matakuliah "KKo21" bagi mahasiswa yang mempunyai NPM "10296832"

```
DELETE FROM NILAI WHERE
NPM="10296832" AND KDMK="KKo21"
```

d. SELECT

Sintaks : SELECT [DISTINCT] nama_kolom
FROM nama_table
WHERE kondisi]
[GROUP BY nama_kolom]
[HAVING kondisi]
[ORDER BY nama_kolom [ASC/DESC]]

Contoh : Tampilkan semua data mahasiswa

```
SELECT NPM, NAMA, ALAMAT
FROM MHS
```

Atau

```
SELECT * FROM MHS
```

Tampilkan Mata Kuliah yang SKSnya 2 maka penulisannya:

```
Select namaMK from matakuliah
```

```
Where sks = 2
```

Tampilkan semua data nilai dimana nilai MID lebih besar sama dengan 60 atau nilai finalnya lebih besar 75. maka penulisannya :

```
SELECT *FROM NILAI
WHERE MID >= 60 OR FINAL > 75
```

3. *Data Manipulation Language* (DML)

Sedangkan DML adalah bahasa yang digunakan oleh pengguna (*user*) untuk mengakses atau memanipulasi data sebagai pengatur yang cocok oleh model data. Jadi tujuan dari DML adalah memudahkan pengguna untuk mengambil data dari basis data dan kemudian mengolah atau memanipulasi data yang telah diambil tersebut. Manipulasi data yang dilakukan pada DML terdiri dari :

Mendapatkan kembali (*retrieval*) penyimpanan informasi dalam basis data.

- Penghapusan (*deletion*) informasi dari basis data.
- Penyisipan (*insertion*) informasi baru dalam basis data.
- Pengubahan (*modification*) penyimpanan data dalam basis data.
- Memperbaharui (*update*) data dalam basis data.

```
SELECT nama_field_yang_akan_ditampilkan  
[ INTO nama_tabel_baru ]  
[ FROM tabel_sumber ]  
[ WHERE kondisi_pencarian ]  
[ GROUP BY daftar_grup ]  
[ HAVING kondisi_pencarian_yang_sesuai ]  
[ ORDER BY daftar_index [ ASC | DEC ] ]
```

Keterangan :

ASC = Ascending

DEC = Decending

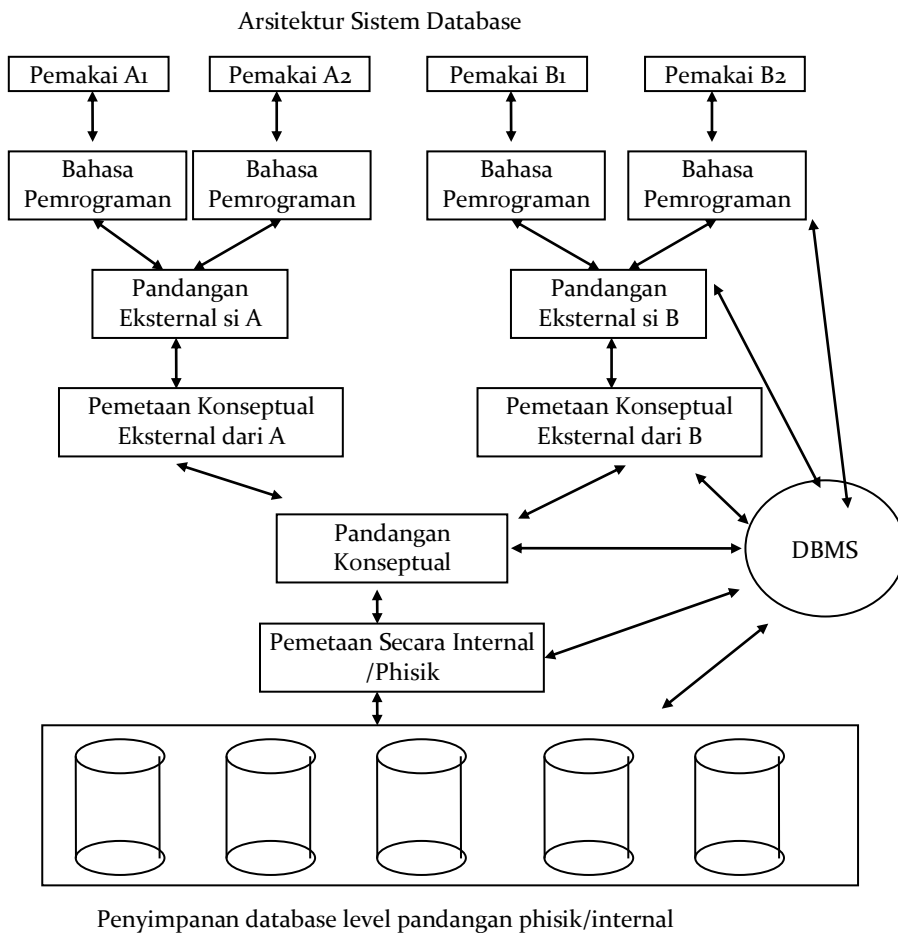
Nama_field_yang_akan_ditampilkan dapat diganti dengan tanda (*), apabila akan menampilkan semua *field* yang ada ditabel. Sedangkan untuk memilih *field* tabel, dapat digunakan tanda koma (,) untuk memisahkan antara *field* satu dengan yang lain.

Contoh :

```
SELECT a.OrderID, a.ProductID, b.ProductName,  
       b.UnitPrice, a.Quantity, a.Discount  
FROM "Order details" a, Products b  
WHERE a.ProductID = b.ProductID
```


4. Query

Paket bahasa yang merupakan bagian dari DML yang digunakan untuk pengambilan informasi sesuai kebutuhan *database* yang dibentuk.



Gambar 2.5. Arsitektur Sistem Database

BAB III

DATABASE MANAGEMENT SYSTEM (DBMS)

3.1. Definisi DBMS

DBMS adalah perangkat lunak yang menangani semua pengaksesan *database*. Dengan DBMS, diharapkan basis data dapat dikelola dengan baik dan mudah dalam penggunaannya. DBMS mempunyai fungsi antara lain :

1. *Data Definition*, DBMS harus dapat mengolah dan pendefinisian data.
2. *Data Manipulation*, DBMS harus dapat menangani permintaan dari pemakai untuk mengakses data.
3. *Data Security and Integrity*, DBMS harus dapat memeriksa keamanan dan integritas data yang didefinisikan oleh DBA. Untuk melaksanakannya, dapat dilakukan sebagai berikut :
 - a. *Data Recovery and Concurrency*, DBMS harus dapat menangani kegagalan – kegagalan pengaksesan *database* yang dapat disebabkan oleh kesalahan sistem, kerusakan disk, dsb.
 - b. *Data Dictionary*, DBMS harus menyediakan kamus data yang berfungsi untuk identifikasi isi data pada sebuah *database*. Penambahan kamus data ini bertujuan untuk memudahkan pembacaan oleh sistem analisis lain atau *user* terhadap data apa saja yang terdapat pada suatu sistem, baik dengan menggunakan DFD maupun ER-D. Untuk pembuatan kamus data, perlu diperhatikan notasi apa saja yang digunakan.

3.2. Komponen DBMS

Dalam penggunaan DBMS, dibutuhkan komponen-komponen, antara lain :

1. *Query Prosesor*, komponen yang mengubah bentuk *query* dalam bentuk instruksi kedalam *database manager*.
2. *Database Manager*, menerima *query*, menguji eksternal dan konseptual untuk menentukan apakah *record – record* tersebut

dibutuhkan untuk memenuhi permintaan kemudian hari dari *database manager* dengan memanggil *file manager* untuk menyelesaikan permintaan

3. *File manager*, memanipulasi penyimpanan *file* dan mengatur alokasi ruang penyimpanan *disk*.
4. *DML Prosessor*, modul yang mengubah perintah DML yang ditempelkan kedalam program aplikasi dalam bentuk fungsi-fungsi.
5. *DDL compiler*, mengubah *statement* DDL menjadi kumpulan tabel atau *file* yang berisi *data dictionary* atau meta data.
6. *Dictionary manajer*, mengatur akses dan memelihara *data dictionary*.

3.3. Data Independence

Merupakan salah satu kelebihan sistem *database* dimana DBA dapat mengubah struktur *storage* dan strategi akses dalam pengembangan sistem *database* tanpa mengganggu program-program aplikasi yang sudah ada. *Data independence* terbagi menjadi 2 (dua) tingkatan, yaitu:

1. *Physical data independence*, yaitu perubahan *internal schema* dapat dilakukan tanpa mengganggu *conceptual schema*.
2. *Logical data independence*, yaitu *Conceptual schema* dapat dirubah tanpa mempengaruhi *eksternal schema*.

Pembuatan *data independence* diperlukan untuk diterapkan pada pengelolaan sistem basis data, disebabkan antara lain :

1. *Database Administrator* dapat mengubah isi, lokasi dan organisasi *database* tanpa mengganggu program aplikasi yang ada.
2. *Vendor hardware* dan *software* pengelolaan data bisa memperkenalkan produk - produk baru tanpa mengganggu program - program aplikasi yang telah ada.
3. Untuk memudahkan perkembangan program aplikasi.
4. Memberikan fasilitas pengontrolan terpusat oleh DBA demi keamanan dan integritas data, dengan memperhatikan perubahan - perubahan kebutuhan *user*.

BAB IV

MODEL DATA

4.1. Pengertian Data model

Data model adalah sekumpulan konsep-konsep untuk menerangkan data, hubungan-hubungan antara data dan batasan-batasan data yang terintegrasi di dalam suatu organisasi. Dengan pembuatan model data, akan didapatkan bentuk data apa saja yang akan dibuat sesuai dengan kebutuhan.

4.2. Jenis-jenis Data model

Model data yang digunakan untuk perancangan basis data terdiri dari :

1. Model data berbasis objek

Model data berbasis objek menggunakan konsep entitas, atribut dan hubungan antar entitas. Adapun model data berbasis objek terdiri dari :

a. *Entity Relationship Model*

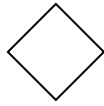
Model untuk menjelaskan hubungan antar data dalam basis data berdasarkan suatu persepsi bahwa *real word* terdiri dari objek-objek dasar yang mempunyai hubungan atau relasi antara objek-objek tersebut. Adapun yang dimaksud *real word* adalah keseluruhan data yang belum terstruktur yang secara nyata ada/terkait dalam sebuah lingkup topik yang ditinjau.

E-R Model berisi ketentuan atau aturan khusus yang harus dipenuhi oleh isi *database*. Aturan terpenting adalah *Mapping Cardinalities*, yang menentukan jumlah *entity* yang dapat dikaitkan dengan *entity* lainnya melalui *relationship set*. Simbol yang digunakan terdiri dari :

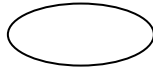
- *Entity* / entitas : menunjukkan objek dasar



- *Relationship* / relasi : Menunjukkan relasi



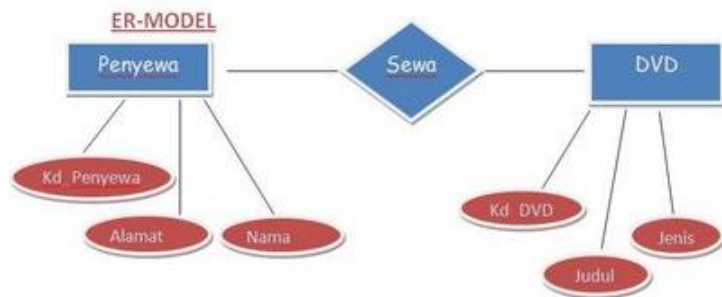
- *Attribute* / atribut : menunjukkan atribut dari objek dasar



- *Line* / garis : menunjukkan adanya relasi



Contoh gambar ER model, dapat dilihat pada gambar dibawah :



Gambar.4.1. Contoh kasus ER-Model

b. Binary Model

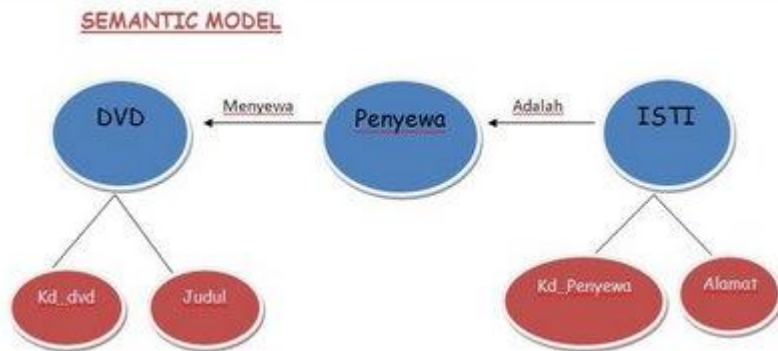
Binary model adalah suatu bentuk model dalam bentuk biner (angka 1 dan 0). Berfungsi untuk mengidentifikasi keselarasan model data yang akan digunakan agar bersinergi dengan data komputer.

c. Semantic Data Model

Hampir sama dengan *Entity Relationship* model dimana relasi antara objek dasar tidak dinyatakan dengan simbol tetapi menggunakan kata-kata (Semantik). Sebagai contoh, dengan masih menggunakan relasi pada Bank X sebagaimana contoh sebelumnya, dalam *semantic model* adalah seperti terlihat pada gambar di atas.

Tanda-tanda yang menggunakan dalam semantic model adalah sebagai berikut :

—————→ : Menunjukkan adanya hubungan
 ————— : Menunjukkan atribut



Gambar 4.2. Contoh Kasus *Semantic Model*

d. *Infological Model*

Suatu model yang menggambarkan informasi dalam bentuk *logical*. Model ini akan menggambarkan informasi yang sesungguhnya secara jelas.

2. Model data berbasis *record*

Model ini berdasarkan pada *record* untuk menjelaskan kepada *user* tentang hubungan *logic* antar data dalam basis data. Model logika berbasis *record* digunakan untuk menggambarkan data pada tingkat konseptual dan *view*. Model data ini bersama dengan model data logika berbasis objek biasanya digunakan untuk menyatakan struktur logika *database* secara keseluruhan. Selain itu juga digunakan untuk mendeskripsikan bagaimana gambaran penerapannya dalam tingkat yang lebih tinggi dari gambaran fisiknya.

Struktur database pada model logika berbasis *record* ini dinyatakan dengan tipe *record* yang mempunyai format tetap. Artinya setiap tipe *record* mempunyai beberapa *field* atau atribut dengan jumlah tetap, dan setiap *field* mempunyai panjang yang tetap. Tiga model data pada kelompok ini yang telah diterima secara meluas adalah model data relasi, jaringan (*network*) dan hierarki. Berikut adalah penjelasan singkat ketiga model data ini.

Tabel SALDO

| No_Rek | Nm_Nasabah | Alm_Nasabah | Saldo |
|----------|------------|-------------|--------|
| 92777001 | Nasabah A | JL A | 10.000 |
| 92777002 | Nasabah B | JL B | 15.000 |
| 92777003 | Nasabah C | JL C | 16.000 |
| 92777004 | Nasabah D | JL D | 75.000 |
| 92777005 | Nasabah E | JL E | 40.000 |
| 92777006 | Nasabah F | JL F | 46.000 |

Tabel TABUNGAN

| No_Rek | Nm_Nasabah | Alm_Nasabah | Tabungan |
|----------|------------|-------------|----------|
| 92777001 | Nasabah A | JL A | 10.000 |
| 92777002 | Nasabah B | JL B | 15.000 |
| 92777003 | Nasabah C | JL C | 16.000 |
| 92777006 | Nasabah F | JL F | 16.000 |

Tabel DEPOSITO

| No_Rek | Nm_Nasabah | Alm_Nasabah | Deposito | Bunga (%) | Jth_Tempo |
|----------|------------|-------------|----------|-----------|------------|
| 92777004 | Nasabah D | JL D | 75.000 | 10 | 12/10/2001 |
| 92777005 | Nasabah E | JL E | 40.000 | 15 | 15/05/2002 |
| 92777006 | Nasabah F | JL F | 46.000 | 20 | 13/09/2003 |

Tabel REKENING

| No_Rek | Nm_Nasabah | Alm_Nasabah | Saldo | Tabungan | Deposito | Bunga (%) | Jth_Tempo |
|----------|------------|-------------|--------|----------|----------|-----------|------------|
| 92777001 | Nasabah A | JL A | 10.000 | 10.000 | 0 | 0 | 0 |
| 92777002 | Nasabah B | JL B | 15.000 | 15.000 | 0 | 0 | 0 |
| 92777003 | Nasabah C | JL C | 16.000 | 16.000 | 0 | 0 | 0 |
| 92777004 | Nasabah D | JL D | 75.000 | 0 | 75.000 | 10 | 12/10/2001 |
| 92777005 | Nasabah E | JL E | 40.000 | 0 | 40.000 | 15 | 15/05/2002 |
| 92777006 | Nasabah F | JL F | 46.000 | 16.000 | 30.000 | 20 | 13/09/2003 |

Gambar 4.3. Gambar model data berbasis record

3. Model data fisik

Tingkatan ini merupakan tingkatan terendah dalam abstraksi data yang menunjukkan bagaimana data secara aktual disimpan. Pada tingkatan fisik ini, akan menggambarkan tingkat struktur data hingga tingkat dasar yang kompleks atau rumit. Pada aplikasi rutin, pengguna akan berurusan dengan data, seperti teks yang terdiri dari angka-angka hingga tampilan besarnya penyimpanan (*bit*).

4. Tingkatan konseptual (*conceptual level*)

Tingkatan berikutnya pada abstraksi data adalah tingkatan konseptual. Tingkatan ini menggambarkan abstraksi bagaimana semua penyimpanan data secara aktual dalam basis data. Dimana seluruh basis data akan digambarkan kedalam bagian kecil struktur relasi yang sederhana. Tingkatan konseptual akan menggambarkan pemisahan penggunaan basis data yang dilakukan oleh pengatur

/pengelola basis data (*database administrator*). Pada kegiatan rutin, seorang pengelola penggunaan *file*/tabel dalam suatu basis data, misalnya pengaturan *file* pegawai, *file* pelanggan dan sebagainya.

5. Tingkatan tampilan (*view level*)

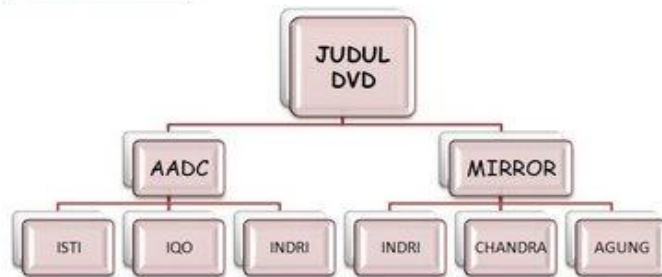
Tingkatan yang paling tinggi pada abstraksi data adalah tingkatan tampilan. Pada tingkatan ini, hanya akan menggambarkan pemisahan sebagian dari seluruh basis data. Banyak pengguna sistem basis data tidak memperhatikan semua yang terdapat pada informasi, karena kebanyakan pengguna hanya akan membutuhkan sebagai data atau informasi yang hanya ditampilkan saja. Mungkin saja sistem akan menampilkan banyak tampilan untuk basis data yang sama, namun itu semua hanya tampilannya saja tanpa pengguna mengetahui dari mana asal data tersebut. Pada tahapan ini pula akan digunakan sistem untuk mengonversikan data yang asli ke data yang lebih bermakna. Misalnya tampilan data dengan kode asli (1/2) akan dikonversikan menjadi kode yang lebih bermakna (pria / wanita). Selain itu dalam tingkatan ini pula dapat ditampilkan dalam bentuk suara atau gambar

6. Konsep data hierarki

Dimana data serta hubungan antar data direpresentasikan dengan *record* dan *link* (*pointer*), dimana *record-record* tersebut disusun dalam bentuk *tree* (pohon), dan masing-masing *node* pada *tree* tersebut merupakan *record/grup* data elemen

Model data hierarki mempunyai kesamaan dengan model jaringan dalam hal representasi data dan hubungan diantaranya, yaitu dengan *record-record* dan *links*. Berbeda dengan model data jaringan, *record-record* dan *links* tersebut dalam *database* diorganisasikan sebagai kumpulan pohon (*tree*).

MODEL HIRARKI



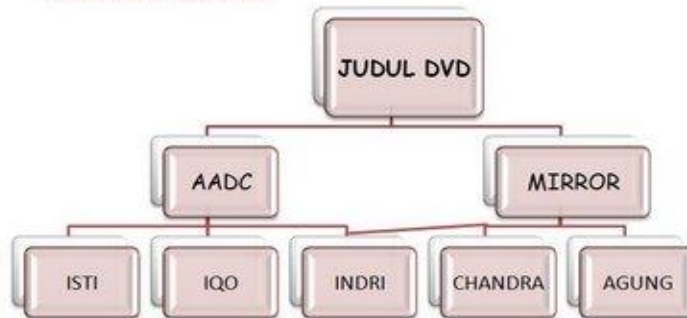
Gambar 4.4. Gambar model data berbasis hirarki

7. Konsep data jaringan

Mirip dengan *hierarchical model*, dimana data dan hubungan antar data direpresentasikan dengan *record* dan *links*. Perbedaannya terletak pada susunan *record* dan *link*-nya, yaitu *network model* menyusun *record-record* dalam bentuk *graph*.

Model data jaringan terdiri dari kumpulan *record* yang dihubungkan satu sama lainnya melalui *link*. Setiap *record* pada kumpulan *record* tersusun dari sekumpulan *field* atau atribut dimana masing-masing *field* atau atribut tersebut berisi hanya satu nilai data. Sebagai gambaran, tinjau sebuah *database* yang berhubungan dengan masalah akademik. Misalkan pada *database* tersebut diketahui bahwa dosen dengan nama Abdul mengajar kuliah Algoritma, Markum mengajar kuliah Struktur Data dan *Database*, dan Kumkum mengajar kuliah Matematika, maka penggambaran model data jaringannya bisa dinyatakan sebagai berikut:

MODEL JARINGAN



Gambar 4.5. Gambar model data berbasis jaringan

8. Model data Relasional

Model data relasional digunakan untuk menghubungkan antar tabel, dengan berpedoman kepada kunci-kunci yang akan digunakan sebagai penghubung.

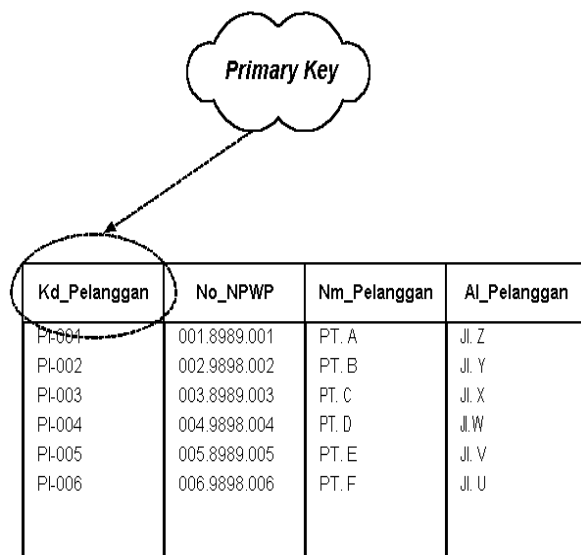


Gambar 4.6. Gambar model data relasional

Kunci (key) relasional

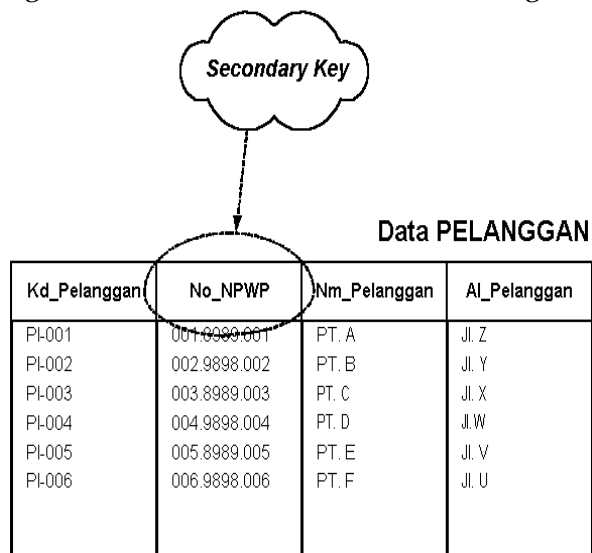
Kunci (*key*) sebagai penghubung dengan tabel lain dan kunci dapat digunakan untuk membedakan relasi yang terjadi antar data pada suatu basis data. Menurut Whitten et. al (2001:473) :

- *Primary Key* (Kunci Primer)
Suatu *field* yang hanya mengidentifikasikan satu nilai dan hanya satu *record* dalam sebuah *file*. Dalam suatu program sering disebut dengan kunci unik (*unique key*).



Gambar 4.7. Gambar contoh *primary key* pada tabel

- **Secondary Key (kunci Sekunder)**
 Suatu kunci sekunder adalah kunci yang mengidentifikasi kunci alternatif pada suatu basis data. Akan digunakan bila kunci Primer tidak berfungsi.



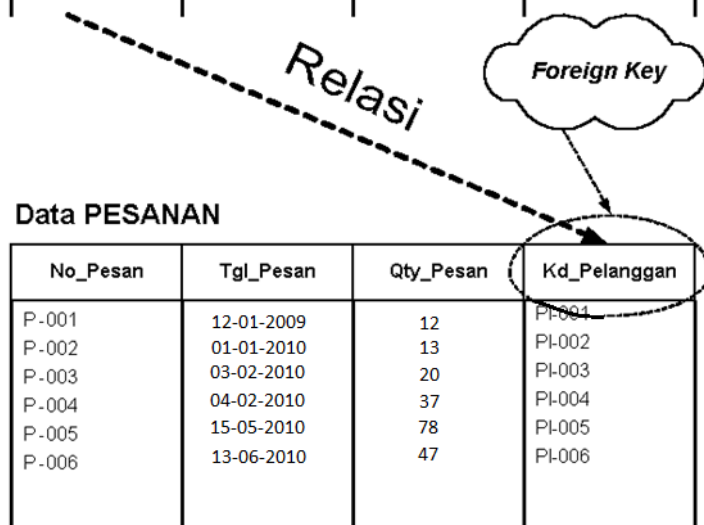
Gambar 4.8. Gambar contoh *Secondary key* pada tabel

- *Foreign Key* (kunci Asing)

Penunjukan *record* yang terdapat pada *file* yang berbeda pada suatu basis data. Penggunaan kunci asing ini dimaksudkan sebagai media penghubung (*link*) *record* basis data dari satu data ke data yang lain dan biasanya digunakan pada saat terjadi relasi (*relationship*) data

Data PELANGGAN

| Kd_Pelanggan | No_NPWP | Nm_Pelanggan | Al_Pelanggan |
|--------------|--------------|--------------|--------------|
| PI-001 | 001.8989.001 | PT. A | Jl. Z |
| PI-002 | 002.9898.002 | PT. B | Jl. Y |
| PI-003 | 003.8989.003 | PT. C | Jl. X |
| PI-004 | 004.9898.004 | PT. D | Jl. W |
| PI-005 | 005.8989.005 | PT. E | Jl. V |
| PI-006 | 006.9898.006 | PT. F | Jl. U |

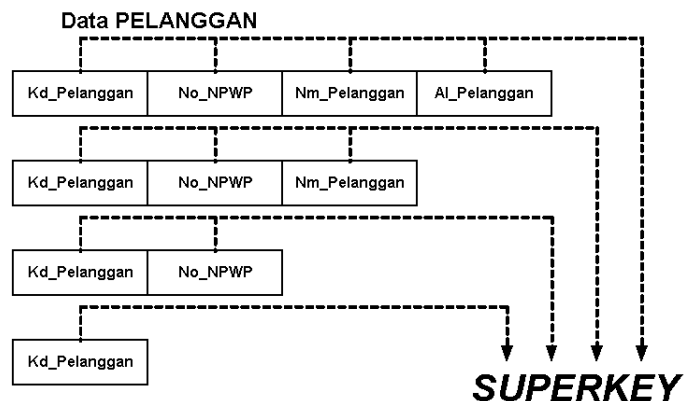


Gambar 4.9. Gambar contoh *Foreign key* pada tabel

Key (kunci) menurut Korth et. al. (1991:33), terdiri dari :

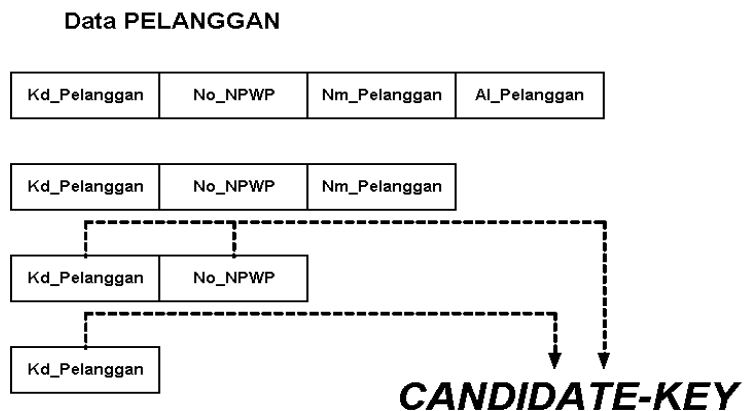
- *Superkey*

Satu atau lebih atribut dalam sebuah himpunan entitas, yang mana akan membentuk identifikasi yang unik dalam himpunan entitas.



Gambar 4.10. Gambar contoh Superkey pada field

- *Candidate-key*
 Sekumpulan minimal dari *superkey*. Jadi semua atribut *candidate-key* sudah pasti *superkey*, namun atribut *superkey* belum tentu *candidate-key*



Gambar 4.11. Gambar contoh Candidatekey pada field

Dalam menentukan *candidate-key*, harus diperhatikan bahwa *setiap record* dalam tabel harus bersifat unik (tidak rangkap)

4.3. Perbedaan dengan Objek Based Data Model

Pada *record based data model* di samping digunakan untuk menguraikan struktur logika keseluruhan dari suatu *database*, juga digunakan untuk menguraikan implementasi dari sistem *database* (*higher level description of implementation*). Terdapat 3 data model pada *record based data model*, yaitu :

a. Model Relational

Dimana data serta hubungan antar data direpresentasikan oleh sejumlah tabel, dan masing-masing tabel terdiri dari beberapa kolom yang namanya unik. Model ini berdasarkan notasi teori himpunan (*set theory*), yaitu *relation*.

Contoh : *database* penjual barang terdiri dari 3 tabel, yaitu :

- Supplier
- Path (Suku_cadang)
- Delivery (pengiriman)

b. Model Jaringan

Mirip dengan *hierarchical model*, dimana data dan hubungan antar data direpresentasikan dengan *record* dan *links*. Perbedaannya terletak pada susunan *record* dan *link*-nya yaitu *network model* menyusun *record-record* dalam bentuk graph.

c. Physical Data Model

Digunakan untuk menguraikan data pada internal level. Beberapa model yang umum digunakan:

- Unifying model
- Frame memory

BAB V

ENTITY RELATIONSHIP DIAGRAM

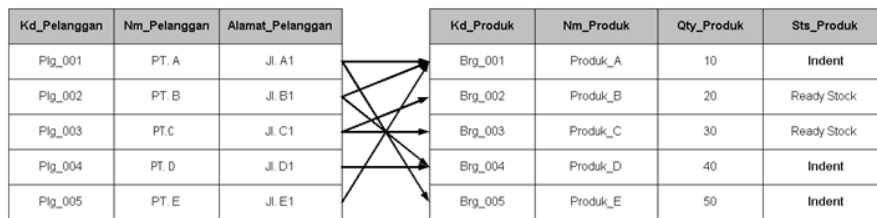
(ER-D)

Model data dengan diagram hubungan entitas (*entity relationship diagram / ER-D*) adalah suatu pemodelan berbasis pada persepsi dunia nyata yang mana terdiri dari kumpulan objek dasar yang disebut dengan entitas (*entity*) dan hubungan diantara objek-objek tersebut dengan menggunakan perangkat konseptual dalam bentuk diagram.

Sebelum menggambarkan ER-D, sebaiknya ditentukan tingkat derajat (*degree*) tiap entitas. Adapun yang dimaksud dengan derajat adalah ukuran yang lain secara kompleks pada hubungan data. Sebuah derajat pada hubungan entitas adalah nomor entitas yang berpartisipasi pada relasi atau hubungan.

Sebuah entitas adalah objek yang dibedakan dari objek yang lain oleh himpunan dari atribut (*attribute*). Misalnya sebuah entitas PRODUK yang dapat dibedakan dengan entitas PELANGGAN. Karena keduanya merupakan kelompok yang berbeda, sedangkan antara keduanya mempunyai ciri yang membedakan. Misalnya PRODUK mempunyai ciri Kd_Produk, Nm_Produk, Hrg_Produk dan Qty_Produk, dan PELANGGAN mempunyai ciri Kd_Pelanggan, Nm_Pelanggan, dan Alamat_Pelanggan.

Kedua atribut tersebut dapat dihubungkan karena adanya suatu kejadian, misalnya terjadi transaksi pesan produk. Untuk menghubungkan antara keduanya, dibutuhkan suatu alat yang disebut ER-D. Sehingga kedua atribut tersebut dapat terlihat hubungannya. Untuk menggambarkan tabelnya dapat dilihat pada gambar 5.1 dibawah ini :



Gambar 5.1. Hubungan antar entitas

Dari gambar diatas, terlihat adanya relasi antara tabel Plg_001 dengan tabel Brg_001 dan Brg_005. Relasi ini mengandung arti bahwa pelanggan Plg_001 memesan barang dengan kode Brg_001 dan Brg_005, demikian seterusnya.

5.1. Simbol yang Digunakan pada ER-D

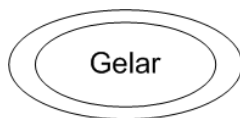
Simbol yang digunakan, terdiri dari :



- Entitas, Empat persegi panjang (*rectangle*) yang mewakili sekumpulan / himpunan objek yang berada pada sebuah sistem.

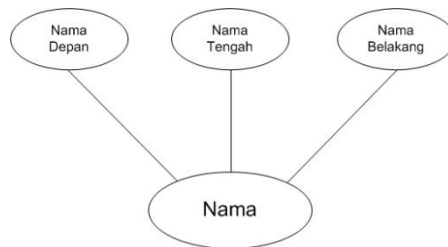


- Elips* yang mewakili atribut biasa. Pada beberapa kasus, penggunaan simbol *elips* dapat diganti dengan titik (.) (lihat gambar 7.44). Hal ini diperbolehkan, untuk mengatasi keterbatasan tempat penulisan.

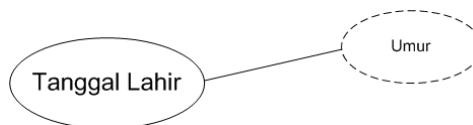


- Double Elips* yang mewakili atribut *multi key* atau *multi value*. Pada beberapa kasus, penggunaan simbol *double elips* dapat digunakan untuk mewakili sekelompok nilai yang digunakan pada setiap *instant entity*. Contoh penggunaannya adalah gelar yang terdiri dari S₁, S₂, S₃. Untuk isi atribut dibatasi hanya 3 isi

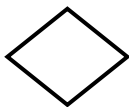
tersebut. Bisa juga untuk agama yang sudah ditentukan oleh negara.



- *Elips* yang menjadi turunan atau penjelasan atribut biasa, yaitu suatu atribut yang terdiri dari beberapa atribut yang lebih kecil yang mempunyai arti tertentu dan lebih spesifik dalam menjabarkan atribut yang lain. Simbol ini dikenal dengan atribut *composite*.



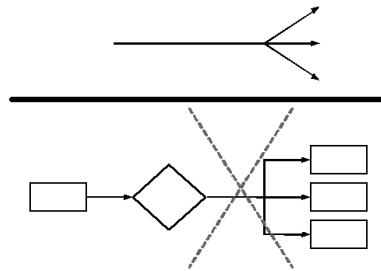
- *Elips* putus-putus yang mewakili atribut *derivative*, yaitu suatu atribut yang dihasilkan dari atribut yang lain. Jadi atribut umur sangat bergantung sekali dengan atribut tanggal lahir. Apabila atribut tanggal lahir dikosongkan, maka atribut umur akan kosong juga, demikian sebaliknya jika atribut tanggal lahir terisi dengan angka maka secara otomatis atribut umur akan terisi dengan tanggal sekarang dikurangi tanggal lahir.



- Intan (*diamond*)/belah ketupat yang mewakili hubungan antar himpunan entitas. Dalam pemberian keterangan hubungan, sebaiknya menggunakan kata kerja. Misalnya keluar, daftar, kerja dan sebagainya.
- Garis (*line*) yang mewakili hubungan antara atribut (*elips*) dengan entitas (*rectangle*) dan himpunan entitas (*diamond*) dengan entitas (*rectangle*) dan sebaliknya

5.2. Hal yang Dilarang dalam Pembuatan ER-D

Dalam ER-D ada hal yang dilarang dalam pembuatan. Hal yang dilarang adalah adanya pencabangan garis (*line*) dari dan ke *relationship* (*elips*) ke entitas (*rectangle*) atau sebaliknya. Pelarangan ini bertujuan untuk menghindari kerancuan dalam pembacaan data hasil hubungan antar entitas.



Gambar 5.2. Penulisan ER-D yang dilarang

5.3. Pembatasan Pemetaan

Sejak dibuat ER-D, tiap-tiap entitas akan dihubungkan dengan entitas lain harus dibatasi nomor/atribut minimal dan maksimal kejadian pada satu entitas. Kejadian ini dapat memperlihatkan kemungkinan yang menjadi hubungan antar entitas.

Dari sejumlah kemungkinan banyaknya hubungan antar entitas, maka dibuatlah batas minimum dan maksimum yang mungkin atau harus terjadi dari himpunan entitas yang satu ke himpunan entitas yang lain dan begitu sebaliknya.

Untuk memperlihatkan jumlah minimal atau maksimal yang dapat dihubungkan dalam tiap entitas dikenal dengan istilah kardinalitas (*cardinality*). Adapun yang dimaksud dengan kardinalitas adalah :

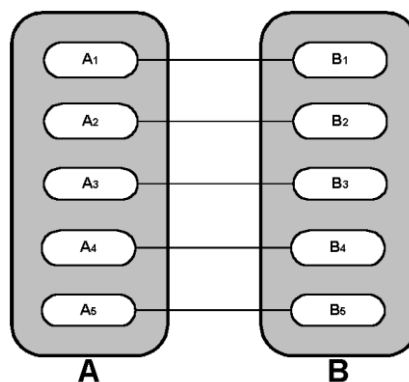
Mendefinisikan batas minimal dan maksimal tiap kejadian/peristiwa pada satu entitas yang mungkin direlasikan untuk satu kejadian pada entitas yang lain. Karena semua hubungan secara langsung, kardinal harus dapat didefinisikan secara langsung diantara kedua entitas setiap dihubungkan. (Whitten et. el., 2001:264)

Atau dapat pula dikatakan sebagai :

Analisis terhadap nilai minimal dan maksimal yang mungkin terjadi pada hubungan antara sebuah entitas dengan entitas yang lain.

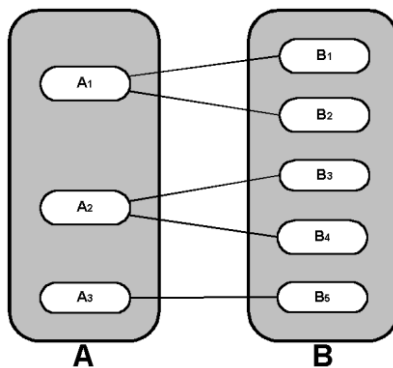
Dalam ER-D hubungan antara entitas dapat dipetakan menjadi beberapa pembatas, yaitu :

- Satu-ke-satu atau *one-to-one* (1-1)
Pembacaan pemetaan satu-ke-satu dalam ER-D, berarti bahwa tiap entitas akan berhubungan dengan paling banyak satu entitas yang lain, demikian sebaliknya. Misalnya entitas A akan berhubungan dengan maksimal satu entitas B. Ini berarti satu entitas A maksimum hanya berhubungan satu pada entitas B. Untuk lebih jelasnya dapat dilihat pada gambar 5.3. berikut ini :



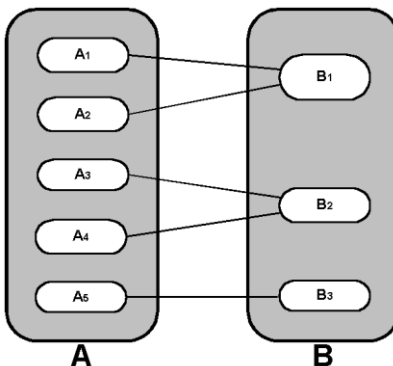
Gambar 5.3. Hubungan *one - to - one* (1 - 1)

- Satu-ke-banyak atau *one-to-many* (1-M / N)
Pembacaan pemetaan satu-ke-banyak adalah satu atribut dapat berhubungan dengan lebih dari satu (banyak) atribut yang lain, tetapi tidak sebaliknya lebih dari satu (banyak) atribut hanya berhubungan dengan satu atribut yang lain. Misalnya entitas A akan berhubungan dengan lebih dari satu entitas B. Ini berarti satu entitas A dapat berhubungan dengan lebih dari satu entitas B, sedangkan banyak atribut B hanya berhubungan dengan satu atribut A. Untuk lebih jelasnya dapat dilihat pada gambar 5.4 dibawah sebagai berikut :



Gambar 5.4. Hubungan *one - to - many* (1 - M/N)

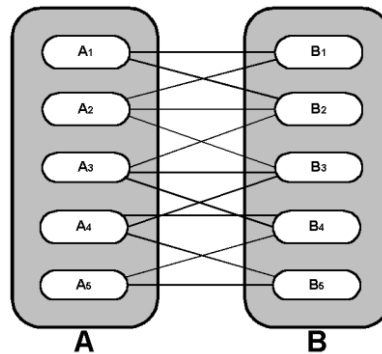
- Banyak-ke-satu atau *many-to-one* (M/N-1)
 Hubungan banyak-ke-satu merupakan kebalikan dari hubungan satu ke banyak, yaitu banyak (lebih dari satu) entitas yang satu akan berhubungan dengan hanya satu pada entitas yang lain, namun tidak sebaliknya. Misalnya entitas A berhubungan dengan hanya satu pada entitas B, sedangkan entitas B hanya dapat berhubungan dengan satu pada entitas A. untuk lebih jelasnya dapat dilihat pada gambar 5.5 :



Gambar 5.5. Hubungan *many - to - one* (M/N - 1)

- Banyak-ke-banyak atau *many-to-many* (M-N)
 Pembacaan pemetaan banyak-ke-banyak, ini berarti banyak entitas dapat dihubungkan dengan banyak entitas yang lain, demikian sebaliknya. Misalnya banyak entitas A berhubungan

dengan banyak entitas B dan berlaku sebaliknya, entitas B berhubungan dengan banyak entitas A. untuk memudahkan pembacaan, dapat dilihat pada gambar 5.6 dibawah ini :



Gambar 5.6. Hubungan *many - to - many* (M - N)

5.4. Tahapan Pembuatan ER-D

Pembuatan ER-D selalu dilakukan dengan bertahap. Biasanya menggunakan 2 (dua) tahap, yaitu :

- Tahap pembuatan diagram E-R awal (*preliminary design*)
Tahapan pertama bertujuan untuk mendapatkan sebuah gambaran basis data yang minimal yang dapat mengakomodasikan kebutuhan penyimpanan data terhadap semua kegiatan sistem atau untuk memulai usaha. Pembuatan desain awal ini hanya bertujuan untuk membuat gambaran secara garis besar dan memperlihatkan adanya hubungan antar entitas dan belumlihatkan relasi kardinalnya., yang kemudian akan dikoreksi bila terjadi kesalahan relasi atau kekurangan entitas. Tingkatan ini disebut dengan *enterprise level*.
- Tahap optimasi diagram E-R (*final design*)
Tahapan optimasi pada diagram E-R terbagi menjadi 2, yaitu :
 - *Entity relationship level*
Tingkatan ini hanya memperlihatkan adanya hubungan antar entitas, namun belum dibuat atribut pada tiap-tiap entitas dan belumlihatkan relasi kardinalnya.

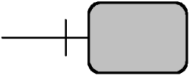

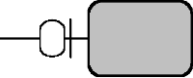
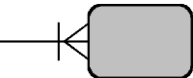


Pembuatan ini bertujuan untuk memudahkan bila terjadi perubahan terhadap penambahan entitas atau perubahan relasi.

- *Entity relationship attribute level*

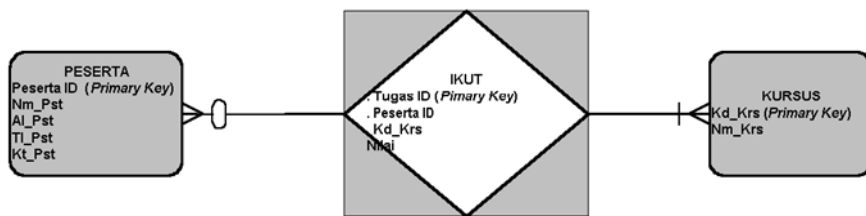
Tingkatan ini telah terbentuk diagram E-R dengan lengkap (terdapat atribut). Pembuatan tahapan ini biasanya dilakukan bila adanya keyakinan analis sistem bahwa diagram E-R yang dibuat telah sempurna (tidak ada perubahan lagi).

Selain dengan simbol diatas, dalam menggambarkan ER-D dapat pula menggunakan simbol/notasi kardinalitas. Simbol tersebut dapat dilihat pada tabel sebagai berikut :

Tabel V.1. Notasi Kardinalitas *Entity Relationship Diagram* (ER-D)

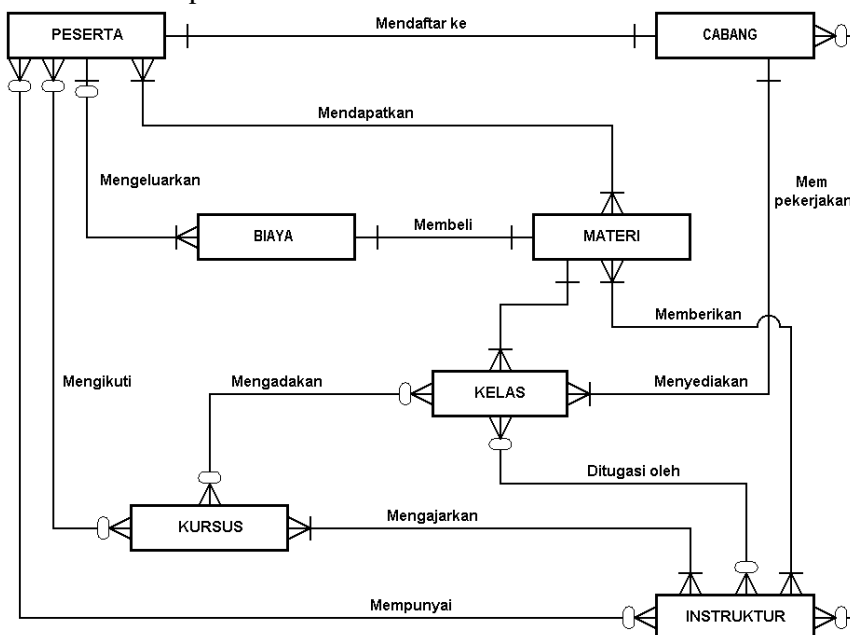
| Interpretasi Kadinal | Kejadian Minimum | Kejadian Maksimum | Notasi Gambar |
|-----------------------------------|------------------|---------------------|---|
| Satu dan hanya satu (1,1) | 1 | 1 |  atau  |
| Kosong atau satu (0,1) | 0 | 1 |  |
| Satu atau lebih (1,N) | 1 | Banyak / Many (> 1) |  |
| Kosong, satu, atau lebih (0,N) | 0 | Banyak / Many (> 1) |  |
| Lebih dari satu (N,N) | > 1 | > 1 |  |

Pembuatan ER-D dengan notasi diatas, sangat efisien dan efektif sekali dalam penggambaran ER-D dibandingkan cara sebelumnya, karena dalam pembuatan ER-D dengan notasi ini langsung dapat diketahui kardinalitasnya (batas minimal dan maksimal yang terjadi).



Gambar 5.7. Sebuah relasi (*many-to-many*)

Pada gambar 5.7 terlihat bahwa entitas PESERTA mempunyai notasi 0, ini dapat terjadi jika saja peserta tidak mengambil kursus / cuti. Sedangkan pada entitas kursus terlihat notasi satu sampai banyak (>1), karena tiap peserta yang ikut kursus minimal harus mengambil 1 kursus dan dapat pula lebih dari satu kursus. Misalnya peserta A sedang cuti dan ia tidak dapat kursus, maka yang terjadi adalah peserta tersebut mempunyai hubungan (0,N). Dan peserta B akan mengambil kursus yang lebih dari satu (1,N). Penggunaan notasi kardinalitas dapat pula digambarkan tanpa menggunakan atribut. Untuk lebih jelasnya mengenai penggunaan notasi kardinalitas tanpa atribut.



Gambar 5.8. Contoh penggunaan notasi kardinalitas hasil

Contoh kasus 1 :

Sebuah lembaga pendidikan Bahasa "LPB Semoga Lancar" memiliki 10 cabang yang tersebar di area JABODETABEK. Materi kursus yang ditawarkan pada setiap cabang adalah bahasa Inggris, Perancis, Jerman, Belanda, Mandarin dan Jepang. Setiap materi kursus memiliki biaya yang berbeda untuk lama waktu belajar yang bervariasi pula.

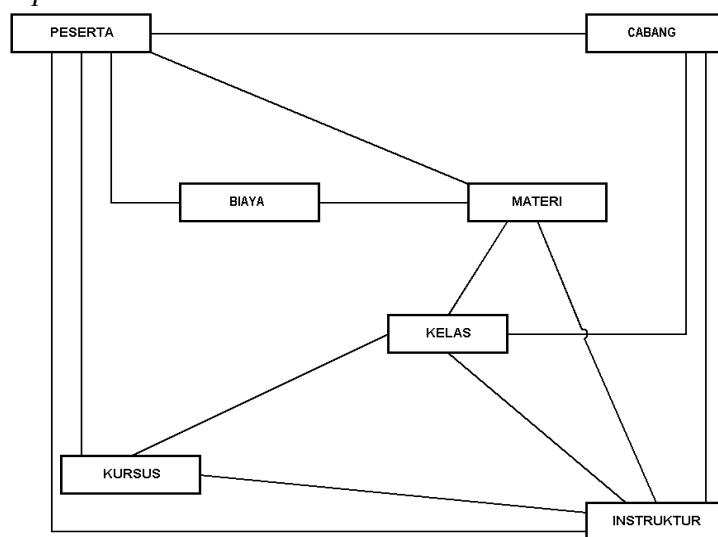
Pada setiap cabang, setiap materi kursus dapat diselenggarakan dalam beberapa kelas sesuai dengan jumlah peserta yang mendaftar untuk materi tersebut. Sedangkan lama waktu kursus diselenggarakan dihitung dalam term (kuartal), dimana seorang peserta dalam satu term hanya boleh ikut satu materi saja di salah satu cabang.

Seorang instruktur pada lembaga tersebut hanya diizinkan mengajar satu materi kursus, tetapi dapat mengajar materi tersebut pada beberapa kelas di semua cabang.

Berdasarkan uraian diatas gambarkan diagram E-R dari basis data yang merupakan model data pada lembaga tersebut. Untuk setiap entitas anda dibebaskan untuk memberikan atribut yang sesuai dengan entitas tersebut.

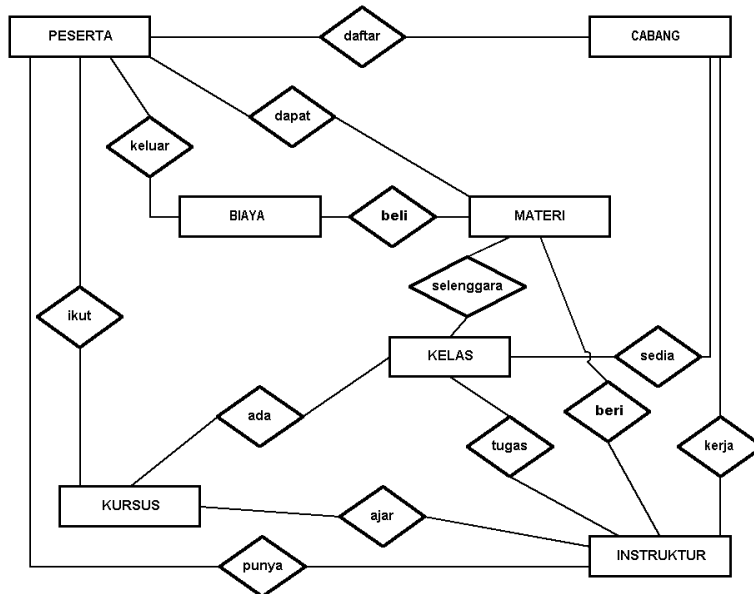
Pemecahan kasus :

1. Enterprise Level



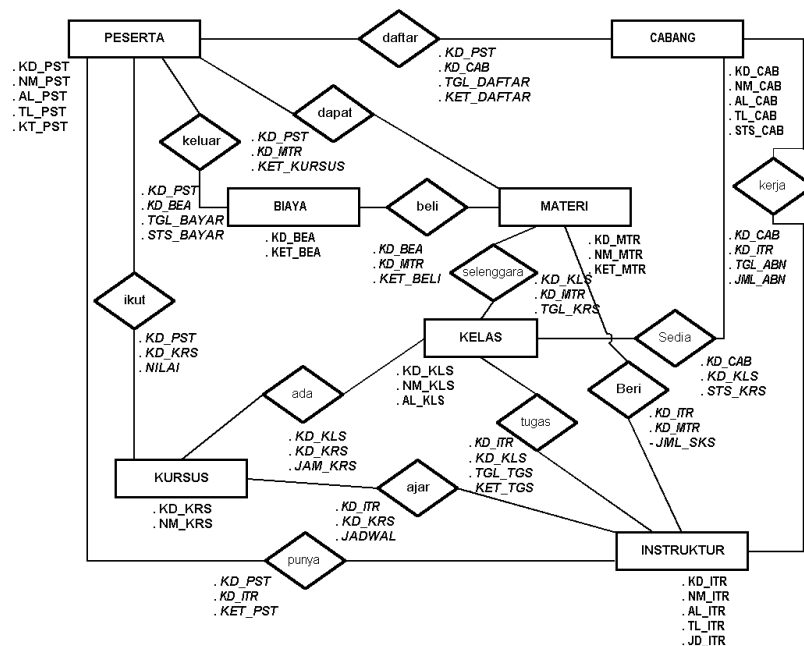
Gambar 5.9. Enterprise level

2. Entity Relationship Level

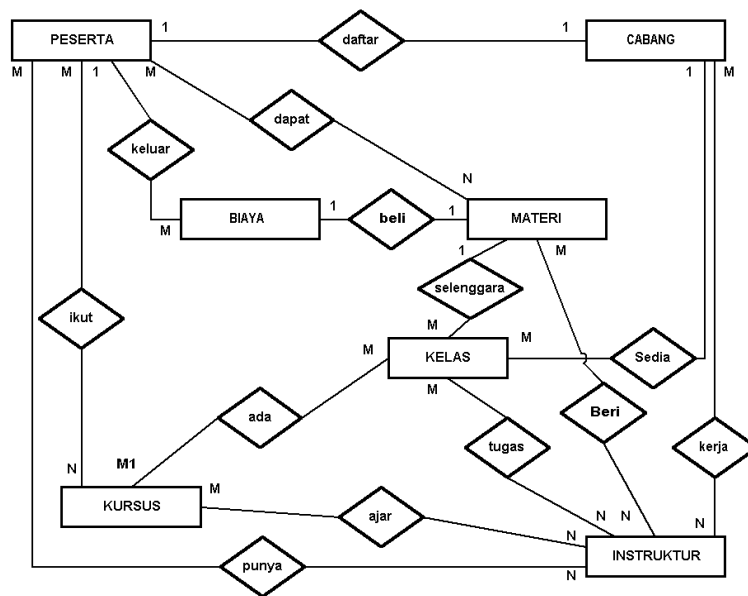


Gambar 5.10. Entity relationship level

3. Entity Relationship Attribute Level



Gambar 5.11. Entity relationship attribute level



Gambar 5.12. Kardinalitas ER-D

Kamus Data :

- Peserta {kd_pst, nm_pst, al_pst, tl_pst, kt_pst}
- Cabang {kd_cab, nm_cab, al_cab, tl_cab, sts_cab}
- Biaya {kd_bea, ket_bea}
- Materi {kd_mtr, nm_mtr, ket_mtr}
- Kelas {kd_kls, nm_kls, al_kls}
- Kursus {kd_krs, nm_krs}
- Instruktur {kd_itr, nm_itr, al_itr, tl_itr, jd_itr}
- Daftar {kd_pst, kd_cab, tgl_daftar, ket_daftar}
- Dapat {kd_pst, kd_mtr, ket_kursus}
- Keluar {kd_pst, kd_bea, tgl_bayar, sts_bayar}
- Beli {kd_bea, kd_mtr, ket_beli}
- Sedia {kd_cab, kd_mtr, tgl_mtr, sts_mtr}
- Kerja {kd_cab, kd_itr, tgl_abn, jml_abn}
- Ikut {kd_pst, kd_krs, nilai}
- Ada {kd_kls, kd_krs, jam_krs}
- Selenggara {kd_kls, kd_mtr, tgl_krs}
- Beri {kd_cab, kd_kls, sts_krs}
- Kasih {kd_itr, kd_mtr, jml_sks}
- Tugas {kd_itr, kd_kls, tgl_tgs, ket_tgs}

- Ajar {kd_itr, kd_krs, jadwal}
- Punya {kd_pst, kd_itr, ket_pst}

Aplikasi dengan menggunakan *SQL Server* dengan *Transac-SQL* atau yang dikenal dengan *T-SQL* adalah sebagai berikut:

- Langkah pertama, pembuatan basis data :
CREATE DATABASE LPB_Semoga_Lancar
ON
(NAME =LPB_SL_Dat
FILE NAME = 'H:\Data\LPB\ LPB_SL.mdf',
SIZE = 10,
MAXSIZE = 50,
FILEGROWTH = 1)
- Langkah kedua, pembuatan tabel-tabel yang digunakan :
CREATE TABLE Peserta
(Kd_Pst CHAR(8) NOT NULL,
Nm_Pst CHAR(30) NOT NULL,
Al_Pst CHAR(40) NOT NULL,
Tl_Pst CHAR(17) NOT NULL,
Kt_Pst CHAR(20) NOT NULL)

CREATE TABLE Cabang
(Kd_Cab CHAR(5) NOT NULL,
Nm_Cab CHAR(30) NOT NULL,
Al_Cab CHAR(40) NOT NULL,
Tl_Cab CHAR(17) NOT NULL,
Sts_Cab CHAR(30) NOT NULL)

Dan seterusnya hingga semua tabel terbentuk.

5.5. Varian Entitas

Semua himpunan entitas dalam sebuah ER-D yang telah dibahas diatas merupakan himpunan entitas yang bebas dan kuat (*strong entity sets*). Himpunan entitas yang demikian tidak bergantung pada entitas yang lain, dapat dilihat pada entitas Peserta, Cabang, Kelas, instruktur dan sebagainya yang saling bergantung.

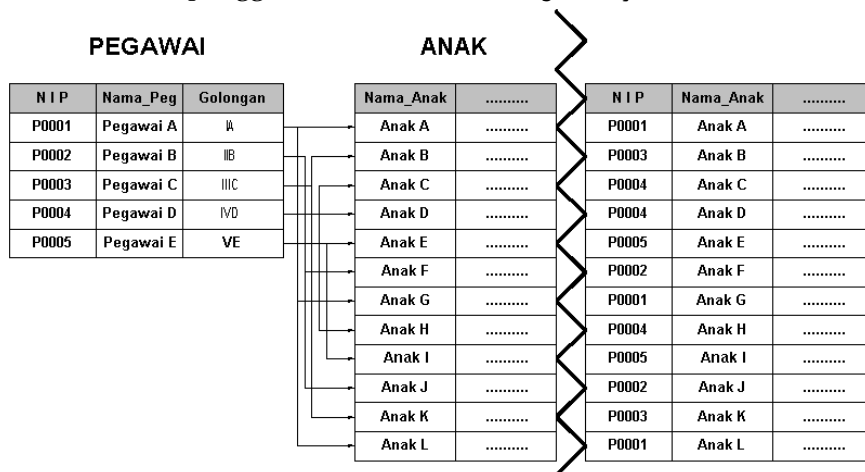
Namun dalam pembuatan ER-D tidak semuanya entitas dapat berdiri sendiri. Dapat pula sebuah entitas melibatkan himpunan entitas yang lemah (*weak entity*). Dalam pembuatan entitas yang lemah dibuat simbol sebagai berikut :



Sedangkan simbol untuk hubungan antar entitas adalah sebagai berikut :



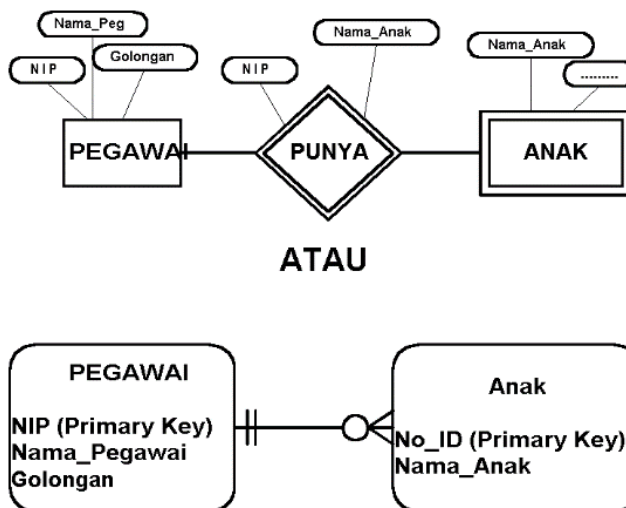
Adapun isi dari entitas sangat lemah bergantung sekali pada entitas yang lebih kuat. Himpunan yang demikian biasanya tidak mempunyai atribut yang dapat digunakan sebagai kunci (*key*) yang menjamin data dalam keadaan yang unik (tidak ganda). Berikut adalah contoh penggunaan *weak* dan *strong* entity :



Gambar 5.13. Contoh hubungan tabel kuat dan lemah

Pada entitas diatas, entitas Anak dapat saja terjadi nama anak sama, misalnya Anak A bernama Budi dan Anak I bernama Budi juga. Selain itu dapat pula seorang pegawai yang mempunyai anak yang

lebih dari satu. Dengan fakta entitas diatas, dapat kita lihat bahwa entitas Anak sangat bergantung sekali dengan entitas Pegawai. Oleh karena itu entitas Anak disebut dengan entitas lemah (*weak entity*) dan entitas Pegawai disebut dengan entitas kuat (*strong entity*). Adapun hasil yang didapat dari penggabungan kedua entitas tersebut, dapat dilihat pada entitas paling kanan. Sedangkan pembuatan ER-D nya dapat dilihat pada gambar dibawah.

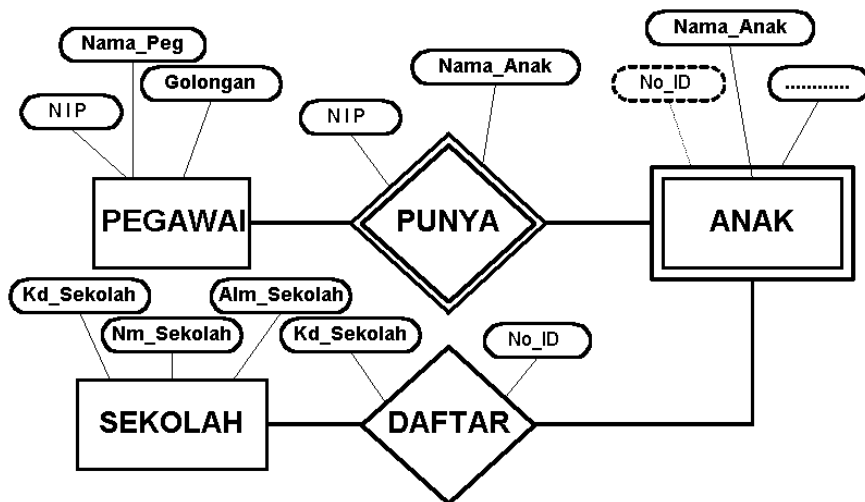


Gambar 5.14. Contoh relasi dengan *weak-entity*

Aplikasi dengan SQL Server adalah sebagai berikut :

```
CREATE TABLE Pegawai
(NIP Char(10) NOT NULL,
Nama_Pegawai Char(30) NOT NULL,
Golongan Int NOT NULL Default 0,
PRIMARY KEY (NIP))
```

Bagaimana bila ER-D diatas akan dikembangkan? misalnya dengan ditambahkan entitas Sekolah? Sedangkan entitas Anak tidak bergantung dengan entitas Sekolah. Untuk lebih jelasnya, dapat dilihat pada gambar sebagai berikut:



Gambar 5.15. Contoh penggunaan gabungan hubungan antara *weak* dan *strong entity*

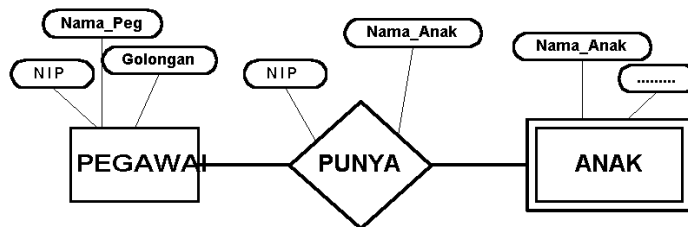
Untuk memudahkan penggabungan seperti diatas, kondisi entitas Anak tidaklah selemah pertama tetapi harus diberikan karakteristik yang berbeda. Untuk mengubahnya, anda dapat menambahkan atribut yang mempunyai kondisi yang unik, misalnya ditambahkan atribut No_ID.

| ANAK | | | | |
|-----------|--------|-------|-----------|--------|
| Nama_Anak | No_ID | NIP | Nama_Anak | No_ID |
| Anak A | ID_001 | P0001 | Anak A | ID_001 |
| Anak B | ID_002 | P0003 | Anak B | ID_002 |
| Anak C | ID_003 | P0004 | Anak C | ID_003 |
| Anak D | ID_004 | P0004 | Anak D | ID_004 |
| Anak E | ID_005 | P0005 | Anak E | ID_005 |
| Anak F | ID_006 | P0002 | Anak F | ID_006 |
| Anak G | ID_007 | P0001 | Anak G | ID_007 |
| Anak H | ID_008 | P0004 | Anak H | ID_008 |
| Anak I | ID_009 | P0005 | Anak I | ID_009 |
| Anak J | ID_010 | P0002 | Anak J | ID_010 |
| Anak K | ID_011 | P0003 | Anak K | ID_011 |
| Anak L | ID_012 | P0001 | Anak L | ID_012 |

Gambar 5.16. Contoh penambahan atribut No_ID Pada tabel lemah

Penambahan No_ID diatas dimaksudkan agar memudahkan dalam penggabungan atribut semata. Selain itu penambahan atribut dapat membuat entitas yang lemah menjadi lebih kuat.

Pada beberapa buku, penggunaan simbol hubungan entitas lemah dan kuat tidak dibedakan. Untuk lebih jelasnya dapat dilihat pada gambar dibawah.



Gambar 5.17. Contoh penggunaan simbol hubungan yang sama digunakan pada *weak-entity*

5.6. Spesialisasi dan Generalisasi

Pada praktik pembuatan hubungan antar entitas, dapat pula terjadi bahwa dalam entitas terdapat entitas yang lain atau disebut dengan sub entitas. Dengan penurunan ini maka akan didapatkan sebuah entitas yang lebih utama atau *superior* yang menjadi sumber dari entitas di bawahnya. Penurunan entitas ini ditujukan untuk membuat entitas yang lebih spesifik atau spesialisasi. Sehingga didapatkan definisi spesialisasi adalah:

Suatu entitas yang membentuk himpunan entitas tingkat yang lebih rendah (*subset*) berdasarkan perbedaan atribut yang dimiliki oleh atribut-atribut sebuah kumpulan entitas.

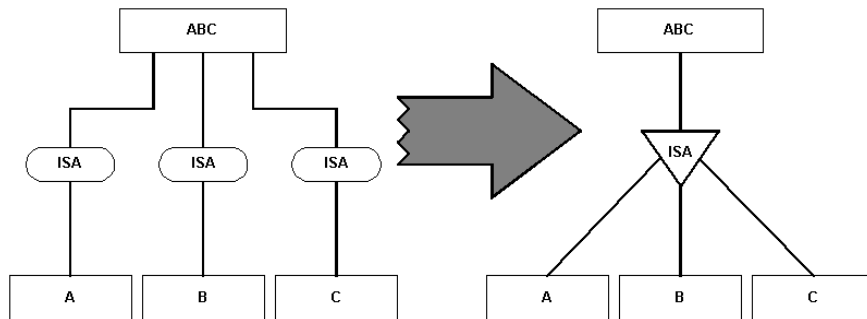
Untuk mempermudah pemahaman spesialisasi, dapat dilihat pada gambar 5.18. Simbol yang digunakan untuk pembuatan spesialisasi digunakan simbol dibawah ini :



Simbol ISA tersebut diatas berasal dari kata "IS A" dan tidak hanya digunakan pada spesialisasi, melainkan juga digunakan pada

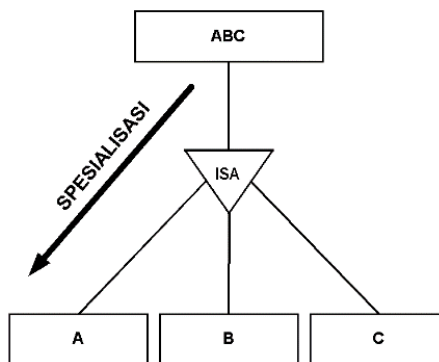
generalisasi (lihat pembahasannya pada halaman selanjutnya pada subbab ini).

Sebenarnya simbol penghubung spesialisasi atau generalisasi merupakan gabungan dari beberapa *subset*. (gambar 5.18)



Gambar 5.18. Cara pembuatan spesialisasi

Untuk memudahkan pembacaan, beberapa ISA diatas (gambar 5.18) digabung menjadi satu ISA. Adapun cara pembuatan spesialisasi adalah dari atas ke bawah (*top-down*) yang dapat dilihat pada gambar 5.19 dibawah ini:



Gambar 5.19. Arah penggunaan spesialisasi

Pada aplikasi pembuatan program, pembuatan spesialisasi ini sangat baik sekali digunakan untuk efisiensi ruang penyimpanan *file*. Untuk lebih jelasnya dapat dilihat pada gambar 5.20 berikut ini :

Tabel REKENING

| No_Rek | Nm_Nasabah | Alm_Nasabah | Saldo | Tabungan | Deposito | Bunga (%) | Jth_Tempo |
|----------|------------|-------------|--------|----------|----------|-----------|------------|
| 92777001 | Nasabah A | Jl. A | 10.000 | 10.000 | 0 | 0 | 0 |
| 92777002 | Nasabah B | Jl. B | 15.000 | 15.000 | 0 | 0 | 0 |
| 92777003 | Nasabah C | Jl. C | 16.000 | 16.000 | 0 | 0 | 0 |
| 92777004 | Nasabah D | Jl. D | 75.000 | 0 | 75.000 | 10 | 12/10/2001 |
| 92777005 | Nasabah E | Jl. E | 40.000 | 0 | 40.000 | 15 | 15/05/2002 |
| 92777006 | Nasabah F | Jl. F | 46.000 | 16.000 | 30.000 | 20 | 13/09/2003 |

Tabel SALDO

| No_Rek | Saldo |
|----------|--------|
| 92777001 | 10.000 |
| 92777002 | 15.000 |
| 92777003 | 16.000 |
| 92777004 | 75.000 |
| 92777005 | 40.000 |
| 92777006 | 46.000 |

Tabel TABUNGAN

| No_Rek | Tabungan |
|----------|----------|
| 92777001 | 10.000 |
| 92777002 | 15.000 |
| 92777003 | 16.000 |
| 92777006 | 16.000 |

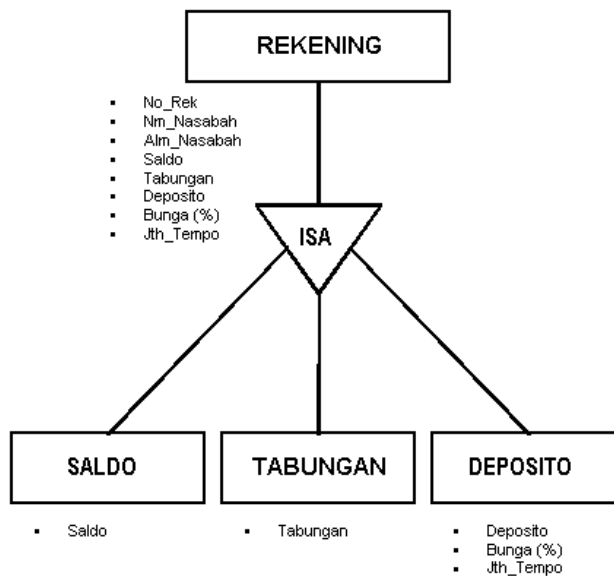
Tabel DEPOSITO

| No_Rek | Deposito | Bunga (%) | Jth_Tempo |
|----------|----------|-----------|------------|
| 92777004 | 75.000 | 10 | 12/10/2001 |
| 92777005 | 40.000 | 15 | 15/05/2002 |
| 92777006 | 30.000 | 20 | 13/09/2003 |

Gambar 5.20. Contoh penggunaan tabel / entitas spesialisasi

Dari gambar terlihat pada tabel rekening terdapat *field* yang bernilai kosong (o), yaitu *field* Tabungan, Deposito, Bunga (%) dan Jth_Tempo. Jika yang kosong terdiri dari ribuan atau bahkan jutaan nasabah, berapa besar pemborosan di penyimpanan? Jika di-kurs ke mata uang, berapa besar uang yang terbuang?

Untuk mengatasinya digunakan cara spesialisasi, sehingga tabel Rekening dipecah menjadi tiga buah tabel anak, yaitu tabel Saldo, Tabungan dan Deposito. Dalam penggambaran ER-D nya dapat dilihat pada gambar 5.21.



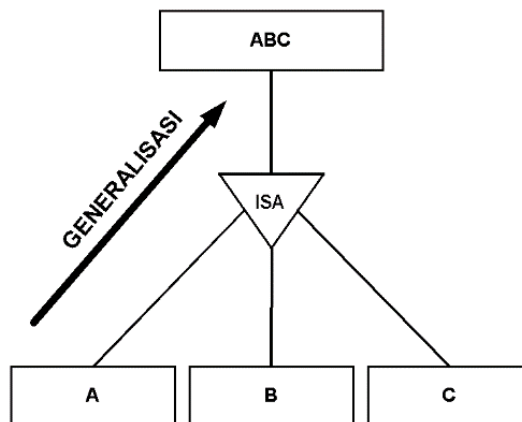
Gambar 5.21. Penggambaran hubungan dengan spesialisasi dari gambar 5.20

Kebalikan dari spesialisasi adalah generalisasi, sehingga generalisasi didefinisikan sebagai :

Penggabungan beberapa himpunan entitas yang berdasarkan atribut, sehingga membentuk sebuah himpunan entitas tingkat yang lebih tinggi.

Pembuatan generalisasi dilakukan dari bawah ke atas (*bottom-up*). Adapun perbedaan dalam aplikasi perancangan basis data, umumnya pada spesialisasi akan terlihat secara eksplisit pada hasil akhir ER-D. Sedangkan pada generalisasi dibuat dengan pertimbangan simplifikasi (penyederhanaan) dan sering kali ditiadakan (tidak diperlihatkan secara eksplisit) pada akhir ER-D. Peniadaan generalisasi ini dipresentasikan dengan adanya atribut baru pada entitas akhir.

Untuk mempermudah pemahaman generalisasi, dapat dilihat pada gambar 5.22.



Gambar 5.22. Arah penggunaan generalisasi

Adapun aplikasi yang akan digunakan untuk contoh pembuatan generalisasi, dapat dilihat pada gambar 5.23 dibawah ini :

Tabel SALDO

| No_Rek | Nm_Nasabah | Alm_Nasabah | Saldo |
|----------|------------|-------------|--------|
| 92777001 | Nasabah A | Jl. A | 10.000 |
| 92777002 | Nasabah B | Jl. B | 15.000 |
| 92777003 | Nasabah C | Jl. C | 16.000 |
| 92777004 | Nasabah D | Jl. D | 75.000 |
| 92777005 | Nasabah E | Jl. E | 40.000 |
| 92777006 | Nasabah F | Jl. F | 46.000 |

Tabel TABUNGAN

| No_Rek | Nm_Nasabah | Alm_Nasabah | Tabungan |
|----------|------------|-------------|----------|
| 92777001 | Nasabah A | Jl. A | 10.000 |
| 92777002 | Nasabah B | Jl. B | 15.000 |
| 92777003 | Nasabah C | Jl. C | 16.000 |
| 92777006 | Nasabah F | Jl. F | 16.000 |

Tabel DEPOSITO

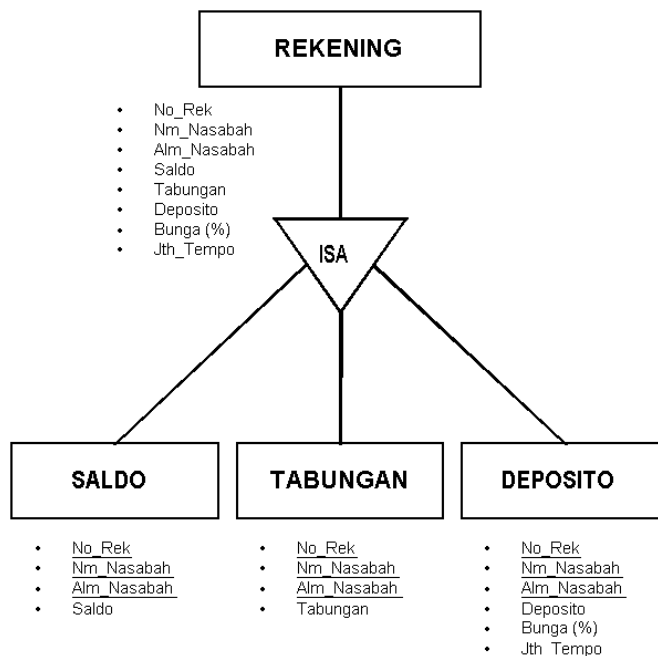
| No_Rek | Nm_Nasabah | Alm_Nasabah | Deposito | Bunga (%) | Jth_Tempo |
|----------|------------|-------------|----------|-----------|------------|
| 92777004 | Nasabah D | Jl. D | 75.000 | 10 | 12/10/2001 |
| 92777005 | Nasabah E | Jl. E | 40.000 | 15 | 15/05/2002 |
| 92777006 | Nasabah F | Jl. F | 46.000 | 20 | 13/09/2003 |

Tabel REKENING

| No_Rek | Nm_Nasabah | Alm_Nasabah | Saldo | Tabungan | Deposito | Bunga (%) | Jth_Tempo |
|----------|------------|-------------|--------|----------|----------|-----------|------------|
| 92777001 | Nasabah A | Jl. A | 10.000 | 10.000 | 0 | 0 | 0 |
| 92777002 | Nasabah B | Jl. B | 15.000 | 15.000 | 0 | 0 | 0 |
| 92777003 | Nasabah C | Jl. C | 16.000 | 16.000 | 0 | 0 | 0 |
| 92777004 | Nasabah D | Jl. D | 75.000 | 0 | 75.000 | 10 | 12/10/2001 |
| 92777005 | Nasabah E | Jl. E | 40.000 | 0 | 40.000 | 15 | 15/05/2002 |
| 92777006 | Nasabah F | Jl. F | 46.000 | 16.000 | 30.000 | 20 | 13/09/2003 |

Gambar 5.23. Contoh penggunaan tabel / entitas generalisasi

Gambar 5.23 diatas terlihat betapa borosnya *field* tabel yang digunakan. Terlihat pada tabel Saldo, Tabungan dan Deposito terdapat *field-field* yang sama, yaitu No_Rek, Nm_Nasabah dan Alm_Nasabah. Untuk menghindari dari pengulangan *field* yang tidak perlu, digunakan generalisasi sehingga hanya satu tabel saja yang digunakan, yaitu tabel Rekening. Untuk memudahkan penggambarannya, dapat dilihat pada gambar 5.24 sebagai berikut :



Gambar 5.24. Penggambaran hubungan dengan generalisasi dari gambar 5.22

Pada gambar 5.24, terdapat atribut yang digarisbawahi untuk menunjukkan persamaan atribut yang digunakan pada entitas yang berbeda.

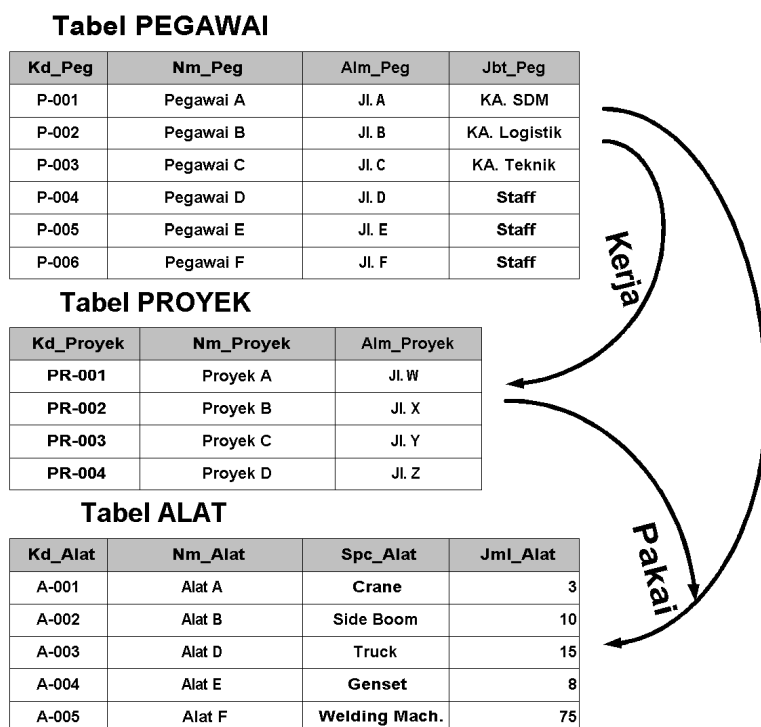
5.7. Agregasi (Aggregation)

Salah satu pembatasan pada ER-D adalah di sana tidak mungkin untuk mempercepat hubungan diantara hubungan entitas. Namun pada pelaksanaannya mungkin saja terjadi hubungan yang

terjadi harus disyaratkan oleh hubungan yang lain. Dengan kata lain, bahwa suatu hubungan terbentuk tidak hanya dari satu entitas saja, melainkan mengandung juga unsur dari hubungan yang lain. Kejadian ini disebut juga dengan agregasi. Oleh karena itu agregasi adalah **suatu abstraksi dari sebuah hubungan yang dianggap sebagai himpunan entitas**.

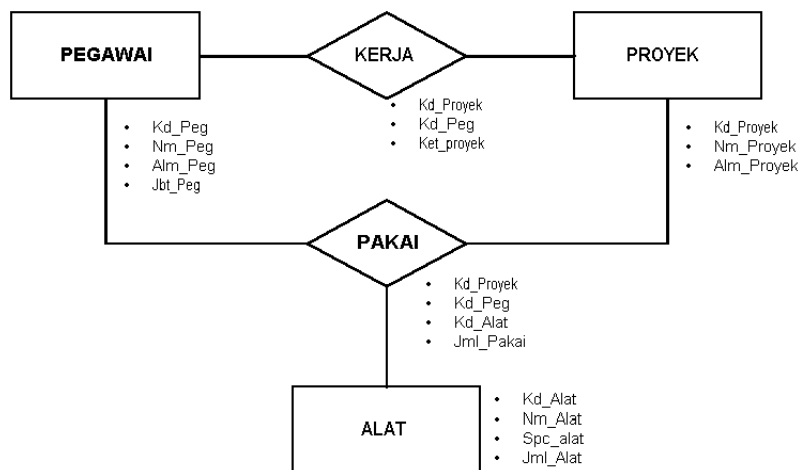
Penggambaran sebuah hubungan yang secara langsung berhubungan dengan yang lain pada ER-D sebenarnya tidak tepat, bahkan pada teori dengan tegas tidak memperbolehkannya. Oleh karena itu sebagai garis tengahnya kita menggunakan notasi khusus untuk menunjukkan adanya agregasi semacam ini.

Untuk contohnya, misalnya terdapat tiga buah himpunan tabel. Ketiga tabel tersebut ternyata saling berhubungan. Misalnya tabel Pegawai mempunyai hubungan dengan tabel Proyek dan tabel Alat. Sedangkan tabel Proyek mempunyai hubungan dengan tabel Alat (gambar 5.25)



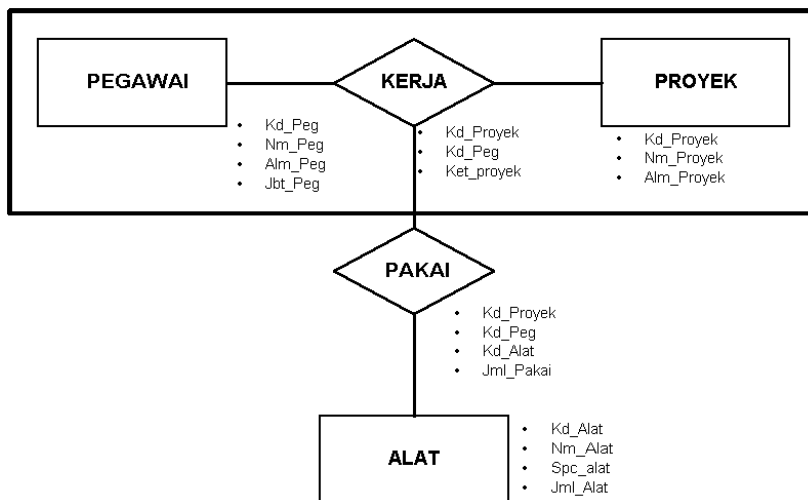
Gambar 5.25. Hubungan antar tabel

Ketiga tabel tersebut akan dihubungkan satu persatu. Pertama hubungan antara Pegawai dengan Proyek yang terhubung akibat adanya aktivitas pekerjaan. Sedangkan Pegawai mempunyai hubungan dengan Alat, karena pegawai dapat bekerja jika ada alat kerjanya. Dari penjelasan tersebut, maka entitas Proyek secara langsung berhubungan dengan Alat karena proyek dapat dilakukan jika menggunakan Alat. Untuk memudahkan-nya, dapat dilihat pada gambar 5.26 berikut ini :



Gambar 5.26. Bentuk ER-D yang digunakan

Penggambaran ER-D diatas sudah benar, namun masih tampak kabur pada faktor kronologisnya. Pada diagram tersebut diatas, kedua himpunan entitas merupakan himpunan bebas, dan tidak mempunyai ketergantungan satu sama lain. Hal tersebut tidaklah demikian adanya, karena sebenarnya terdapat ketergantungan antara Pegawai dengan Proyek dan Alat. Oleh karena itu digunakannya agregasi untuk membuat satu hubungan akan membutuhkan hubungan yang lain. (gambar 5.27)



Gambar 5.27. Agregasi dari ER-D pada gambar 5.25

Dengan dibuatnya agregasi, maka hubungan dapat dilihat prioritas hubungan yang utama. Dalam gambar 5.27 penggunaan alat sangat bergantung sekali dengan hubungan antara Pegawai dengan Proyek. Tanpa adanya Pegawai dan Proyek, maka penggunaan Alat tidak dibutuhkan.

Di samping untuk menampilkan nilai-nilai atribut yang ada di dalam tabel seperti contoh sebelumnya, sering pula data agregasi digunakan untuk penghitungan banyaknya *record*, total, rata-rata nilai atribut, nilai terbesar dan nilai terkecil. Data agregasi semacam ini dapat diperoleh dengan menggunakan fungsi-fungsi sebagai berikut:

- **Count**, digunakan untuk mendapatkan banyaknya *record* dari hasil *query*.

Contoh :

Menampilkan banyaknya *record* pegawai :

Select Count (*)

From PEGAWAI → tabel pegawai

Menampilkan banyaknya *record* pegawai yang nikah :

Select Count (*)

From PEGAWAI → tabel pegawai

Where status **like** 'Nikah'

- **Sum**, digunakan untuk mendapatkan nilai total dari suatu atribut yang berisi data numerik dari hasil *query*.

Contoh :

Menampilkan banyaknya total gaji pegawai :

Select Sum (gaji)

From PEGAWAI → tabel pegawai

- **Avg**, digunakan untuk menghitung nilai rata-rata dari suatu atribut yang berisi data numerik dari hasil *query*.

Contoh :

Menampilkan rata-rata gaji pegawai :

Select Avg (gaji)

From PEGAWAI → tabel pegawai

- **Max**, digunakan untuk mencari nilai tertinggi (maksimum) dari suatu hasil *query*.

Contoh :

Menampilkan gaji pegawai yang tertinggi :

Select Max (gaji)

From PEGAWAI → tabel pegawai

- **Min**, digunakan untuk mencari nilai terendah (minimum) dari suatu hasil *query*.

Contoh :

Menampilkan gaji pegawai yang terendah :

Select Min (gaji)

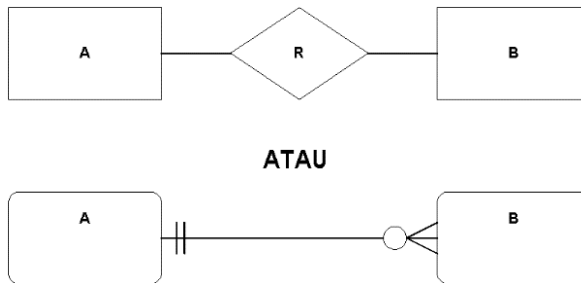
From PEGAWAI → tabel pegawai

5.8. Varian relasi

Relasi atau hubungan yang terjadi pada umumnya dilakukan antar dua entitas yang berbeda yang dikenal dengan nama hubungan biner (*binary relations*). Ini dapat dilihat pada contoh-contoh sebelumnya. Namun pada pelaksanaannya dapat pula terjadi hubungan dengan hanya satu entitas atau lebih dari dua entitas. Oleh karena itu hubungan (*relationship*) dalam ER-D dikelompokkan kedalam tiga hubungan, yaitu:

1. Hubungan biner (*Binary Relationship*)

Hubungan atau relasi biner telah dicontohkan dalam contoh-contoh sebelumnya. Adapun yang dimaksud dengan hubungan biner adalah hubungan antar dua buah entitas yang berbeda. Skema penulisannya dapat dilihat dibawah ini :

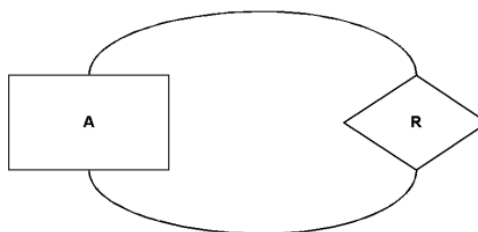


Gambar 5.28. Skema penulisan hubungan biner

Penulisan dengan biner ini sangat dianjurkan sekali dalam penulisan ER-D, karena dapat memudahkan membedakan derajat hubungan yang terjadi dalam ER-D. Pada gambar diatas ditampilkan dalam dua notasi yang mempunyai fungsi yang sama. Notasi pertama yang umum dan sering digunakan, sedangkan notasi yang kedua adalah notasi yang baru dan kini sudah banyak digunakan. Adapun contohnya dapat dilihat pada contoh-contoh sebelumnya (gambar 7.44-45, 9.23-25 dan seterusnya)

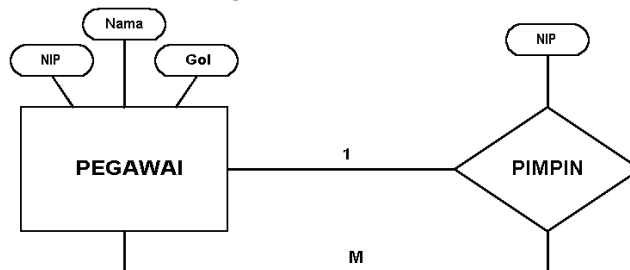
- a. Hubungan tunggal (*Unary Relationship*) atau *Recursive Relationship*

Hubungan tunggal adalah hubungan yang terjadi dari himpunan entitas yang sama. (gambar 5.28) Penggunaan hubungan tunggal bisa saja terjadi pada kasus-kasus yang ditemui sehari-hari

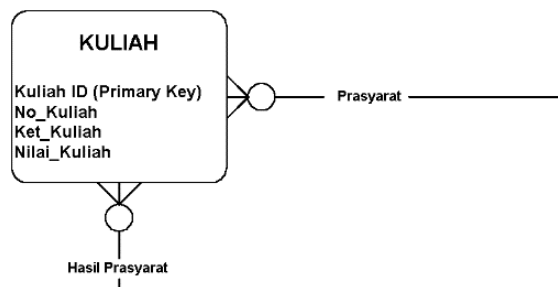


Gambar 5.29. Skema penulisan hubungan tunggal

Hubungan tunggal pada ER-D dapat terjadi pada suatu perusahaan atau lembaga pendidikan. Hal yang paling mudah sekali bila sekumpulan pegawai atau karyawan yang lebih dari satu akan memimpin pegawai yang lain (gambar 5.30). Oleh karena itu hubungan tunggal sebaiknya digunakan bila dalam keadaan yang terdesak dan penggunaan hubungan biner sudah tidak dapat dilakukan lagi.



Gambar 5.30. Contoh hubungan tunggal



Gambar 5.31. Contoh hubungan tunggal dengan notasi kardinalitas

Pada gambar 5.30 terlihat bahwa seorang pegawai akan memimpin pegawai yang lain. Dalam memimpin pegawai lain, tabel atau entitas yang digunakan tetap satu, baik pegawai yang memimpin maupun pegawai yang dipimpin.

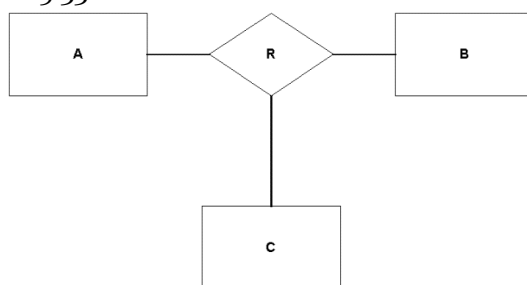
Demikian pula pada contoh gambar 5.31, untuk kuliah suatu bidang studi dapat berhubungan dengan yang lain dalam satu tabel. Misalnya ada mahasiswa yang akan mengambil mata kuliah pemrograman, boleh mengambil setelah mahasiswa tersebut telah mengambil mata kuliah logika dan algoritma, dan seterusnya.

b. Hubungan multi entitas (*N-Ary Relationship*)

Hubungan multi entitas adalah hubungan dari tiga himpunan entitas atau lebih. Sebaiknya bentuk hubungan ini sedapat mungkin dihindari, karena pada hubungan ini dapat mengaburkan derajat hubungan yang ada pada hubungan tersebut.

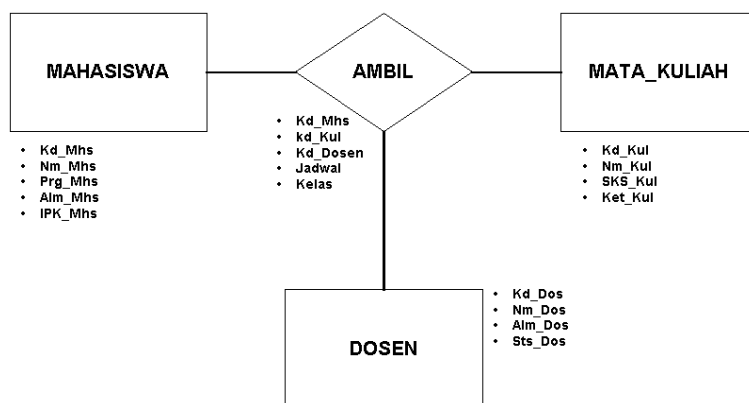
Adapun skema penulisan hubungan multi entitas dapat dilihat pada gambar 5.32.

Sedangkan contoh penulisannya dalam ER-D dapat dilihat pada gambar 5.33.



Gambar 5.32. Skema penulisan hubungan multi entitas

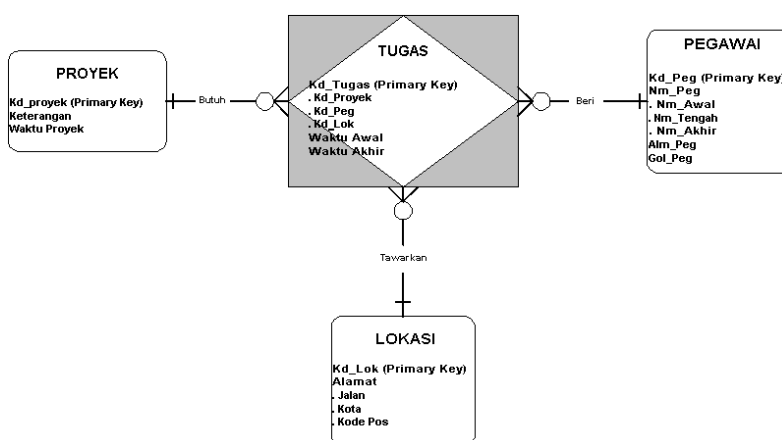
Penulisan ER-D dengan multi entitas dapat terjadi pada perusahaan (gambar 5.34) maupun lembaga pendidikan (gambar 5.33), namun sekali lagi sebaiknya hindari pemakaian multi entitas ini.



Gambar 5.33. Contoh hubungan multi entitas

Pada gambar 5.33, dapat terjadi pada kampus yang menggunakan sistem SKS murni, yaitu tiap mahasiswa dapat menentukan sendiri mata kuliah yang akan diambil, dosen yang akan mengajar dan waktu kuliahnya. Oleh karena itu pada gambar tersebut tertera mahasiswa akan mengambil Dosen dan Mata_Kuliah. Yang menjadi tidak jelas pada hubungan diatas adalah derajat hubungan yang mengacu pada seberapa besar tingkat responden antara sebuah entitas dengan himpunan entitas yang lain.

Pertama kali mahasiswa akan mengambil mata kuliah untuk semester depan, kemudian mahasiswa akan mencari dosen yang akan mengajar mata kuliah tersebut. Ketiganya akan disatukan dalam suatu ruang dan waktu yang sesuai.



Gambar 5.34. Contoh hubungan multi entitas dengan notasi kardinalitas

Pada gambar 5.34, terlihat bahwa suatu objek proyek akan ditentukan oleh lokasi dan pegawai, sedangkan ketiganya akan dihubungkan dengan tugas. Pertama kali proyek akan dikerjakan oleh pegawai. Proyek tersebut akan ditentukan oleh lokasi, kemudian ketiga entitas tersebut disatukan dalam suatu hubungan tugas.

Contoh kasus 2 :

Sebuah rumah sakit akan merancang basisdata untuk aplikasi pelayanan pasien rawat jalan, dimana jumlah pasien yang berkunjung ke rumah sakit tersebut rata-rata dalam sehari mencapai 1000 orang. Untuk setiap pasien yang datang berobat langsung dirujuk ke klinik / unit yang sesuai. Rumah sakit tersebut memiliki 19 klinik antara lain : klinik gigi, THT, jantung, kulit, poliklinik umum dan lain-lain. Pada setiap klinik ditugaskan beberapa dokter yang bertugas secara bergiliran yang selalu didampingi oleh seorang perawat. Setelah pasien dipelihara oleh dokter, kemudian bagi pasien tersebut diberikan resep berisi daftar obat-obatan untuk menyembuhkan penyakit yang diderita oleh pasien tersebut.

Keterangan :

- Buatlah *database*, semua tabel dan lainnya dengan menggunakan perintah *Transac-SQL* atau *T-SQL*.
- Buat *database* dengan nama : **RSAKIT**
- Buatlah tabel-tabel dibawah ini dengan menggunakan *Transac-SQL*, dengan nama seperti tertera diatas tabel, sebagai berikut :

Nama Tabel : **Obat**

| KdObat | NmObat | KetObat | Harga |
|--------|--------------------|-------------------|-------|
| O001 | Ponstan | Sakit Gigi | 5000 |
| O002 | Cutton Buds | Sakit Telinga | 1000 |
| O003 | Pacekap | Sakit Hati | 2500 |
| O004 | Canesten | Sakit Kulit | 9500 |
| O005 | Bodrek | Sakit Flu | 3000 |
| O006 | Sinse | Segala Penyakit | 55000 |
| O007 | Jamu Urat Nadi | Sakit Otak | 90000 |
| O008 | Sr. Biji Kedondong | Sakit Tenggorokan | 2000 |
| O009 | Kuku Bima TL | Bersalin | 1500 |

Nama Tabel : **Penyakit**

| KdSakit | NmSakit |
|---------|-------------------|
| S001 | Sakit Gigi |
| S002 | Sakit Telinga |
| S003 | Sakit Hati |
| S004 | Sakit Tenggorokan |
| S005 | Sakit Jantung |
| S006 | Sakit Kulit |

| KdSakit | NmSakit |
|---------|----------------|
| S007 | Sakit Flu |
| S008 | Sakit Bersalin |
| S009 | Sakit Mata |
| S010 | Sakit DBD |

Nama Tabel : **Klinik**

| KdKln | NmKln | RngKln |
|-------|-----------------|-----------|
| K001 | Klinik Gigi | Ruang 201 |
| K002 | Klinik THT | Ruang 101 |
| K003 | Klinik Jantung | Ruang 102 |
| K004 | Klinik Kulit | Ruang 200 |
| K005 | Klinik Umum | Ruang 100 |
| K006 | Klinik Bersalin | Ruang 104 |

Nama Tabel : **Perawat**

| KdPrwt | NmPrwt | AlpPwto1 |
|--------|-------------|--------------------------|
| PR001 | Nava Urbach | Jl. Boleh Juga 201 |
| PR002 | Tamara B. | Jl. Singo 76 |
| PR003 | Marshanda | Jl. Singo Di Mejo 56 |
| PR004 | Dian Sastro | Jl. Suka Donk 001 |
| PR005 | Puput Novel | Jl. Gandaria Terpojok 59 |
| PR006 | Lulu Tobing | Jl. Bungur Gede 212 |

| AlPrwto2 | KtPrwt | KpPrwt | TlPrwt |
|--------------|-------------|--------|----------|
| Dago | Bandung | 65930 | 698-3296 |
| Parangtritis | Yogyakarta | 23180 | 235-5689 |
| Parangtjajam | Yogyakarta | 23050 | 236-8970 |
| Bringin sewu | Jakarta | 17980 | 902-1310 |
| Gandaria | Jakarta | 12450 | 752-6986 |
| Kedu | Jawa Tengah | 12340 | 56-523 |

Nama Tabel : **Dokter**

| KdDok | NmDok | AlDoko1 |
|-------|---------|--------------------------------|
| DK001 | Lula K. | Jl. Banteng Nyeruduk 01 |
| DK002 | Timbul | Injak-Injak Bumi Kav. 054 |
| DK003 | Topan | Jl. Topan Tiada Tara 002 |
| DK004 | Laysus | Jl. Angin Ribut Sekali Kav. 03 |
| DK005 | Parto | Jl. Kual Blok 90 Kav. V |
| DK006 | Tessi | Jl. Pojok-Pojok Sekali 90 |

| AlDok02 | KtDok | KpDok | TlDok |
|----------------|--------------|--------------|--------------|
| Ciledug | Jakarta | 12260 | 8563-5003 |
| Sabah | Malaysia | 69876 | 56-982 |
| Pejaten | Jakarta | 12580 | 789-3624 |
| Daan Mogot | Jakarta | 13690 | 569-5320 |
| Kuala Simpang | Jambi | 69860 | 987-4652 |
| Rawamangun | Jakarta | 56980 | 659-8672 |

Nama Tabel : **TarifDok**

| KdDok | TrfDok |
|--------------|---------------|
| DK001 | 50000 |
| DK002 | 50000 |
| DK003 | 50000 |
| DK004 | 50000 |
| DK005 | 50000 |
| DK006 | 50000 |

Nama Tabel : **Pasien**

| KdPsn | NmPsn | AlPsn01 |
|--------------|--------------|---------------------------|
| PS001 | Ruhayah | Jl. Panti Asuhan |
| PS002 | Syamsudin | Jl. KH.Muhasyim |
| PS003 | Suminah | Jl.Pesantren Al-Mamur |
| PS004 | Sumadi | Jl. Raya Ceger |
| PS005 | Rosyanti | Kampung Pondok Aren |
| PS006 | Bayhaqi | Jl.Kav.Setia Budi |
| PS007 | Rusmiati | Jl.M. Saidi |
| PS008 | Zulfikar | Jl.Ciledug Raya Gg.Shinta |
| PS009 | Muhaimin | Jl.Ulujami Raya Gg. Warga |
| PS010 | Simanjuntak | Jl.Ciptomangunkusumo |

| AlPsn02 | KtPsn | KpPsn | TlPsn |
|-----------------|--------------|--------------|--------------|
| Pondok Aren | Tangerang | 12270 | 731-7765 |
| Cilandak | Jakarta | 69876 | 765-9812 |
| Ciledug | Tangerang | 15156 | - |
| Pondok Aren | Tangerang | 15222 | 732-7415 |
| Pondok Kacang | Tangerang | 69860 | 732-5689 |
| Ciledug | Tangerang | 15155 | - |
| Petukangan Sel. | Jakarta | 12270 | 736-5691 |
| Ciledug | Tangerang | 15154 | - |
| Larangan | Tangerang | 12320 | 737-8055 |
| Ciledug | Tangerang | 15153 | 732-8888 |

Nama Tabel : **Resep**

| KdRsp | KdObat | Jumlah |
|-------|--------|--------|
| R001 | O002 | 20 |
| R002 | O006 | 15 |
| R003 | O006 | 13 |
| R004 | O001 | 19 |
| R005 | O008 | 56 |
| R006 | O005 | 52 |
| R007 | O009 | 100 |
| R008 | O006 | 32 |
| R009 | O006 | 65 |
| R010 | O003 | 40 |

Nama Tabel : **Servis**

| KdSakit | KdKln | KdDok | KdPrwt | Tarif |
|---------|-------|-------|--------|-------|
| S001 | K001 | DK001 | PR001 | 50000 |
| S002 | K002 | DK002 | PR002 | 50000 |
| S003 | K002 | DK002 | PR002 | 50000 |
| S004 | K002 | DK002 | PR002 | 50000 |
| S005 | K003 | DK003 | PR003 | 50000 |
| S006 | K004 | DK004 | PR004 | 50000 |
| S007 | K005 | DK005 | PR005 | 50000 |
| S008 | K006 | DK006 | PR006 | 50000 |
| S009 | K005 | DK005 | PR005 | 50000 |
| S010 | K005 | DK005 | PR005 | 50000 |

Nama Tabel : **Kunjung**

| KdPsn | KdSakit | KdRsp |
|-------|---------|-------|
| PS001 | S002 | R001 |
| PS002 | S006 | R002 |
| PS003 | S005 | R003 |
| PS004 | S001 | R004 |
| PS005 | S004 | R005 |
| PS006 | S007 | R006 |
| PS007 | S008 | R007 |
| PS008 | S010 | R008 |
| PS009 | S009 | R009 |
| PS010 | S003 | R010 |

- Buat relasi dengan nama sebagai berikut :
Tabel : **Layan**
Field : **NmSakit, NmKln, RngKln**
Tabel : **Datang**


Field : NmPsn, NmSakit, NmKln
 Tabel : Derita
 Field : NmPsn, NmSakit, NmObat, NmKln, NmDok
 Tabel : Bayar
 Field : NmPsn, NmSakit, NmObat, TrfDok, Tarif, Harga, Jumlah, Total = (TrfDok + Tarif) + (Harga*Jml) (*otomatis*)

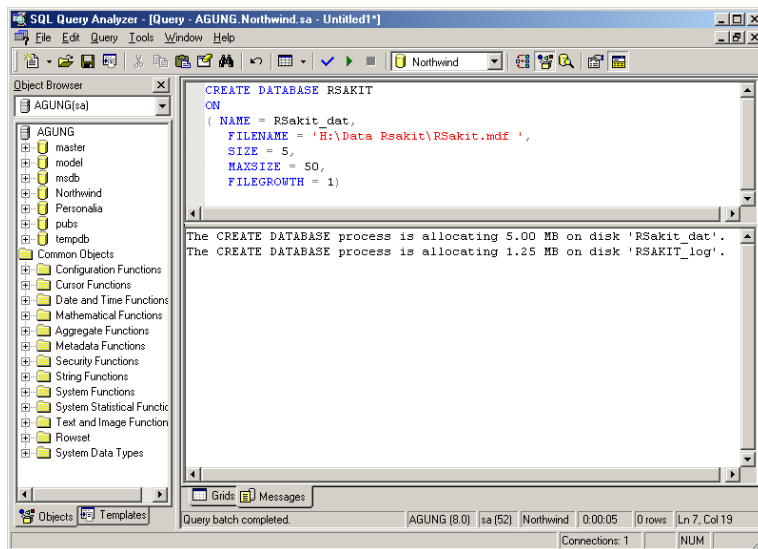
- Rubahlah semua harga pada tabel **Obat** dengan kenaikan sebesar 10%.
- Rubah pula **TrfDok** pada tabel **Bayar** dengan nama **Bayhaqi** dengan potongan sebesar 25%.
- Tampilkan isi *field* dengan mengubah judul pada tabel **Datang** sesuai dengan nama yang sesuai, sebagai berikut :
 - NmPsn = Nama Pasien
 - NmSakit = Nama Penyakit
 - NmKln = Nama Klinik

Penyelesaian kasus 2 :

1. Langkah pertama dalam menyelesaikan soal diatas adalah dengan membuat *database* sebagai tempat penampungan tabel-tabel. Setelah diketahui nama *database* yaitu **RSAKIT**, kemudian kita buat *database* tersebut dengan perintah T-SQL yaitu :

```
CREATE DATABASE RSAKIT
ON
(NAME = RSakit_dat,
FILENAME = 'H:\Data Rsakit\RSakit.mdf',
SIZE = 5,
MAXSIZE = 50,
FILEGROWTH = 1)
```

Setelah perintah tersebut dibuat, kemudian jalankan dengan tekan tombol **F5** atau klik icon **Execute** (). Hasilnya dapat dilihat pada gambar dibawah ini :



Gambar 5.35. Tampilan pembuatan *database* RSAKIT

2. Kemudian anda buat semua tabel yang diminta. Sebelum anda membuat tabel, sebaiknya definisikan nama *field*, tipe dan panjangnya. Pada contoh diatas, maka didapatkan definisi tabel sebagai berikut :

Nama Tabel : **Obat**

| Nama Field | Tipe | Panjang |
|------------|-------|---------|
| KdObat | Char | 4 |
| NmObat | Char | 20 |
| KetObat | Char | 30 |
| Harga | Float | |

Nama Tabel : **Penyakit**

| Nama Field | Tipe | Panjang |
|------------|------|---------|
| KdSakit | Char | 4 |
| NmSakit | Char | 20 |

Nama Tabel : **Klinik**

| Nama Field | Tipe | Panjang |
|------------|------|---------|
| KdKln | Char | 4 |
| NmKln | Char | 20 |
| RngKln | Char | 10 |

Nama Tabel : **Perawat**

| Nama Field | Tipe | Panjang |
|------------|------|---------|
| KdPrwt | Char | 5 |
| NmPrwt | Char | 20 |
| AlPrwt01 | Char | 30 |
| AlPrwt02 | Char | 30 |
| KtPrwt | Char | 20 |
| KpPrwt | Char | 5 |
| TlPrwt | Char | 15 |

Nama Tabel : **Dokter**

| Nama Field | Tipe | Panjang |
|------------|------|---------|
| KdDok | Char | 5 |
| NmDok | Char | 20 |
| AlDok01 | Char | 30 |
| AlDok02 | Char | 30 |
| KtDok | Char | 20 |
| KpDok | Char | 5 |
| TlDok | Char | 15 |

Nama Tabel : **TarifDok**

| Nama Field | Tipe | Panjang |
|------------|-------|---------|
| KdDok | Char | 5 |
| TrfDok | Float | |

Nama Tabel : **Pasien**

| Nama Field | Tipe | Panjang |
|------------|------|---------|
| KdPsn | Char | 5 |
| NmPsn | Char | 20 |
| AlPsn01 | Char | 30 |
| AlPsn02 | Char | 30 |
| KtPsn | Char | 20 |
| KpPsn | Char | 5 |
| TlPsn | Char | 15 |

Nama Tabel : **Resep**

| Nama Field | Tipe | Panjang |
|------------|-------|---------|
| KdRsp | Char | 4 |
| KdObat | Char | 4 |
| Jumlah | Float | |

Nama Tabel : **Servis**

| Nama Field | Tipe | Panjang |
|------------|-------|---------|
| KdSakit | Char | 4 |
| KdKln | Char | 4 |
| KdDok | Char | 5 |
| KdPrwt | Char | 5 |
| Tarif | Float | |

Nama Tabel : **Kunjung**

| Nama Field | Tipe | Panjang |
|------------|------|---------|
| KdPsn | Char | 5 |
| KdSakit | Char | 4 |
| KdRsp | Char | 4 |

3. Setelah didapatkan definisi tabel, kemudian buatlah tabel tersebut dengan menggunakan T-SQL. Untuk pembuatan tabel, anda dapat membuat satu persatu tabel dengan T-SQL atau membuat tabel tersebut sekaligus dalam satu T-SQL. Dalam buku ini penulis akan membuat sekaligus dalam satu buah T-SQL untuk mempersingkat penulisan. Untuk membuat tabel-tabel tersebut, anda dapat menuliskan perintah berikut ini :

```
USE RSAKIT
GO
CREATE TABLE Obat
(KdObat Char (4) NOT NULL,
NmObat Char (20) NOT NULL,
KetObat Char (30) NOT NULL,
Harga Float NOT NULL DEFAULT 0,
PRIMARY KEY (KdObat))
```

```
CREATE TABLE Penyakit
(KdSakit Char (4) NOT NULL,
NmSakit Char (20) NOT NULL,
PRIMARY KEY (KdSakit))
```

```
CREATE TABLE Klinik
(KdKln Char (4) NOT NULL,
NmKln Char (20) NOT NULL,
```

**RngKln Char (10) NOT NULL,
PRIMARY KEY (Kdkln))**

**CREATE TABLE Perawat
(KdPrwt Char (5) NOT NULL,
NmPrwt Char (20) NOT NULL,
AlPrwt01 Char (30) NOT NULL,
AlPrwt02 Char (30) NOT NULL,
KtPrwt Char (20) NOT NULL,
KpPrwt Char (5) NOT NULL,
TlPrwt Char (15) NOT NULL,
PRIMARY KEY (KdPrwt))**

**CREATE TABLE Dokter
(KdDok Char (5) NOT NULL,
NmDok Char (20) NOT NULL,
AlDok01 Char (30) NOT NULL,
AlDok02 Char (30) NOT NULL,
KtDok Char (20) NOT NULL,
KpDok Char (5) NOT NULL,
TlDok Char (15) NOT NULL,
PRIMARY KEY (KdDok))**

**CREATE TABLE TarifDok
(KdDok Char (5) NOT NULL,
TrfDok Float NOT NULL DEFAULT 0)**

**CREATE TABLE Pasien
(KdPsn Char (5) NOT NULL,
NmPsn Char (20) NOT NULL,
AlPsn01 Char (30) NOT NULL,
AlPsn02 Char (30) NOT NULL,
KtPsn Char (20) NOT NULL,
KpPsn Char (5) NOT NULL,
TlPsn Char (15) NOT NULL,
PRIMARY KEY (KdPsn))**


**CREATE TABLE Resep
(KdRsp Char (4) NOT NULL,
KdObat Char (4) NOT NULL,
Jumlah Float NOT NULL DEFAULT 0,
PRIMARY KEY (KdRsp))**

```
CREATE TABLE Servis
(KdSakit Char (4) NOT NULL,
KdKln Char (4) NOT NULL,
KdDok Char (5) NOT NULL,
KdPrwt Char (5) NOT NULL,
Tarif Float NOT NULL DEFAULT 0)
```

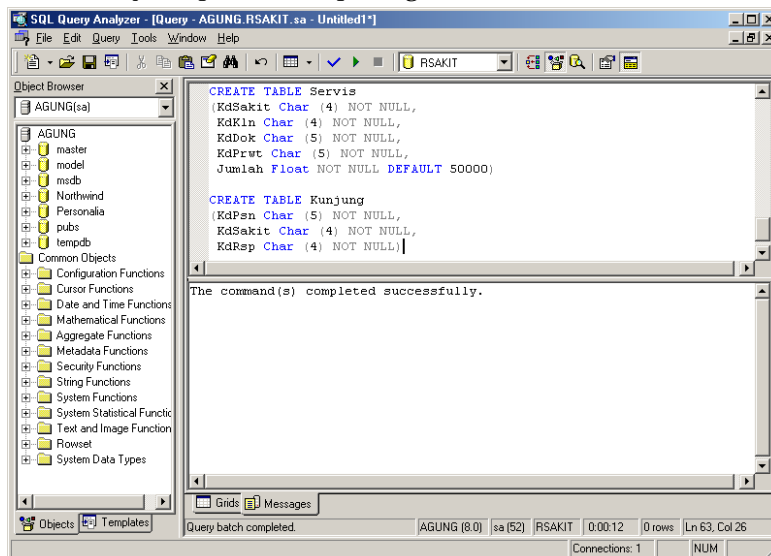
```
CREATE TABLE Kunjung
(KdPsn Char (5) NOT NULL,
KdSakit Char (4) NOT NULL,
KdRsp Char (4) NOT NULL)
```

Untuk pembuatan **DEFAULT** o pada pembuatan tabel diatas sangat disarankan, karena isi pada *field* tersebut adalah dengan tipe **float** yang akan digunakan untuk kalkulasi matematis. Bila diisi dengan **NULL**, maka *field* tersebut tidak dapat digunakan untuk kalkulasi matematis.

Untuk *field* yang berupa mata uang, anda dapat pula menggunakan tipe **Money**.

Setelah perintah tersebut dibuat, kemudian jalankan dengan tekan tombol **F5** atau klik icon **Execute** ().

Hasilnya dapat dilihat pada gambar dibawah ini :




Gambar 5.36. Tampilan pembuatan tabel-tabel dengan T-SQL

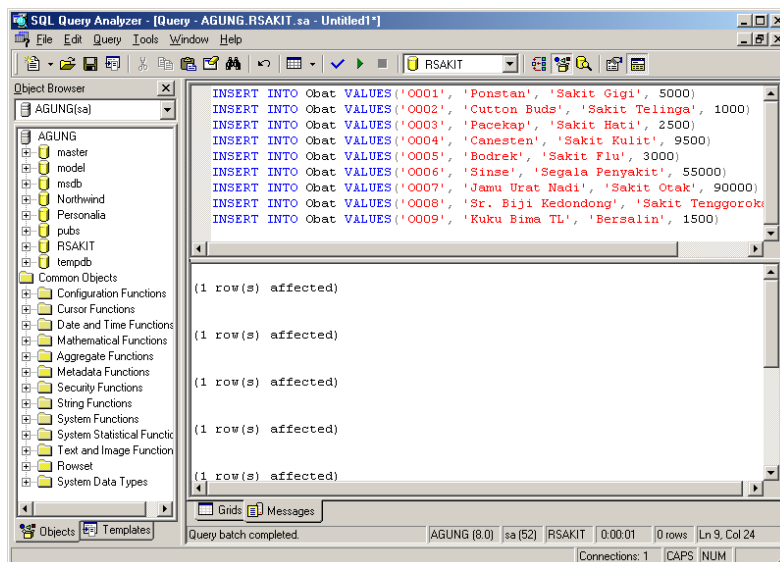
4. Langkah selanjutnya adalah mengisi tabel-tabel tersebut dengan data yang tersedia. Untuk mengisi tabel tersebut diatas, anda dapat mengikuti mengisi dengan 2 (dua) cara, yaitu :

a. Dengan **INSERT INTO**

Untuk pengisian tabel dengan **INSERT INTO**, anda dapat menuliskan perintah sebagai berikut :

```
INSERT INTO Obat VALUES('O001', 'Ponstan', 'Sakit Gigi', 5000)
INSERT INTO Obat VALUES('O002', 'Cutton Buds', 'Sakit Telinga',
1000)
INSERT INTO Obat VALUES('O003', 'Pacekap', 'Sakit Hati', 2500)
INSERT INTO Obat VALUES('O004', 'Canesten', 'Sakit Kulit', 9500)
INSERT INTO Obat VALUES('O005', 'Bodrek', 'Sakit Flu', 3000)
INSERT INTO Obat VALUES('O006', 'Sinse', 'Segala Penyakit', 55000)
INSERT INTO Obat VALUES('O007', 'Jamu Urat Nadi', 'Sakit Otak',
90000)
INSERT INTO Obat VALUES('O008', 'Sr. Biji Kedondong', 'Sakit
Tenggorokan', 2000)
INSERT INTO Obat VALUES('O009', 'Kuku Bima TL', 'Bersalin', 1500)
```

Langkah selanjutnya, jalankan perintah diatas dengan tekan tombol **F5** atau klik icon **Execute** (). Lihat gambar dibawah ini :



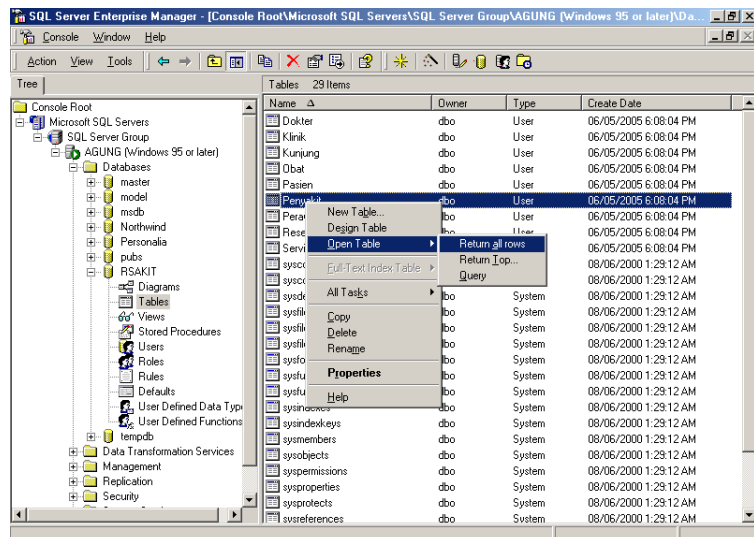
Gambar 5.37. Tampilan masukan data ke tabel dengan INSERT INTO

b. Dengan **Enterprise Manager**

Untuk memasukkan data dengan **Enterprise Manager**, dapat diikuti langkah berikut ini :

- Anda masuk ke area tabel dengan cara klik tanda (+) pada bagian **Tree** yang terdapat pada **Enterprise Manager**. Misalkan anda akan mengisi tabel yang anda buat pada *database* **RSAKIT**, anda harus masuk ke area tabel yang terdapat pada *database* **RSAKIT**. Kemudian anda klik kanan pada tabel tersebut, sehingga tampil menu *pop-up*. Setelah tampil menu *pop-up*, anda klik **Open Table** dan pilih **Return all rows**.

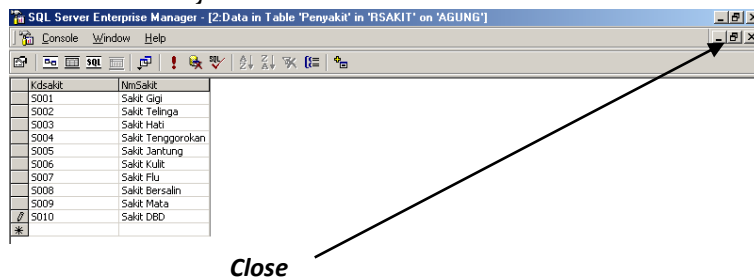
Misalkan anda akan mengisi tabel Penyakit. Anda klik kanan pada tabel penyakit hingga tampil menu *pop-up*. Kemudian anda klik menu **Open Table** dan pilih atau klik **Return all rows**.



Gambar 5.38. Tampilan menu *pop-up* klik kanan pada tabel

- Setelah tampil *form* isian tabel, kemudian isilah satu persatu data hingga selesai. Dalam pengisian ini, tabel akan secara otomatis menyimpan data *record* yang

anda ketik bila kursor pindah ke *record* kosong di bawahnya.

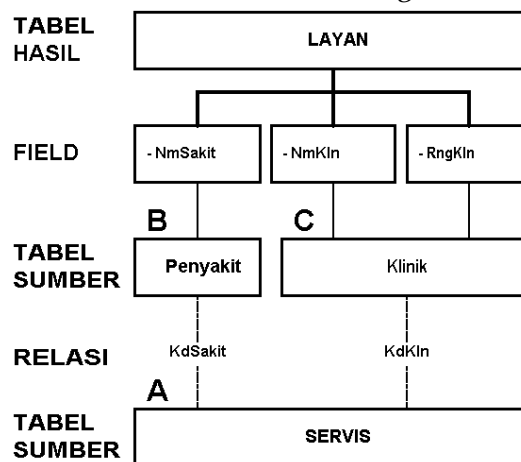


Gambar 5.39. Tampilan form pengisian tabel Penyakit

- Apabila semua data telah dimasukan, anda dapat menutup *form* tersebut dengan mengklik tombol **Close** () di pojok kanan atas. (lihat gambar 5)

Isilah semua tabel yang kosong dengan data yang telah disediakan. Mengenai cara pengisian tabel apakah dengan **Enterprise Manager** atau dengan **INSERT INTO**, terserah anda yang mana yang paling mudah dan cepat menurut anda.

5. Setelah semua tabel diisi dengan data, langkah selanjutnya adalah membuat relasi antar tabel. Dari masalah diatas, maka dapat digambarkan relasi antar tabel sebagai berikut :

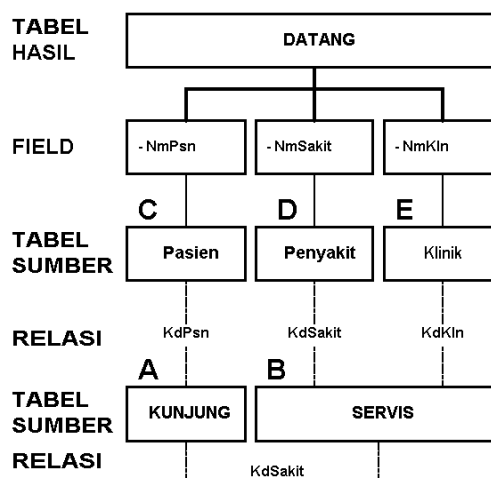


Gambar 5.40. Tampilan relasi dengan hasil tabel Layan

Keterangan :

- Tabel **Layan** merupakan tabel relasi yang akan berisi *field* dengan nama **NmSakit**, **NmKln** dan **RngKln**.
- Lalu anda lanjutkan dengan mencari tabel sumber dari *field-field* tersebut. Adapun *field* **NmSakit** diambil dari tabel **Penyakit**, sedangkan *field* **NmKln** dan **RngKln** diambil dari tabel **Klinik**. Untuk menghubungkan antara tabel **Penyakit** dan tabel **Klinik**, dibutuhkan tabel penghubung yang didalamnya berisi kode-kode yang unik (tidak dobel) dari kedua tabel tersebut, atau yang dikenal dengan *primary key*. Tabel **Penyakit** mempunyai **KdSakit** dan tabel **Klinik** mempunyai **KdKln**. Setelah dicari, ternyata tabel **Servis** memenuhi kriteria, yakni didalamnya terdapat *field* dengan nama **KdSakit** dan **KdKln**.
- Kemudian anda letakan tabel-tabel tersebut (**Servis**, **Penyakit** dan **Klinik**) kedalam alias dengan tabel **Servis** didalam alias **A**, **Penyakit** didalam alias **B** dan **Klinik** didalam alias **C**.
- Barulah anda buat hubungan SQL-nya sebagai berikut :
Servis.KdSakit= Penyakit.KdSakit → a.KdSakit = b.KdSakit
Servis.KdKlinik = Klinik.KdKln → a.KdKln = c.KdKln

Adapun bentuk perintah SQL-nya, dapat anda lihat pada no. 6.

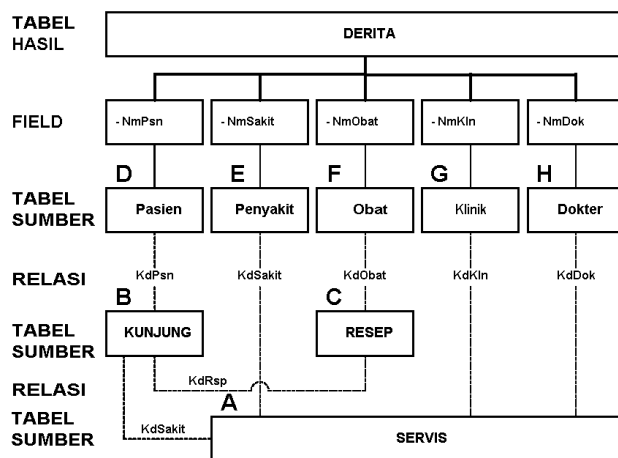


Gambar 5.41. Tampilan relasi dengan hasil tabel Datang

Keterangan :

- Tabel **Datang** merupakan tabel relasi yang akan berisi *field* dengan nama **NmPsn**, **NmSakit** dan **NmKln**.
- Lalu anda lanjutkan dengan mencari tabel sumber dari *field-field* tersebut. Adapun *field* **NmPasien** diambil dari tabel **Pasien**, *field* **NmSakit** diambil dari tabel **Penyakit**, dan *field* **NmKln** diambil dari tabel **Klinik**. Untuk menghubungkan antara tabel **Pasien**, tabel **Penyakit** dan tabel **Klinik**, dibutuhkan tabel penghubung yang didalamnya berisi kode-kode yang unik (tidak dobel) dari kedua tabel tersebut, atau yang dikenal dengan *primary key*. Tabel **Pasien** mempunyai *field* **KdPsn**, tabel **Penyakit** mempunyai *field* **KdSakit** dan tabel **Klinik** mempunyai *field* **KdKln**.
- Dari ketiga *field* tersebut, carilah tabel yang didalamnya mempunyai ketiga *field* tersebut (**KdPsn**, **KdSakit** dan **KdKln**). Setelah dicari, ternyata tabel **KdPsn** berada pada tabel **Kunjung**, **KdSakit** dan **KdKln** berada pada tabel **Servis**. Kemudian anda cari *field* yang dapat digunakan untuk menghubungkan antara tabel **Kunjung** dengan tabel **Servis**, jika dilihat maka *field* **KdSakit** yang dapat digunakan sebagai penghubung.
- Kemudian anda letakan tabel-tabel tersebut (**Kunjung**, **Servis**, **Pasien**, **Penyakit** dan **Klinik**) kedalam alias dengan tabel **Kunjung** didalam alias **A**, **Servis** didalam alias **B**, **Pasien** didalam alias **C**, **Penyakit** didalam alias **D** dan **Klinik** didalam alias **E**.
- Barulah anda buat hubungan SQL-nya sebagai berikut :
Kunjung.KdSakit = Servis.KdSakit → **a.KdSakit = b.KdSakit**
Kunjung.KdPsn = Pasien.KdPsn → **a.KdKln = c.KdKln**
Servis.KdSakit = Penyakit.KdSakit → **b.KdSakit = d.KdSakit**
Servis.KdKln = Klinik.KdKln → **b.KdKln = e.KdKln**

Adapun bentuk perintah SQL-nya, dapat anda lihat pada no. 6.



Gambar 5.42. Tampilan relasi dengan hasil tabel Derita

Keterangan :

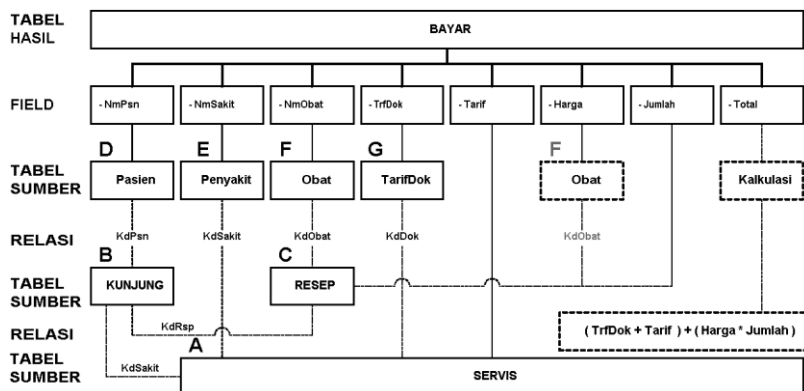
- Tabel **Derita** merupakan tabel relasi yang akan berisi *field* dengan nama **NmPsn**, **NmSakit**, **NmObat**, **NmKln** dan **NmDok**.
- Lalu anda lanjutkan dengan mencari tabel sumber dari *field-field* tersebut. Adapun *field NmPasien* diambil dari tabel **Pasien**, *field NmSakit* diambil dari tabel **Penyakit**, *field NmObat* diambil dari tabel **Obat**, *field NmKln* diambil dari tabel **Klinik** dan *field NmDok* diambil dari tabel **Dokter**. Untuk menghubungkan antara tabel **Pasien**, tabel **Penyakit**, tabel **Obat**, tabel **Klinik** dan tabel **Dokter**, dibutuhkan tabel penghubung yang didalamnya berisi kode-kode yang unik (tidak doble) dari kedua tabel tersebut, atau yang dikenal dengan *primary key*. Tabel **Pasien** mempunyai *field KdPsn*, tabel **Penyakit** mempunyai *field KdSakit*, tabel **Obat** mempunyai *field KdObat*, tabel **Klinik** mempunyai *field KdKln*, dan tabel **Dokter** mempunyai *field KdDok*.
- Dari kelima *field* tersebut, carilah tabel yang didalamnya mempunyai kelima *field* tersebut (**KdPsn**, **KdSakit**, **KdObat**, **KdKln** dan **KdDok**). Setelah dicari, ternyata tabel **KdPsn** berada pada tabel **Kunjung**, **KdObat** berada pada tabel **Resep**, dan **KdSakit**, **KdKln** dan **KdDok**

berada pada tabel **Servis**. Kemudian anda cari *field* yang dapat digunakan untuk menghubungkan antara tabel **Kunjung**, tabel **Resep** dengan tabel **Servis**. Jika dilihat maka *field* **KdResep** dapat digunakan sebagai penghubung antara tabel **Kunjung** dan **Resep**, serta **KdSakit** dapat digunakan sebagai penghubung antara tabel **Kunjung** dengan tabel **Servis**.

- Kemudian anda letakan tabel-tabel tersebut (**Servis**, **Kunjung**, **Resep**, **Pasien**, **Penyakit**, **Obat**, **Klinik**, dan **Dokter**) kedalam alias dengan tabel **Servis** didalam alias **A**, **Kunjung** didalam alias **B**, **Resep** didalam alias **C**, **Pasien** didalam alias **D**, **Penyakit** didalam alias **E**, **Obat** didalam alias **F**, **Klinik** didalam alias **G** dan **Dokter** didalam alias **H**.
- Barulah anda buat hubungan SQL-nya sebagai berikut :
Servis.KdSakit = Kunjung.KdSakit → **a.KdSakit = b.KdSakit**
Kunjung.KdRsp = Resep.KdRsp → **b.KdRsp = c.KdRsp**
Kunjung.KdPsn = Pasien.KdPsn → **b.KdPsn = d.KdPsn**
Servis.KdSakit = Penyakit.KdSakit → **a.KdSakit = e.KdSakit**
Resep.KdObat = Obat.KdObat → **c.KdObat = f.KdObat**
Servis.KdKln = Klinik.KdKln → **a.KdKln = g.KdKln**
Servis.KdDok = Dokter.Kddok → **a.KdDok = h.KdDok**

Adapun bentuk perintah SQL-nya, dapat anda lihat pada

no. 6.



Gambar 5.43. Tampilan relasi dengan hasil tabel Bayar

Keterangan :

- Tabel **Bayar** merupakan tabel relasi yang akan berisi *field* dengan nama **NmPsn**, **NmSakit**, **NmObat**, **TrfDok**, **Tarif**, **Harga**, **Jumlah** dan **Total**.
- Lalu anda lanjutkan dengan mencari tabel sumber dari *field-field* tersebut. Adapun *field NmPsn* diambil dari tabel **Pasien**, *field NmSakit* diambil dari tabel **Penyakit**, *field NmObat* dan **Harga** diambil dari tabel **Obat**, *field TrfDok* diambil dari tabel **TarifDok**, *field Tarif* diambil dari tabel **Servis**, *field Jumlah* diambil dari tabel **Resep** dan *field Total* berisi hasil kalkulasi dari *field* yang telah direlasikan $[(TrfDok+Tarif) + (Harga*Jumlah)]$.
- Untuk menghubungkan antara tabel **Pasien**, tabel **Penyakit**, tabel **Obat**, dan tabel **TarifDok**, dibutuhkan tabel penghubung yang didalamnya berisi kode-kode yang unik (tidak dobel) dari kedua tabel tersebut, atau yang dikenal dengan *primary key*. Tabel **Pasien** mempunyai *field KdPsn*, tabel **Penyakit** mempunyai *field KdSakit*, tabel **Obat** mempunyai *field KdObat* dan tabel **TarifDok** mempunyai *field KdDok*.
- Dari keempat *field* tersebut, carilah tabel yang didalamnya mempunyai kelima *field* tersebut (**KdPsn**, **KdSakit**, **KdObat**, dan **KdDok**). Setelah dicari, ternyata tabel **KdPsn** berada pada tabel **Kunjung**, **Kdsakit** dan **KdDok** berada pada tabel **Servis** dan **KdObat** berada pada tabel **Resep**. Kemudian anda cari *field* yang dapat digunakan untuk menghubungkan antara tabel **Kunjung**, tabel **Resep** dengan tabel **Servis**. Jika dilihat maka *field KdResep* dapat digunakan sebagai penghubung antara tabel **Kunjung** dan **Resep**, serta **KdSakit** dapat digunakan sebagai penghubung antara tabel **Kunjung** dengan tabel **Servis**.
- Kemudian anda letakkan tabel-tabel tersebut (**Servis**, **Kunjung**, **Resep**, **Pasien**, **Penyakit**, **Obat**, dan **TarifDok**) kedalam alias dengan tabel **Servis** didalam alias **A**, **Kunjung** didalam alias **B**, **Resep** didalam alias **C**,

Pasien didalam alias **D**, **Penyakit** didalam alias **E**, **Obat** didalam alias **F**, dan **TarifDok** didalam alias **G**.

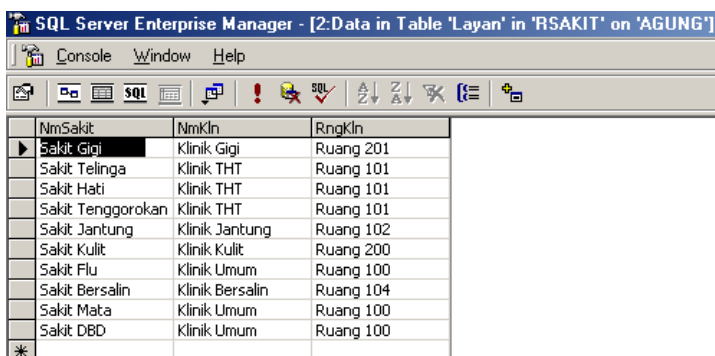
- Barulah anda buat hubungan SQL-nya sebagai berikut :
Servis.KdSakit = Kunjung.KdSakit → a.KdSakit = b.KdSakit
Kunjung.Kdrsp = Resep.KdRsp → b.KdRsp = c.KdRsp
Kunjung.KdPsn = Pasien.KdPsn → b.KdPsn = d.KdPsn
Servis.KdSakit = Penyakit.KdSakit → a.KdSakit = e.KdSakit
Resep.KdObat = Obat.KdObat → c.KdObat = f.KdObat
Servis.KdDok = Dokter.Kddok → a.KdDok = g.KdDok

Adapun bentuk perintah SQL-nya, dapat anda lihat pada no. 6.

6. Buatlah perintah T-SQL nya untuk mengaplikasikan gambar relasi tabel tersebut diatas, dengan ketentuan hubungan seperti yang telah ditetapkan pada no. 5.
 Pada gambar 6. dibuat perintah SQL nya sebagai berikut :

```
SELECT b.NmSakit, c.NmKln, c.RngKln
INTO Layan
FROM Servis a, Penyakit b, Klinik c
WHERE a.KdSakit = b.KdSakit AND
      a.KdKln = c.KdKln
```

Kemudian tekan tombol **F5** pada *keyboard* untuk menjalankan perintah tersebut. Hasilnya dapat anda lihat pada gambar dibawah ini :



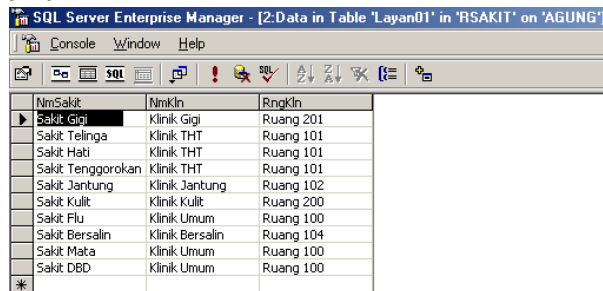
| NmSakit | NmKln | RngKln |
|-------------------|-----------------|-----------|
| Sakit Gigi | Klinik Gigi | Ruang 201 |
| Sakit Telinga | Klinik THT | Ruang 101 |
| Sakit Hati | Klinik THT | Ruang 101 |
| Sakit Tenggorokan | Klinik THT | Ruang 101 |
| Sakit Jantung | Klinik Jantung | Ruang 102 |
| Sakit Kulit | Klinik Kulit | Ruang 200 |
| Sakit Flu | Klinik Umum | Ruang 100 |
| Sakit Bersalin | Klinik Bersalin | Ruang 104 |
| Sakit Mata | Klinik Umum | Ruang 100 |
| Sakit DBD | Klinik Umum | Ruang 100 |

Gambar 5.44. Tampilan tabel Layan

Atau dapat pula dengan menggunakan **INNER JOIN**, dengan menuliskan perintah sebagai berikut :

```
SELECT b.NmSakit, c.NmKln, c.RngKln
INTO Layan01
FROM Servis a INNER JOIN Penyakit b
ON a.KdSakit = b.KdSakit
INNER JOIN Klinik c
ON a.KdKln = c.KdKln
```

Kemudian tekan tombol **F5** pada *keyboard* untuk menjalankan perintah tersebut. Hasilnya dapat anda lihat pada gambar 5.45 dibawah ini :



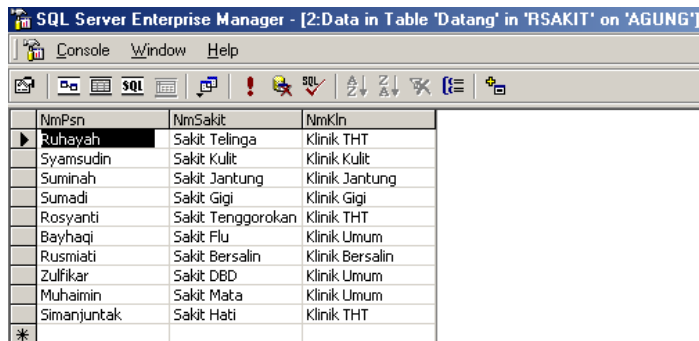
| NmSakit | NmKln | RngKln |
|-------------------|-----------------|-----------|
| Sakit Gigi | Klinik Gigi | Ruang 201 |
| Sakit Telinga | Klinik THT | Ruang 101 |
| Sakit Hati | Klinik THT | Ruang 101 |
| Sakit Tenggorokan | Klinik THT | Ruang 101 |
| Sakit Jantung | Klinik Jantung | Ruang 102 |
| Sakit Kulit | Klinik Kulit | Ruang 200 |
| Sakit Flu | Klinik Umum | Ruang 100 |
| Sakit Bersalin | Klinik Bersalin | Ruang 104 |
| Sakit Mata | Klinik Umum | Ruang 100 |
| Sakit DBD | Klinik Umum | Ruang 100 |

Gambar 5.45. Tampilan tabel Layan01 hasil penggabungan dengan INNER JOIN

Pada gambar diatas dibuat perintah SQL nya sebagai berikut :

```
SELECT c.NmPsn, d.NmSakit, e.NmKln
INTO Datang
FROM Kunjung a, Servis b, Pasien c, Penyakit d, Klinik e
WHERE a.KdSakit = b.KdSakit AND
a.KdPsn = c.KdPsn AND
b.KdSakit = d.KdSakit AND
b.KdKln = e.KdKln
```

Kemudian tekan tombol **F5** pada *keyboard* untuk menjalankan perintah tersebut. Hasilnya dapat anda lihat pada gambar 5.46 dibawah ini :



SQL Server Enterprise Manager - [2:Data in Table 'Datang' in 'RSAKIT' on 'AGUNG']

| NmPsn | NmSakit | NmKln |
|-------------|-------------------|-----------------|
| Ruhayah | Sakit Telinga | Klinik THT |
| Syamsudin | Sakit Kulit | Klinik Kulit |
| Suminah | Sakit Jantung | Klinik Jantung |
| Sumadi | Sakit Gigi | Klinik Gigi |
| Rosyanti | Sakit Tenggorokan | Klinik THT |
| Bayhaqi | Sakit Flu | Klinik Umum |
| Rusmiati | Sakit Bersalin | Klinik Bersalin |
| Zulfikar | Sakit DBD | Klinik Umum |
| Muhaimin | Sakit Mata | Klinik Umum |
| Simanjuntak | Sakit Hati | Klinik THT |

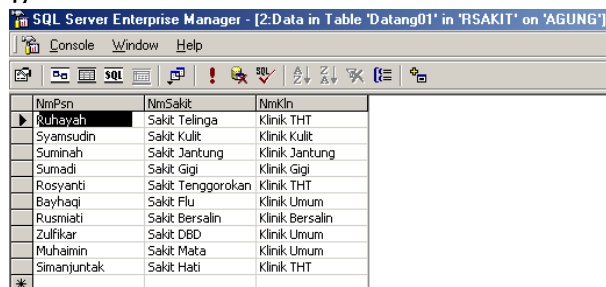
Gambar 5.46. Tampilan tabel Datang

Atau dapat pula dengan menggunakan **INNER JOIN**, dengan menuliskan perintah sebagai berikut :

```

SELECT c.NmPsn, d.NmSakit, e.NmKln
INTO Datango1
FROM Kunjung a INNER JOIN Servis b
    ON a.KdSakit = b.KdSakit
INNER JOIN Pasien c
    ON a.KdPsn = c.KdPsn
INNER JOIN Penyakit d
    ON b.KdSakit = d.KdSakit
INNER JOIN Klinik e
    ON b.KdKln = e.KdKln
  
```

Kemudian tekan tombol **F5** pada *keyboard* untuk menjalankan perintah tersebut. Hasilnya dapat anda lihat pada gambar 5.47 dibawah ini :



SQL Server Enterprise Manager - [2:Data in Table 'Datango1' in 'RSAKIT' on 'AGUNG']

| NmPsn | NmSakit | NmKln |
|-------------|-------------------|-----------------|
| Ruhayah | Sakit Telinga | Klinik THT |
| Syamsudin | Sakit Kulit | Klinik Kulit |
| Suminah | Sakit Jantung | Klinik Jantung |
| Sumadi | Sakit Gigi | Klinik Gigi |
| Rosyanti | Sakit Tenggorokan | Klinik THT |
| Bayhaqi | Sakit Flu | Klinik Umum |
| Rusmiati | Sakit Bersalin | Klinik Bersalin |
| Zulfikar | Sakit DBD | Klinik Umum |
| Muhaimin | Sakit Mata | Klinik Umum |
| Simanjuntak | Sakit Hati | Klinik THT |

Gambar 5.47. Tampilan tabel Datango1 hasil penggabungan dengan INNER JOIN

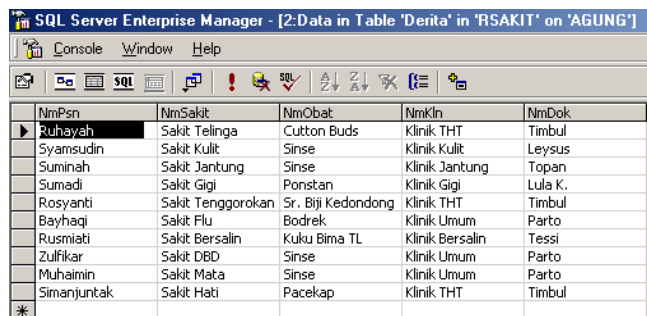
Pada gambar diatas. dibuat perintah SQL nya sebagai berikut :

```

SELECT d.NmPsn, e.NmSakit, f.NmObat, g.NmKln,
h.NmDok
INTO Derita
FROM Servis a, Kunjung b, Resep c, Pasien d, Penyakit e,
Obat f, Klinik g, Dokter h
WHERE a.KdSakit = b.KdSakit AND
b.KdRsp = c.KdRsp AND
b.KdPsn = d.KdPsn AND
a.KdSakit = e.KdSakit AND
c.KdObat = f.KdObat AND
a.KdKln = g.KdKln AND
a.KdDok = h.KdDok

```

Kemudian tekan tombol **F5** pada *keyboard* untuk menjalankan perintah tersebut. Hasilnya dapat anda lihat pada gambar 5.48 dibawah ini :



| NmPsn | NmSakit | NmObat | NmKln | NmDok |
|-------------|-------------------|--------------------|-----------------|---------|
| Ruhayah | Sakit Telinga | Cutton Buds | Klinik THT | Timbul |
| Syamsudin | Sakit Kulit | Sinse | Klinik Kulit | Leysus |
| Suminah | Sakit Jantung | Sinse | Klinik Jantung | Topan |
| Sumadi | Sakit Gigi | Ponstan | Klinik Gigi | Lula K. |
| Rosyanti | Sakit Tenggorokan | Sr. Biji Kedondong | Klinik THT | Timbul |
| Bayhaqi | Sakit Flu | Bodrek | Klinik Umum | Parto |
| Rusmiati | Sakit Bersalin | Kuku Bima TL | Klinik Bersalin | Tessi |
| Zulfikar | Sakit DBD | Sinse | Klinik Umum | Parto |
| Muhaimin | Sakit Mata | Sinse | Klinik Umum | Parto |
| Simanjuntak | Sakit Hati | Pacekap | Klinik THT | Timbul |

Gambar 5.48. Tampilan tabel Derita

Atau dapat pula dengan menggunakan **INNER JOIN**, dengan menuliskan perintah sebagai berikut :

```

SELECT d.NmPsn, e.NmSakit, f.NmObat, g.NmKln,
h.NmDok
INTO Derita01
FROM Servis a INNER JOIN Kunjung b
ON a.KdSakit = b.KdSakit

```

INNER JOIN Resep c
 ON b.KdRsp = c.KdRsp
INNER JOIN Pasien d
 ON b.KdPsn = d.KdPsn
INNER JOIN Penyakit e
 ON a.KdSakit = e.KdSakit
INNER JOIN Obat f
 ON c.KdObat = f.KdObat
INNER JOIN Klinik g
 ON a.KdKln = g.KdKln
INNER JOIN Dokter h
 ON a.KdDok = h.KdDok

Kemudian tekan tombol **F5** pada *keyboard* untuk menjalankan perintah tersebut. Hasilnya dapat anda lihat pada gambar 5.49 dibawah ini :

| NmPsn | NmSakit | NmObat | NmKln | NmDok |
|-------------|-------------------|--------------------|-----------------|---------|
| Ruhayah | Sakit Telinga | Cutton Buds | Klinik THT | Timbul |
| Syamsudin | Sakit Kulit | Sinse | Klinik Kulit | Leysus |
| Suminah | Sakit Jantung | Sinse | Klinik Jantung | Topan |
| Sumadi | Sakit Gigi | Ponstan | Klinik Gigi | Lula K. |
| Rosyanti | Sakit Tenggorokan | Sr. Biji Kedondong | Klinik THT | Timbul |
| Bayhaqi | Sakit Flu | Bodrek | Klinik Umum | Parto |
| Rusmiati | Sakit Bersalin | Kuku Bima TL | Klinik Bersalin | Tessi |
| Zulfikar | Sakit DBD | Sinse | Klinik Umum | Parto |
| Muhalimin | Sakit Mata | Sinse | Klinik Umum | Parto |
| Simanjuntak | Sakit Hati | Pacekap | Klinik THT | Timbul |

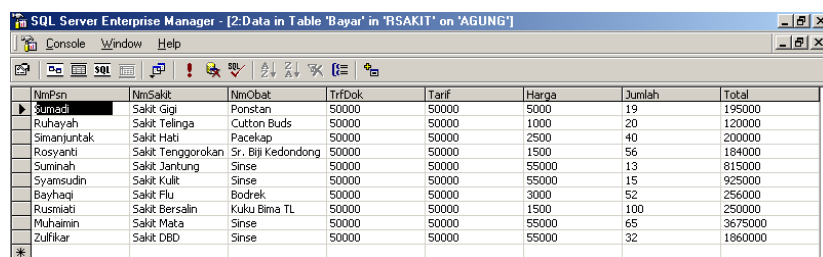
Gambar 5.49. Tampilan tabel Derita01 hasil penggabungan dengan INNER JOIN

Pada gambar diatas. dibuat perintah SQL nya sebagai berikut :

SELECT d.NmPsn, e.NmSakit, f.NmObat, g.TrfDok,
 a.Tarif, f.Harga, c.Jumlah, Total = (g.TrfDok+a.Tarif) +
 (f.Harga*c.Jumlah)
INTO Bayar
FROM Servis a, Kunjung b, Resep c, Pasien d, Penyakit e,
 Obat f, TarifDok g
WHERE a.KdSakit = b.KdSakit **AND**

b.KdRsp = c.KdRsp AND
 b.KdPsn = d.KdPsn AND
 a.KdSakit = e.KdSakit AND
 c.KdObat = f.KdObat AND
 a.KdDok = g.KdDok

Kemudian tekan tombol **F5** pada *keyboard* untuk menjalankan perintah tersebut. Hasilnya dapat anda lihat pada gambar 5.50 dibawah ini :



| NmPsn | NmSakit | NmObat | TrfDok | Tarif | Harga | Jumlah | Total |
|-------------|-------------------|--------------------|--------|-------|-------|--------|---------|
| Sumadi | Sakit Gigi | Ponstan | 50000 | 50000 | 5000 | 19 | 195000 |
| Ruhayah | Sakit Telinga | Cutton Buds | 50000 | 50000 | 1000 | 20 | 120000 |
| Simanjuntak | Sakit Hati | Pacekap | 50000 | 50000 | 2500 | 40 | 200000 |
| Rosyanti | Sakit Tenggorokan | Sr. Biji Kedondong | 50000 | 50000 | 1500 | 56 | 184000 |
| Suminah | Sakit Janbung | Sinse | 50000 | 50000 | 55000 | 13 | 815000 |
| Syamsudin | Sakit Kulit | Sinse | 50000 | 50000 | 55000 | 15 | 925000 |
| Bayhadri | Sakit Flu | Bofhrek | 50000 | 50000 | 3000 | 52 | 256000 |
| Rusmiati | Sakit Bersalin | Kuku Bima TL | 50000 | 50000 | 1500 | 100 | 250000 |
| Muhamin | Sakit Mata | Sinse | 50000 | 50000 | 55000 | 65 | 3675000 |
| Zulfikar | Sakit DBD | Sinse | 50000 | 50000 | 55000 | 32 | 1860000 |

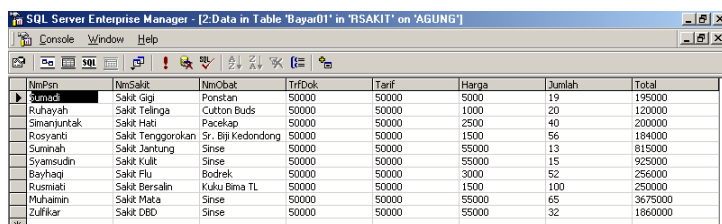
Gambar 5.50. Tampilan tabel Bayar

Atau dapat pula dengan menggunakan **INNER JOIN**, dengan menuliskan perintah sebagai berikut :

```

SELECT d.NmPsn, e.NmSakit, f.NmObat, g.TrfDok,
a.Tarif, f.Harga, c.Jumlah, Total = (g.TrfDok+a.Tarif) +
(f.Harga*c.Jumlah)
INTO Bayar01
FROM Servis a INNER JOIN Kunjung b
    ON a.KdSakit = b.KdSakit
INNER JOIN Resep c
    ON b.KdRsp = c.KdRsp
INNER JOIN Pasien d
    ON b.KdPsn = d.KdPsn
INNER JOIN Penyakit e
    ON a.KdSakit = e.KdSakit
INNER JOIN Obat f
    ON c.KdObat = f.KdObat
INNER JOIN TarifDok g
    ON a.KdDok = g.KdDok
  
```

Kemudian tekan tombol **F5** pada *keyboard* untuk menjalankan perintah tersebut. Hasilnya dapat anda lihat pada gambar 5.51 dibawah ini :



| NmPsn | NmSakit | NmObat | TrfDok | Tarif | Harga | Jumlah | Total |
|-------------|-------------------|--------------------|--------|-------|-------|--------|---------|
| Bumadi | Sakit Gigi | Ponstan | 50000 | 50000 | 5000 | 19 | 195000 |
| Ruhayah | Sakit Telinga | Cutton Buds | 50000 | 50000 | 1000 | 20 | 120000 |
| Simenjuntak | Sakit Hati | Pasekap | 50000 | 50000 | 2500 | 40 | 200000 |
| Rosyanti | Sakit Tenggorokan | Sr. Bijl Kedondong | 50000 | 50000 | 1500 | 56 | 184000 |
| Suminah | Sakit Jantung | Sinse | 50000 | 50000 | 55000 | 13 | 815000 |
| Syamsudin | Sakit Kulit | Sinse | 50000 | 50000 | 55000 | 15 | 925000 |
| Bayhaqi | Sakit Flu | Bodrek | 50000 | 50000 | 3000 | 52 | 256000 |
| Rusmiati | Sakit Bersalin | Kuku Bina TL | 50000 | 50000 | 1500 | 100 | 250000 |
| Muhamin | Sakit Mata | Sinse | 50000 | 50000 | 55000 | 65 | 3675000 |
| Zulfikar | Sakit DED | Sinse | 50000 | 50000 | 55000 | 32 | 1860000 |

Gambar 5.51. Tampilan tabel Bayar01 hasil penggabungan dengan INNER JOIN

- Periksalah satu persatu tabel **Layan**, **Datang**, **Derita** dan **Bayar**. Pastikan nama-nama *field* telah sesuai dengan relasi/SQL yang diharapkan. Apabila belum sesuai, anda dapat menghapus tabel hasil relasi tersebut dengan perintah **DROP TABLE**.

Misalkan tabel **Bayar** tidak sesuai dan anda akan mengganti dengan tabel **Bayar** yang baru. Anda dapat menghapus tabel tersebut dengan perintah sebagai berikut:

DROP TABLE Bayar

Dan dilanjutkan dengan membuat relasi yang telah disesuaikan, dengan cara mengetikkan perintah SQL seperti sebelumnya.

Atau dapat pula menggabungkan antara **Drop Table** dengan perintah SQL yang lain. Untuk lebih jelasnya, dapat dilihat perintah dibawah ini :

DROP TABLE Bayar

```
SELECT d.NmPsn, e.NmSakit, f.NmObat, g.TrfDok,
a.Tarif, f.Harga, c.Jumlah, Total = (g.TrfDok+a.Tarif) +
(f.Harga*c.Jumlah)
```

INTO Bayar

```
FROM Servis a, Kunjung b, Resep c, Pasien d, Penyakit e,
Obat f, TarifDok g
```

```
WHERE a.KdSakit = b.KdSakit AND
```

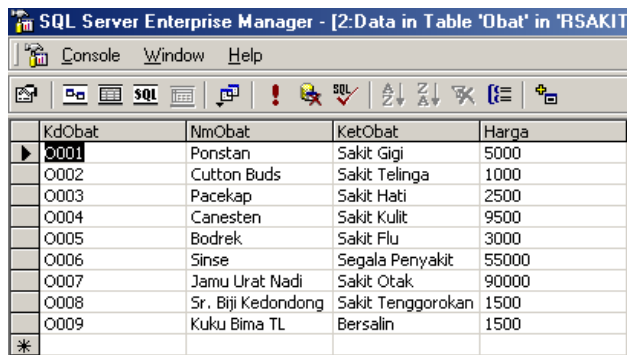
b.KdRsp = c.KdRsp AND
 b.KdPsn = d.KdPsn AND
 a.KdSakit = e.KdSakit AND
 c.KdObat = f.KdObat AND
 a.KdDok = g.KdDok

8. Untuk mengubah isi *field*, gunakan perintah **UPDATE**. Pada kasus diatas, yang akan dirubah adalah semua isi *field* **Harga** pada tabel **Obat** dengan kenaikan sebesar 10%, sehingga didapatkan rumus sebagai berikut (**Harga*1.1**). Adapun bentuk penulisannya adalah sebagai berikut :

```

USE Rsakit
GO
UPDATE Obat
SET Harga = Harga*1.1
  
```

Kemudian tekan tombol **F5** pada *keyboard* untuk menjalankan perintah tersebut. Perbedaan hasilnya dapat anda lihat pada gambar 5.52 dan 5.53 dibawah ini :



| KdObat | NmObat | KetObat | Harga |
|--------|--------------------|-------------------|-------|
| O001 | Ponstan | Sakit Gigi | 5000 |
| O002 | Cutton Buds | Sakit Telinga | 1000 |
| O003 | Pacekap | Sakit Hati | 2500 |
| O004 | Canesten | Sakit Kulit | 9500 |
| O005 | Bodrek | Sakit Flu | 3000 |
| O006 | Sinse | Segala Penyakit | 55000 |
| O007 | Jamu Urat Nadi | Sakit Otak | 90000 |
| O008 | Sr. Biji Kedondong | Sakit Tenggorokan | 1500 |
| O009 | Kuku Bima TL | Bersalin | 1500 |

Gambar 5.52. Tampilan tabel Obat sebelum perubahan

Semua harga telah berubah

| KdObat | NmObat | KetObat | Harga |
|--------|--------------------|-------------------|-------|
| O001 | Ponstan | Sakit Gigi | 5500 |
| O002 | Cutton Buds | Sakit Telinga | 1100 |
| O003 | Pacekap | Sakit Hati | 2750 |
| O004 | Canesten | Sakit Kulit | 10450 |
| O005 | Bodrek | Sakit Flu | 3300 |
| O006 | Sinse | Segala Penyakit | 60500 |
| O007 | Jamu Urat Nadi | Sakit Otak | 99000 |
| O008 | Sr. Biji Kedondong | Sakit Tenggorokan | 2200 |
| O009 | Kuku Bima TL | Bersalin | 1650 |

Gambar 5.53. Tampilan tabel Obat setelah perubahan

9. Untuk mengubah isi *field* dengan kondisi, gunakan perintah **UPDATE** dan **WHERE**. Pada kasus diatas, yang akan dirubah adalah semua isi *field* **TrfDok** pada tabel **Bayar** dengan potongan sebesar 25%, sehingga didapatkan rumus sebagai berikut [**TrfDok**-(**TrfDok***0.25)]. Adapun bentuk penulisannya adalah sebagai berikut :

```

USE Rsakit
GO
UPDATE Bayar
SET TrfDok = TrfDok - (TrfDok*0.25)
WHERE NmPsn = 'Bayhaqi'

```

Kemudian tekan tombol **F5** pada *keyboard* untuk menjalankan perintah tersebut. Perbedaan hasilnya dapat anda lihat pada gambar 5.54 dan 5.55 dibawah ini :

| NmPsn | NmSakit | NmObat | TrfDok | Tarif | Harga | Jumlah | Total |
|-------------|-------------------|--------------------|--------|-------|--------|--------|---------|
| Sunadi | Sakit Gigi | Ponstan | 50000 | 50000 | 5000 | 19 | 195000 |
| Ruhayah | Sakit Telinga | Cutton Buds | 50000 | 50000 | 1000 | 20 | 120000 |
| Simanjuntak | Sakit Hati | Pacekap | 50000 | 50000 | 2500 | 40 | 200000 |
| Rosyanti | Sakit Tenggorokan | Sr. Biji Kedondong | 50000 | 50000 | 1500 | 56 | 184000 |
| Suminah | Sakit Jantung | Sinse | 50000 | 50000 | 55000 | 13 | 815000 |
| Syamsudin | Sakit Kulit | Sinse | 50000 | 50000 | 925000 | 15 | 925000 |
| Bayhaqi | Sakit Flu | Bodrek | 50000 | 50000 | 3000 | 52 | 256000 |
| Rusniati | Sakit Bersalin | Kuku Bima TL | 50000 | 50000 | 1500 | 100 | 250000 |
| Muhamin | Sakit Mata | Sinse | 50000 | 50000 | 55000 | 65 | 3675000 |
| Zulfikar | Sakit DED | Sinse | 50000 | 50000 | 55000 | 32 | 1860000 |

Gambar 5.54. Tampilan tabel Bayar sebelum perubahan

SQL Server Enterprise Manager - [2>Data in Table 'Bayar' in 'RSAKIT' on 'AGUNG']

| NmPsn | NmSakit | NmObat | TrfDok | Tarif | Harga | Jumlah | Total |
|-------------|-------------------|--------------------|--------|-------|-------|--------|---------|
| Sumadi | Sakit Gigi | Ponstan | 50000 | 50000 | 5000 | 19 | 195000 |
| Ruhayah | Sakit Telinga | Cutlon Buds | 50000 | 50000 | 1000 | 20 | 120000 |
| Simanjuntak | Sakit Hati | Pacekap | 50000 | 50000 | 2500 | 40 | 200000 |
| Rosyanti | Sakit Tenggorokan | Sr. Biji Kedondong | 50000 | 50000 | 1500 | 56 | 184000 |
| Suminah | Sakit Jantung | Sinse | 50000 | 50000 | 55000 | 13 | 815000 |
| Syamsudin | Sakit Kulit | Sinse | 50000 | 50000 | 55000 | 15 | 925000 |
| Bayhaqi | Sakit Flu | Bodrek | 37500 | 50000 | 3000 | 52 | 256000 |
| Rusmiati | Sakit Bersalin | Kuku Bima TL | 50000 | 50000 | 1500 | 100 | 250000 |
| Muhamin | Sakit Mata | Sinse | 50000 | 50000 | 55000 | 65 | 3675000 |
| Zulfikar | Sakit DED | Sinse | 50000 | 50000 | 55000 | 32 | 1860000 |

TrfDok pada Bayhaqi telah berubah

Gambar 5.55. Tampilan tabel Bayar setelah perubahan

- Untuk mengubah judul *field* pada tabel Datang, anda dapat mengikuti penulisan sebagai berikut :

USE Rsakit

GO

SELECT "Nama Pasien" = NmPsn, "Nama Penyakit" = NmSakit,
"Nama Klinik" = NmKln

FROM Datang

Kemudian tekan tombol **F5** pada *keyboard* untuk menjalankan perintah tersebut. Hasilnya dapat anda lihat pada gambar 5.56 dibawah ini :

| <pre> USE Rsakit GO SELECT "Nama Pasien" = NmPsn, "Nama Penyakit" = NmSakit, "Nama Klinik" = NmKln FROM Datang </pre> | | | |
|---|-------------|-------------------|-----------------|
| | Nama Pasien | Nama Penyakit | Nama Klinik |
| 1 | Ruhayah | Sakit Telinga | Klinik THT |
| 2 | Syamsudin | Sakit Kulit | Klinik Kulit |
| 3 | Suminah | Sakit Jantung | Klinik Jantung |
| 4 | Sumadi | Sakit Gigi | Klinik Gigi |
| 5 | Rosyanti | Sakit Tenggorokan | Klinik THT |
| 6 | Bayhaqi | Sakit Flu | Klinik Umum |
| 7 | Rusmiati | Sakit Bersalin | Klinik Bersalin |
| 8 | Zulfikar | Sakit DBD | Klinik Umum |
| 9 | Muhaimin | Sakit Mata | Klinik Umum |
| 10 | Simanjuntak | Sakit Hati | Klinik THT |

Gambar 5.56. Tampilan dengan perubahan nama *field*

Buatlah tampilan untuk tabel yang lain dengan perubahan nama *field* sesuai dengan peruntukan.

BAB VI

NORMALISASI DAN DENORMALISASI

6.1. Normalisasi Data

Pembuatan perancangan basis data, dibutuhkan seorang analis sistem untuk menganalisa data yang ada dengan menggunakan model data. Saat model data telah efektif mengomunikasikan kebutuhan-kebutuhan basis data, mungkin model data tidak diperlukan untuk merepresentasikan desain basis data dengan baik. Mungkin saja model data berisi struktur karakteristik yang hanya menampilkan fleksibilitas dan perluasan atau membuat hal-hal yang tidak penting secara berlebihan. Oleh karena itu kita harus menyiapkan semua atribut model data untuk desain dan implementasi basis data.

Bagaimana membuat model data yang baik? Untuk membuat model data yang baik, harus mempunyai kriteria dibawah ini :

1. Model data yang baik adalah sederhana.

Peraturan secara umum, atribut-atribut data menggambarkan pemberian setiap entitas yang hanya dapat melukiskan entitas itu sendiri. Jadi setiap kita melihat atribut, kita akan langsung dapat membaca bahwa atribut tersebut termasuk dalam entitas tertentu. Sebagai contoh :

| <u>Entitas</u> | <u>Atribut</u> |
|----------------|--|
| Mahasiswa | : NIM (<i>primary key</i>) + Nm_Mhs + Alm_Mhs + Kd_Kul (<i>foreign key</i>) + Sks + Semester + Nilai |

2. Model data yang baik adalah tidak berlebihan terhadap hal-hal yang tidak perlu.

Pembuatan model data yang baik harus dapat menampilkan entitas-entitas yang hanya diperlukan saja. Oleh karena itu pembuatan atribut harus memperhatikan fungsi dari atribut tersebut, apakah kelak digunakan atau tidak. Bila atribut tersebut sangat jarang digunakan, sebaiknya buat satu

entitas tersendiri yang kelak akan menampung atribut tersebut. Ini dimaksudkan untuk efisiensi *record* dan kecepatan data dapat ditingkatkan.

3. Model data yang baik adalah fleksibel dan dapat beradaptasi untuk kebutuhan yang akan datang.

Pembuatan model data memang dilakukan untuk kebutuhan perusahaan atau organisasi pada masa kini, namun sebaiknya pembuatan basis data harus diperhatikan kemudahan dalam mengubah basis data untuk masa yang akan datang, termasuk juga untuk perubahan struktur dari basis data tersebut yang dapat ditambahkan atau dimodifikasi terhadap pengaruh yang kuat dari program yang berbeda. Sebaiknya dalam pembuatan basis data digunakan program DBMS yang sering digunakan, agar kelak dapat memudahkan dalam modifikasi basis data.

Adapun teknik yang digunakan untuk memperbaiki model data dalam persiapan untuk desain basis data disebut dengan analisis data. Analisis data adalah :

Proses persiapan model data untuk implementasi sederhana, tidak berlebih-lebihan, fleksibel dan adaptasi basis data (Whitten et. al., 2001:288).

Atau teknik secara spesifik disebut dengan normalisasi. Sedangkan yang dimaksud dengan normalisasi adalah :

Teknik analisis data untuk mengelola atribut data seperti pengelompokan (*group*) untuk tampilan yang tidak berlebih-lebihan, tetap atau tidak berubah-ubah, fleksibel dan dapat beradaptasi terhadap entitas-entitas yang ada.

Sebelum melangkah lebih jauh mengenai normalisasi, ada baiknya kita mengenal terlebih dahulu tipe data dan domain. Tipe data seperti yang telah dibahas sebelumnya, merupakan penentuan struktur setiap tabel. Pembuatan tipe data berimplikasi pada batas-batas nilai yang mungkin disimpan pada tiap-tiap atribut, contohnya *character*, *numeric*, *integer* dan seterusnya. Secara umum tipe data lebih berfokus pada kemampuan penyimpanan data terhadap atribut secara fisik, tanpa melihat kelayakan data tersebut bila dilihat dari kenyamanan pemakaiannya. Untuk lebih jelasnya dapat anda

Sedangkan yang dimaksud dengan domain lebih berfokus pada penekanan batas-batas nilai yang diperbolehkan bagi suatu atribut dilihat dari kenyataan yang ada.

The diagram illustrates the following entities and their attributes:

- MEMBER ORDER**: Primary Key: Member Number (PK1); Non-Key Attributes: Order Creation Date, Member Name, Member Address, Shipping Address Name, Shipping Street Address, Shipping City, Shipping State, Shipping Zip, Shipping Instructions, Order Sub Total, Order Sales Tax, Order Shipping Method, Order Shipping & Handling Cost, Order Status, Order Preparation Method, Member Number (FK), Member Number (FK), Member Number (FK), Ordered Product Number, Ordered Shipped, Ordered Product Description, Current Product Title, Current Unit Price, Extended Price.
- MEMBER**: Primary Key: Member Number (PK1); Non-Key Attributes: Member Name, Member Status, Member Street Address, Member Post Office Box, Member City, Member State, Member Zip Code, Member Cellphone Phone Number, Member Date of Last Order, Member Credit Card Type, Member Credit Card Number, Member Credit Card Expiration Date, Member Screen Balance Available, Audio Media Preference, Auto Media Preference, Auto Media Preference Date Entered, Genre Category Preference, Genre Media Preference, Number of Credits Earned, Video Category Preference, Audio Media Preference, Agreement Number (FK), Privacy Code, Email Address.
- TRANSACTION**: Primary Key: Transaction Number (PK1); Non-Key Attributes: Transaction Date, Transaction Type, Transaction Description, Transaction Amount, Member Number (FK), Order Number (FK).
- AGREEMENT**: Primary Key: Agreement Number (PK1); Non-Key Attributes: Agreement Start Date, Agreement Active Date, Fulfillment Period, Requirement of Credits.
- PRODUCT**: Primary Key: Product Number (PK1); Non-Key Attributes: Universal Product Code (Alternate Key), Quantity in Stock, Product Type, Suggested Retail Price, Default Unit Price, Current Special Unit Price, Current Month Unit Price, Current Year Unit Price, Total Units Under Sold.
- MERCHANDISE**: Primary Key: Product Number (PK1) (FK); Non-Key Attributes: Merchandise Name, Merchandise Description, Merchandise Type, Unit of Measure.
- TITLE**: Primary Key: Product Number (PK1) (FK); Non-Key Attributes: Title Cover, Catalog Description, Copyright Date, Entertainment Category, Credit Value.
- PROMOTION**: Primary Key: Promotion Number (PK1); Non-Key Attributes: Promotion Release Date, Promotion Status, Promotion Type, Product Number, Title of Work.
- AUDIO TITLE**: Primary Key: Product Number (PK1) (FK); Non-Key Attributes: Artist, Audio Sub Category, Number of Tracks in Package, Audio Media Code, Custom Arbitrary Code.
- VIDEO TITLE**: Primary Key: Product Number (PK1) (FK); Non-Key Attributes: Director, Video Category, Video Sub Category, Closed Captioned, Language, Running Time, Video Media Type, Video Encoding, Screen Aspect, MP Rating Code.
- GAME TITLE**: Primary Key: Product Number (PK1) (FK); Non-Key Attributes: Manufacturer, Game Category, Game Sub Category, Game Platform, Game Media Type, Number of Players, Parent Subject Code.
- TITLE PROMOTION**: Primary Key: Promotion Number (PK1) (FK); Foreign Key: Product Number (FK2) (FK).

Relationships and Cardinalities:

- Seller**: Between MEMBER ORDER and PRODUCT (1:M).
- Responded To**: Between MEMBER ORDER and MEMBER (1:M).
- Places**: Between MEMBER ORDER and MEMBER (1:M).
- Is Textured As**: Between PROMOTION and TITLE PROMOTION (1:M).
- Features**: Between PROMOTION and TITLE PROMOTION (1:M).
- Agreement**: Between TRANSACTION and AGREEMENT (1:M).
- Product**: Between PRODUCT and MERCHANDISE (1:M).
- Title**: Between PRODUCT and TITLE (1:M).
- Promotion**: Between PRODUCT and PROMOTION (1:M).
- Audio Title**: Between PRODUCT and AUDIO TITLE (1:M).
- Video Title**: Between PRODUCT and VIDEO TITLE (1:M).
- Game Title**: Between PRODUCT and GAME TITLE (1:M).

Gambar 6.1. Atribut model data secara keseluruhan

Pada suatu perancangan basis data yang sedang dilakukan, yang perlu kita lihat dan pertimbangkan domain dari setiap atribut. Penetapan tipe data bagi suatu atribut baru akan relevan dan penting untuk perhitungan pada saat implementasi basis data.

Adapun yang dimaksud dengan normalisasi di sini adalah suatu teknik dengan 3 tahap untuk menempatkan model data kedalam bentuk normal. Pembuatan normalisasi terlaksana jika terdapat ketergantungan relasi antara atribut satu entitas dengan atribut entitas yang lain, yang dikenal dengan fungsional tergantung penuh (*fully functional dependence/FFD*). Misalnya atribut Y dikatakan tergantung penuh secara fungsional pada atribut X, bila Y tergantung pada X namun tidak tergantung pada *subset* dari X.

Banyak pendekatan dalam penggunaan normalisasi. Pada pembahasan ini, pilihannya adalah menampilkan pendekatan yang tidak terlalu teorikal dan tidak matematis. Kita tinggalkan teori, relasional aljabar dan implikasi detail untuk bagian basis data dan semua yang mengarah ke buku teks. Kita akan menampilkan studi kasus untuk menampilkan selangkah demi selangkah pada normalisasi. Sebagai contoh, kita gunakan studi kasus pada gambar 6.1. pada gambar tersebut terlihat bentuk model data secara keseluruhan yang dibuat sebagai awal untuk pembuatan normalisasi. Sedangkan bentuk-bentuk normal tersebut menurut Jeffrey L. Whitten, Lonnie D. Bentley, Kevin C. Dittman adalah :

1. Bentuk normal tahap pertama (*1st Normal Form / 1NF*)

Bentuk normal tahap pertama, jika pada entitas tidak ada atribut yang dapat mempunyai lebih dari satu nilai pada gambaran tunggal sebuah entitas atau dapat pula dikatakan bila dan hanya bila semua domain hanya mengandung harga atomik. Tiap atribut pada entitas dapat mempunyai nilai ganda yang digambarkan secara aktual pada entitas yang terpisah, mungkin sebuah entitas dan relasinya.

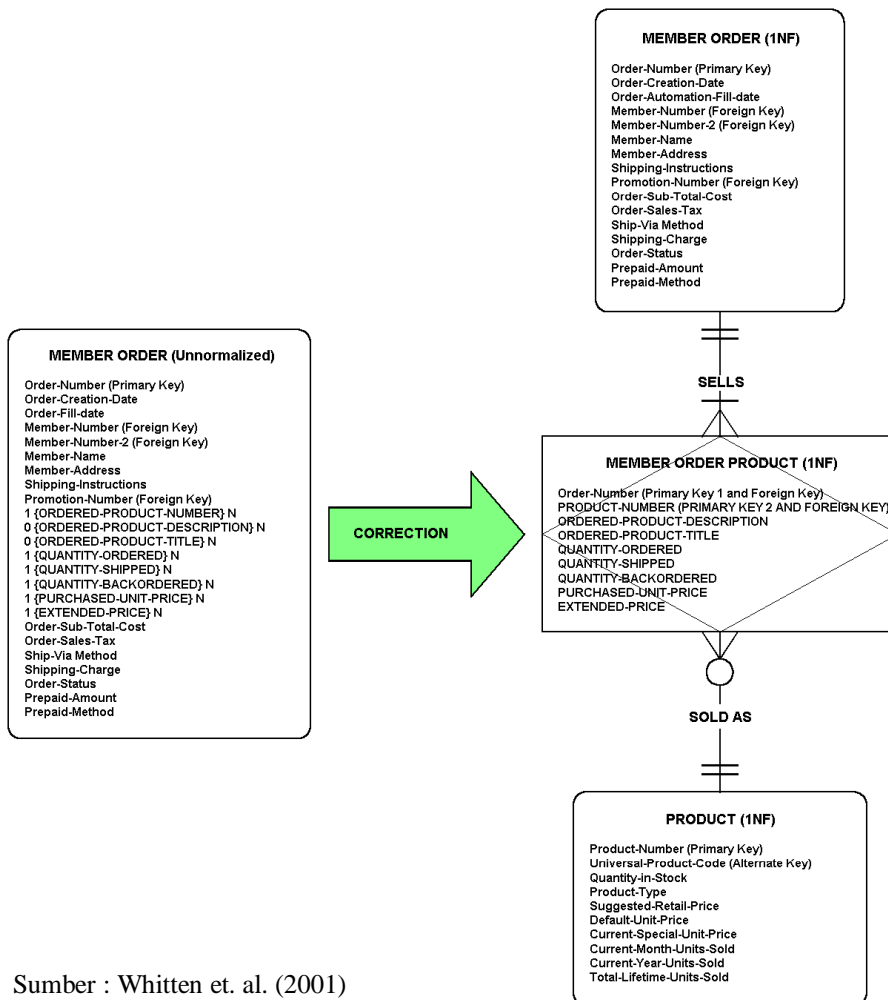
Tahap pertama pada analisis data adalah untuk menempatkan tiap entitas (gambar 6.1) kedalam 1NF. Kita temukan dua buah entitas sebagai awal model data yang belum terpengaruh dengan pengelompokan dan / atau turunan, yaitu MEMBER ORDER dan PROMOTION. Tiap atribut di sana

memuat pengulangan grup, sebuah grup atribut dapat mempunyai nilai ganda untuk sebuah pendekatan entitas. Atribut tersebut akan menyita banyak waktu. Sebagai contoh, atribut pada entitas MEMBER ORDER mungkin dipunyai oleh banyak PRODUCT, seperti ORDERED PRODUCT NUMBER, ORDERED PRODUCT DESCRIPTION, ORDERED PRODUCT TITLE, QUANTITY ORDERED, QUANTITY SHIPPED, QUANTITY BACKORDERED, PURCHASED UNIT PRICE dan EXTENDED PRICE. Atribut-atribut tersebut mungkin melakukan pengulangan untuk tiap hal dalam MEMBER ORDER.

Demikian pula, Sejak PROMOTION mungkin dibentuk lebih dari satu atribut PRODUCT TITLE, PRODUCT NUMBER dan TITLE OF WORK yang mengalami pengulangan. Bagaimana untuk merapikan entitas tersebut kedalam model?

Pada gambar 6.2 dan 6.3 terlihat bahwa entitas tersebut dibagi menjadi 2 (dua) entitas dan 1 (satu) relasi kedalam 1NF. Entitas yang asli adalah dilukiskan di pihak sebelah kiri dari gambar tersebut, dan entitas 1NF dilukiskan disebelah kanannya. Pada gambar tersebut terlihat bagaimana perubahan normalisasi model data dan penugasan atribut.

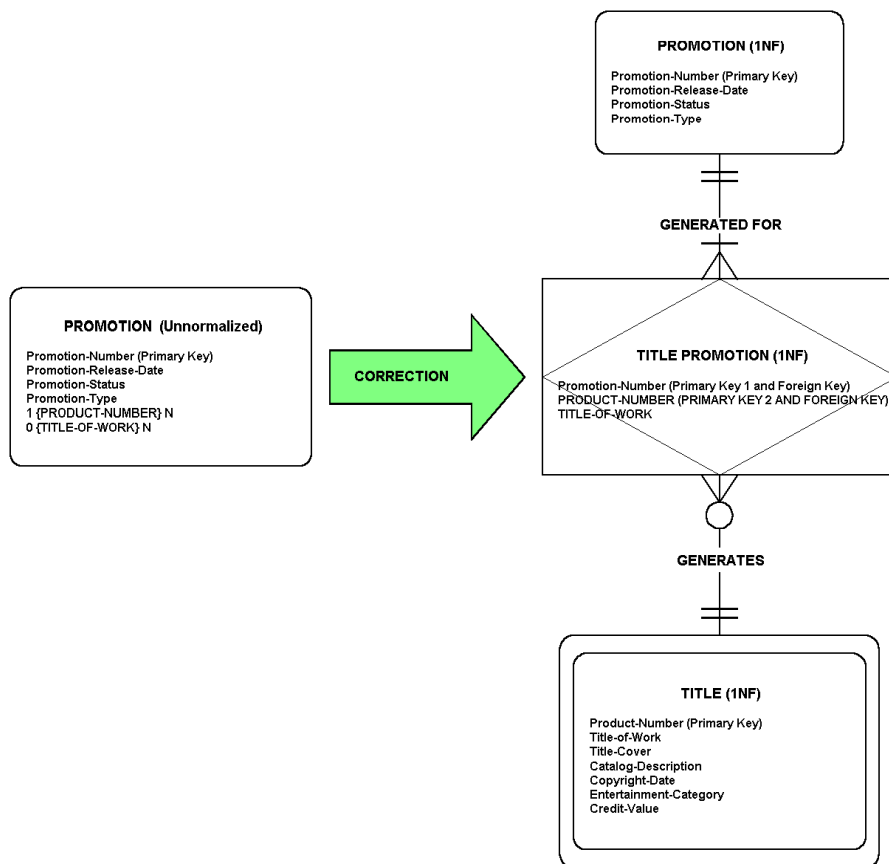
Pada gambar 6.2, pertama kali kita pindahkan atribut yang dapat mempunyai lebih dari satu nilai pada kejadian entitas MEMBER ORDER dalam 1NF. Kemudian kita pindahkan semua grup atribut ke entitas baru yang merupakan penghubung atau relasi untuk 2 entitas yang lain, yaitu MEMBER ORDERED PRODUCT. Tiap hal yang berada di atribut digambarkan kedalam satu PRODUCT pada MEMBER ORDER secara tunggal. Kemudian, jika PRODUCT mempunyai 5 (lima) pengiriman secara khusus, maka ada lima hal pada entitas MEMBER ORDERED PRODUCT. Tiap hal pada entitas hanya mempunyai 1 (satu) nilai untuk tiap atribut pada entitas baru yang berada pada bentuk normal pertama (1NF).



Sumber : Whitten et. al. (2001)

Gambar 6.2. Bentuk normal ke satu (1NF) Member Order

Bentuk normal pertama yang lain dapat dilihat pada gambar 6.3.



Sumber : Whitten et. al. (2001)

Gambar 6.3. Bentuk normal ke satu (1NF) Promotion

Jadi pada bentuk normal tahap pertama, yang harus dilakukan adalah jika sebuah tabel yang tidak memiliki lebih dari satu atribut yang bernilai banyak dengan domain nilai yang sama, tidak perlu membuat struktur tabel berubah tetapi cukup dengan menambahkan atau membuat tabel baru.

2. Bentuk normal tahap ke dua (2^{nd} Normal Form / 2NF)

Bentuk normal tahap kedua dapat terbentuk jika bentuk normal tingkat pertama sudah ada dan jika nilai semua yang bukan kunci primer (*primary key*) bergantung pada kunci primer, atau dapat pula dikatakan bila dan hanya bila ada dalam 1NF dan tiap atribut non kunci bergantung penuh pada

kunci primer. Tiap atribut bukan kunci yang bergantung hanya pada bagian kunci primer harus dipindahkan ke tiap entitas, dimana bagian kunci merupakan kunci penuh yang sebenarnya. Disini mungkin dibutuhkan pembuatan entitas baru dan relasi pada model.

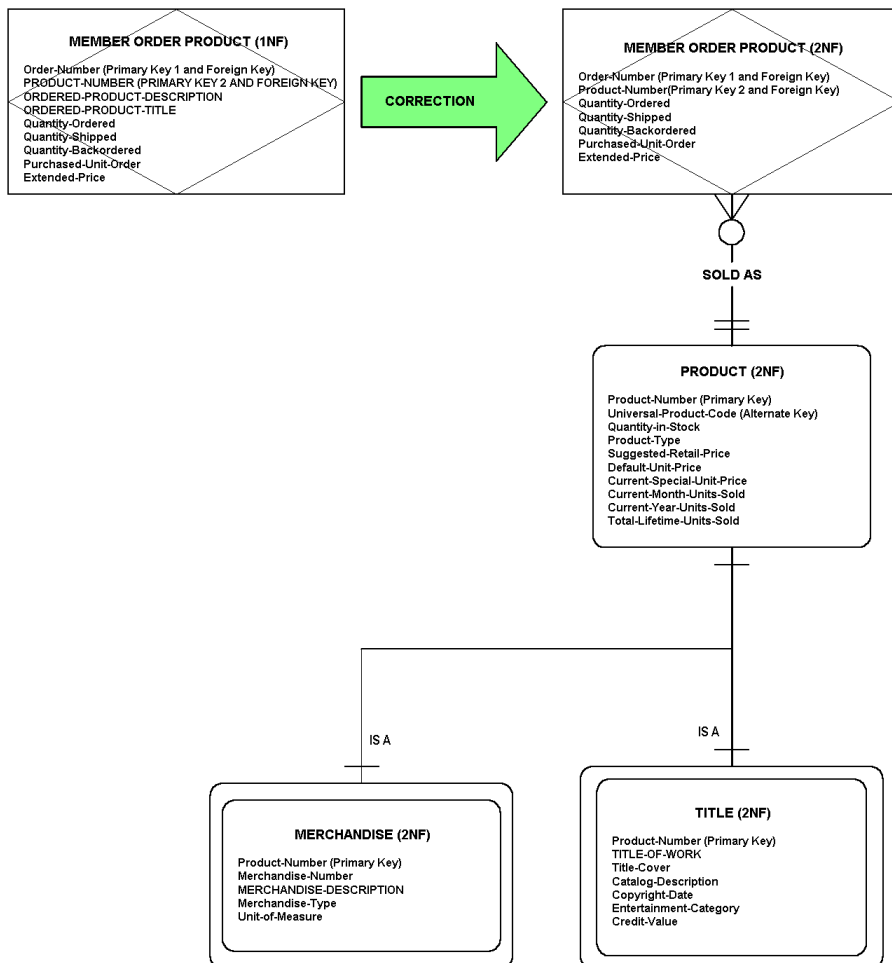
Bentuk normal tahap kedua (2NF) merupakan kelanjutan dari bentuk normal tahap pertama (1NF) pada analisis data. Perlu diingat kembali bahwa 2NF membutuhkan semua penempatan entitas yang tersedia pada 1NF. Perlu diingat juga bahwa 2NF dilihat pada siapa yang mempunyai nilai atribut dapat ditentukan hanya oleh bagian pada kunci primer dan tidak digunakan pada rangkaian semua kunci. Begitu pula dengan entitas hanya mempunyai satu atribut kunci primer yang tersedia dalam 2NF. Kita berikan perhatian pada PRODUCT (termasuk atributnya), MEMBER ORDER, MEMBER, PROMOTION, AGREEMENT, dan TRANSACTION. Jadi yang kita butuhkan hanya untuk memeriksa kebutuhan rangkaian kunci pada MEMBER ORDERED PRODUCT dan TITLE PROMOTION.

Pertama, mari kita cek entitas MEMBER ORDERED PRODUCT. Sebagian besar atribut-atribut akan bergantung pada kunci primer secara keseluruhan. Sebagai contoh, QUANTITY ORDERED dibuat tidak mempunyai arti kecuali anda menginginkan kedua ORDER NUMBER dan PRODUCT NUMBER. Berfikir tentang hal tersebut bahwa ORDER NUMBER adalah tidak cukup sejak mempunyai banyak jumlah produk pada pengiriman. Begitu pula dengan PRODUCT NUMBER tidak cukup sejak produk yang sama dapat terlihat pada pengiriman. Lebih dari itu, QUANTITY ORDERED membutuhkan kedua bagian pada kunci dan hal tersebut harus berdiri sendiri pada kunci secara penuh. Untuk hal yang sama dapat dikatakan sebagai QUANTITY SHIPPED, QUANTITY BACKORDERED, PURCHASE UNIT PRICE, dan EXTENDED PRICE.

Tetapi bagaimana dengan ORDERED PRODUCT DESCRIPTION dan ORDERED PRODUCT TITLE? Apakah kita

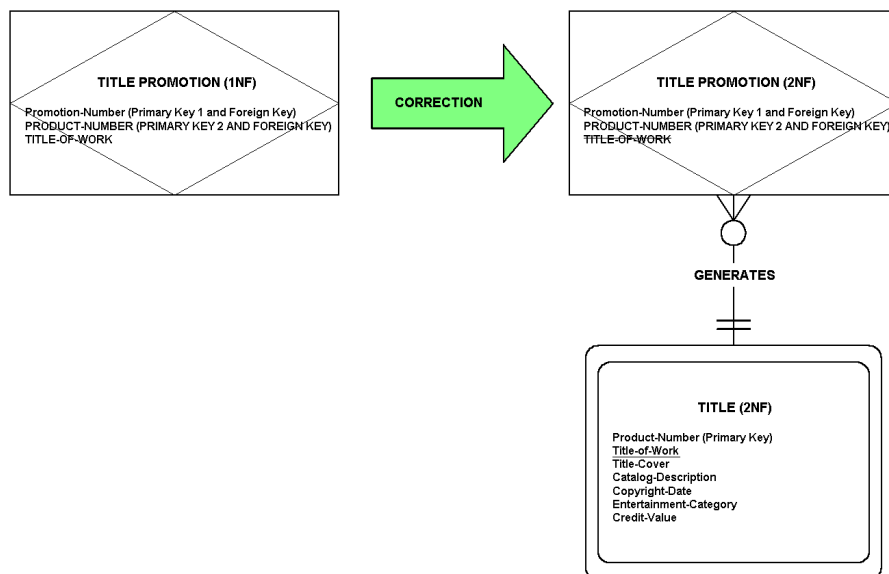
benar-benar membutuhkan ORDER NUMBER untuk menentukan nilai yang lain? Ternyata tidak, terlebih nilai pada atribut tersebut bergantung pada nilai PRODUCT NUMBER. Jadi semua atribut-atribut tersebut tidak bergantung pada kunci secara penuh. Kita tidak harus menemukan kembali sebagian dari penyimpangan bawahan / turunan yang harus dibetulkan. Bagaimana untuk membetulkan normalisasi yang salah?

Lihat gambar 6.3, untuk membetulkan masalah, kita pindahkan atribut yang bukan kunci (ORDERED PRODUCT DESCRIPTION dan ORDERED PRODUCT TITLE) ke sebuah entitas yang hanya PRODUCT NUMBER sebagai kuncinya. Jika dibutuhkan, kita akan membuat entitas baru, tetapi entitas PRODUCT akan digunakan dengan kunci yang telah ada atau tersedia. Namun kita harus hati-hati, karena PRODUCT adalah supertipe (*supertype*). Pada inspeksi subtype (*subtype*), kita temukan atribut-atribut di sana telah tersedia pada entitas MERCHANDISE dan TITLE, sekalipun semuanya satu arti. Demikian pula, kita tidak mengerjakan secara aktual untuk memindahkan atribut dari entitas MEMBER ORDERED PRODUCT. Kita hanya perlu menghapusnya sebagai entitas yang berlebihan.



Gambar 6.4. Bentuk normal ke ke dua (2NF) Member Order Product

Selanjutnya mari menguji entitas TITLE PROMOTION. Sedangkan rangkaian kunci diartikan sebagai kombinasi antara PROMOTION NUMBER dan PRODUCT NUMBER. Dan TITLE OF WORK dinyatakan berlebihan pada porsi PRODUCT NUMBER pada rangkaian kunci. Demikian pula TITLE OF WORK harus dipindahkan dari TITLE PROMOTION (lihat gambar 6.5). Untuk diketahui, pada TITLE OF WORK telah ada pada entitas TITLE, yang mana PRODUCT NUMBER digunakan sebagai kunci primer di sana.



Gambar 6.5. Bentuk normal ke ke dua (2NF) Title Promotion

3. Bentuk normal tahap ke tiga (3^{th} Normal Form / 3NF)

Bentuk normal tahap ketiga (3NF) terjadi jika 2NF telah tersedia dan jika nilai atribut yang bukan kunci primer tidak bergantung pada atribut yang bukan kunci primer yang lain, atau dapat pula dikatakan sebagai bila dan hanya bila ada dalam 2NF dan tiap atribut non kunci tergantung non transitif atas kunci primer. Bentuk normal tahap ke tiga (3NF) dikenal pula dengan sebutan *boyce-codd normal form* (BCNF).

3NF dalam model data mempunyai masalah, tidak memenuhi untuk relasi yang:

- 1) Mempunyai banyak kunci kandidat.
- 2) Kunci kandidat saling tumpang tindih (mempunyai paling tidak 1 (satu) atribut bersama).

Tiap atribut yang bukan kunci bergantung pada atribut bukan kunci yang lain harus dipindahkan atau dihapus. Sekali lagi, entitas baru dan relasinya mungkin dapat ditambahkan untuk model data.

Kita dapat menyederhanakan lebih lanjut entitas-entitas dengan ditematkannya kedalam 3NF. Semua kebutuhan

entitas dalam 2NF harus diidentifikasi sebelum memulai analisis 3NF. Analisis 3NF dapat dilihat pada 2 (dua) tipe masalah, yaitu pengambilan data dan ketergantungan-ketergantungan transitif. Dalam kasus-kasus sebelumnya, kesalahan secara fundamental adalah ketergantungan pada atribut yang tidak mempunyai kunci. Langkah awal pada tipe 3NF adalah sangat mudah, yaitu menguji atau memeriksa tiap entitas untuk pengambilan data.

Yang dimaksud dengan pengambilan data di sini adalah dimana salah satu nilai dapat dikalkulasi dari atribut-atribut lain atau pengiriman logika secara langsung dari nilai pada atribut yang lain.

Jika anda berpikir tentang pengiriman data, cerita pengiriman data membuat sedikit pengertian. Pertama, tentang pemborosan ruang penyimpanan disk. Kedua, kesulitan pengiriman basis data apakah akan mudah untuk diperbaharui (*update*). Mengapa demikian?, karena tiap anda mengubah dasar atribut-atribut, anda harus melakukan kembali penghitungan dan kemudian akan mengubah hasil pengiriman data.

Sebagai contoh, lihat entitas MEMBER ORDERED PRODUCT pada gambar 9.66. Pada atribut EXTENDED PRICE adalah hasil kalkulasi oleh perkalian antara QUANTITY ORDERED dan PURCHASED UNIT PRICE. Selanjutnya, EXTENDED PRICE (atribut bukan kunci) adalah tidak bergantung lebih banyak pada kunci primer sebagai ketergantungan pada atribut-atribut bukan kunci QUANTITY ORDERED dan PURCHASED UNIT PRICE. Setelah itu, kita koreksi entitas dengan menghapus EXTENDED PRICE.

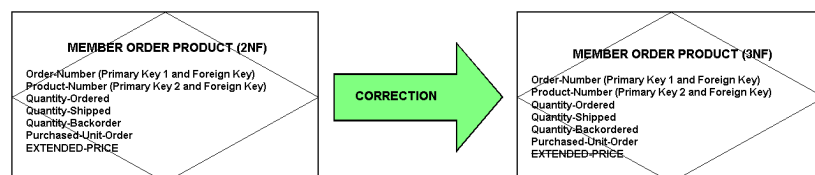
Pembuatan 3NF memang terlihat sangat mudah, namun tidak demikian. Di sana terdapat pertentangan bagaimana anda membuat peraturan untuk digunakan di akhir. Beberapa pakar basis data membantah bahwa peraturan harus diaplikasikan hanya dalam sebuah entitas. Jadi, para pakar basis data tidak akan menghapus pengiriman atribut jika

kebutuhan atribut untuk mengirim di sana diberikan untuk entitas-entitas yang berbeda.

Kita setuju bahwa dasar persetujuan pada argumen di sana dikirimkan atribut yang meliputi entitas-entitas ganda lebih berbahaya untuk permasalahan data yang tidak konsisten akibat pembaharuan atribut pada satu entitas dan dilupakan untuk pembaharuan berikutnya pada pengiriman atribut dalam entitas yang lain.

Tampilan lain dari 3NF adalah pemeriksaan analisis data untuk bergantung secara transitif. Ketergantungan transitif ada ketika atribut bukan kunci adalah tergantung pada atribut bukan kunci yang lain (oleh atribut asal). Indikasi kesalahan yang tidak ditemukan entitas biasanya adalah menghilangkan penyimpanan dalam masalah entitas. Seperti kondisi misalnya, jika tidak dibetulkan, dapat menyebabkan masalah fleksibilitas dan adaptasi jika akhirnya kebutuhan baru dibutuhkan semua untuk melaksanakan entitas biasanya sebagai tabel basis data yang dilakukan secara terpisah.

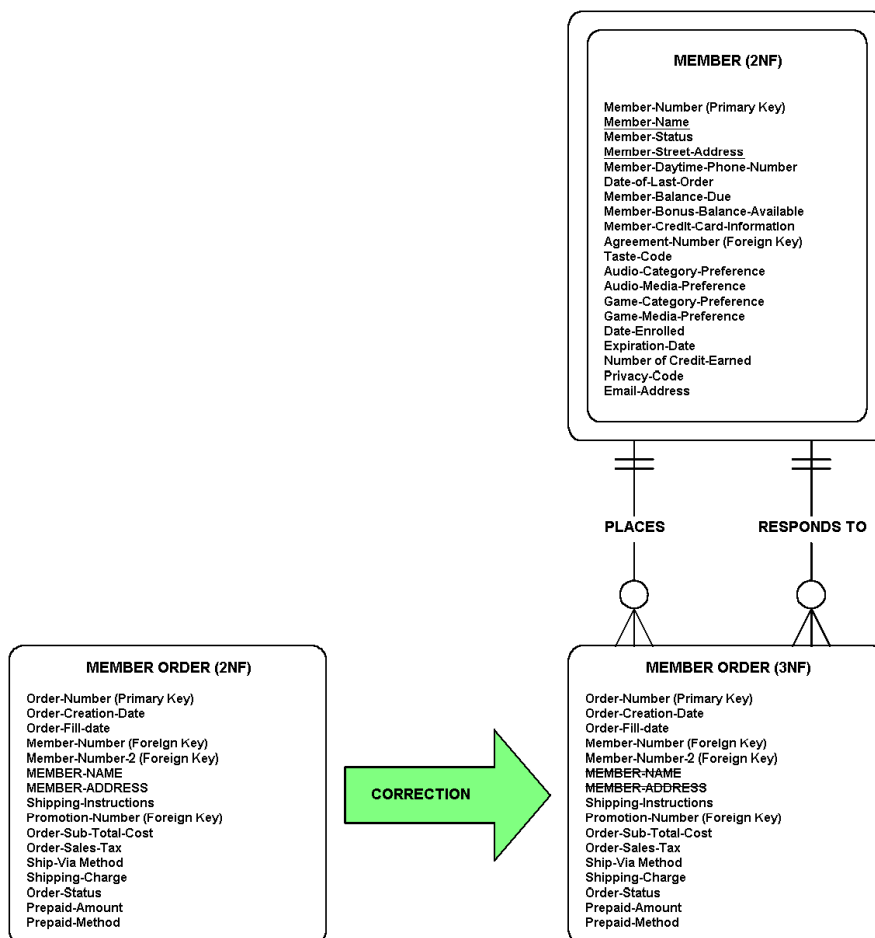
Analisis transitif adalah hanya kinerja pada entitas-entitas yang tidak membutuhkan kunci kandidat (*candidate key*). Contoh dalam buku ini, termasuk dalam PRODUCT, MEMBER ORDER, PROMOTION, AGREEMEENT, MEMBER dan TRANSACTION. Untuk entitas PRODUCT, semua atribut bukan kunci bergantung pada kunci primer. Selain itu, PRODUCT telah ada dalam 3NF. Analisis yang mirip pada PROMOTION, AGREEMENT dan TRANSACTION dinyatakan semua termasuk dalam 3NF.



Sumber : Whitten et. al. (2001)

Gambar 6.6. Bentuk normal ke ketiga (3NF) Member Order Product

Lihat entitas MEMBER ORDER pada gambar 6.6. Dalam fakta-fakta, di uji atribut MEMBER NAME dan MEMBER ADDRESS. Atribut-atribut tersebut bergantung pada kunci primer. Sedangkan kunci primer ORDER NUMBER bukan jalan untuk menentukan nilai MEMBER NAME dan MEMBER ADDRESS. Adapun nilai dari MEMBER NAME dan MEMBER ADDRESS bergantung pada nilai-nilai lain yang bukan kunci primer dalam entitas MEMBER NUMBER.



Gambar 6.7. Bentuk normal ke ke tiga (3NF) Member Order

Bagaimana untuk membetulkan masalah tersebut? MEMBER NAME dan MEMBER ADDRESS dibutuhkan untuk dipindahkan dari entitas MEMBER ORDER pada entitas yang mempunyai kunci primer dan ternyata hanya MEMBER ORDER. Jika dibutuhkan, kita dapat membuat entitas baru, tetapi dalam kasus ini kita telah mempunyai entitas MEMBER dengan kebutuhan kunci primernya. Seperti yang diharapkan, kita tidak membutuhkan pemindahan yang sebenarnya pada perpindahan atribut sebenarnya karena mereka sudah tersedia untuk entitas MEMBER. Yang kita lakukan adalah bagaimanapun juga kita harus memperhatikan MEMBER ADDRESS sebagai persamaan untuk MEMBER STREET ADDRESS. Kita harus memilih orang-orang untuk disimpan pada terminal berikutnya dalam MEMBER.

4. Bentuk normal tahap ke empat (4^{th} Normal Form / 4NF) dan bentuk normal tahap ke lima (5^{th} Normal Form / 5NF)

Penerapan normalisasi sampai dengan bentuk normal tahap ke tiga (3NF) sesungguhnya sudah memadai untuk membentuk tabel yang berkualitas baik. Namun, dari sejumlah literatur dapat dijumpai adanya pembahasan tentang bentuk normal tahap ke empat (4NF) dan bentuk normal tahap ke lima (5NF).

Pembuatan bentuk 4NF berkaitan dengan ketergantungan banyak nilai (*multivalued dependency*), yaitu suatu tabel yang merupakan pengembangan ketergantungan. Atribut tersebut untuk diisi dengan lebih dari satu, namun mempunyai nilai yang sama. Untuk lebih jelasnya dapat dilihat pada gambar 6.8.

MEMBER ORDER

Atribut bernilai tunggal

| ORDER NUMBER | ORDER CREATION DATE | ORDER FILL DATE | MEMBER NUMBER | SHIPPING ADDRESS | |
|--------------|---------------------|--|---------------|------------------|-------|
| 2004-O-001 | 12-01-2004 | 30-01-2004 25-02-2004 30-03-2004 | M-001-0379 | Jl. Angkasa 2/30 | |
| | | | | | |
| | | | | | |

Atribut bernilai banyak

Gambar 6.8. Atribut bernilai tunggal dan bernilai banyak

form, yaitu mendefinisikan cara yang serupa dengan 3NF dan 4NF, kecuali fungsinya untuk mengikutsertakan ketergantungan-ketergantungan yang digunakan.

Pembahasan kedua bentuk normal ke empat (4NF) dan ke lima (5NF) ini cukup kompleks, sedangkan manfaatnya sendiri tidak begitu besar. Oleh karena itu pembahasan detailnya tidak disertakan dalam buku ini.

Yang terpenting bahwa kesuksesan bentuk normal akan berpengaruh dalam pembuatan model data yang sederhana, yaitu kurangnya *redundancy* data dan data lebih fleksibel. Bagaimanapun juga, analisis sistem (termasuk pakar basis data) sangat jarang memberikan model data melebihi bentuk normal ke tiga (3NF). Konsekuensinya, kita akan melanjutkan meninggalkan diskusi normalisasi menuju ke buku teks basis data.

Yang pertama dalam pembuatan model data dengan sedikit waktu adalah proses akan muncul lambat dan membosankan. Bagaimanapun juga, dengan waktu dan latihan, model data akan dibuat dengan cepat dan rutin. Banyak pengalaman pemodelan secara signifikan akan mengurangi waktu pembuatan model dan usaha selama pengerjaan normalisasi. Hal ini mungkin dapat membantu untuk selalu mengingat dan mengikuti perubahan (mungkin ada yang tidak diketahui), untuk memperbaiki pembuatan bentuk pertama, kedua dan ketiga. Yang perlu diingat dalam pembuatan normalisasi adalah bahwa tiap entitas akan dikatakan

bentuk ketiga (3NF), jika setiap atribut bukan kunci primer adalah bergantung pada kunci primer dan / atau keseluruhan kunci primer.

Untuk memudahkan anda dalam pembuatan normalisasi, anda dapat melihat gambar 6.9 sebagai langkah-langkah pembuatan normalisasi. Namun yang terpenting dalam normalisasi adalah bagaimana data yang dibuat tidak rangkap dalam sebuah tabel data.

1NF

| A | B |
|----|----|
| a1 | b1 |
| a2 | b2 |

2NF

| C | G | H |
|----|----|----|
| c1 | g1 | h1 |
| c2 | g2 | h2 |

3NF

| A | I |
|----|----|
| a1 | i1 |
| a2 | i2 |

4NF

| A | C | G |
|----|----|----|
| a1 | c1 | g1 |
| a2 | c2 | g2 |

5NF

| A | B | C | G | H | I |
|----|----|----|----|----|----|
| a1 | b1 | c1 | g1 | h1 | i1 |
| a2 | b2 | c2 | g2 | h2 | i2 |

Gambar 6.9. Langkah normalisasi dari 1NF sampai 5NF

Banyak alat bantu CASE untuk mendukung konsep normalisasi yang digunakan. Semua alat bantu tersebut akan membaca mode data dan melakukan percobaan untuk mengisolasi kemungkinan kesalahan normalisasi. Dari semua alat bantu tersebut, biasanya hanya dapat membuat bentuk normal tingkat pertama (1NF). Semuanya di lihat untuk relasi banyak ke banyak dan memutuskan hubungannya kedalam asosiasi entitas-entitas. Atau semuanya melihat pada gambaran atribut-atribut khusus sebagai nilai ganda untuk kejadian entitas tunggal.

Hal tersebut diatas terlampau sulit untuk alat bantu CASE pada identifikasi bentuk normal tahap kedua (2NF) dan ketiga (3NF). Adapun alat bantu CASE akan dibutuhkan untuk penalaran pada sebagian pengakuan dan transisi ketergantungan. Dalam kenyataan,

tiap ketergantungan hanya dapat ditemukan pada kurang dari kebiasaan atau pekerjaan rutin oleh analis sistem atau pakar basis data.

Ketika sebuah model logika data adalah digunakan secara efektif untuk menggambarkan data apakah disimpan untuk sistem baru, alat bantu tidak dapat melakukan komunikasi terhadap kebutuhan pada dasar lokasi operasional perusahaan. Kita butuh untuk identifikasi apakah data dan akses benar dibutuhkan untuk lokasi. Kita mungkin akan bertanya:

- Yang mana *subset* entitas-entitas dan atribut dibutuhkan untuk performa kerja pada tiap lokasi?
- Apakah tingkatan pada akses adalah kebutuhan?
- Dapatkah kejadian *create* pada entitas?
- Dapatkah kejadian *read* pada entitas?
- Dapatkah kejadian *delete* pada entitas?
- Dapatkah kejadian *update* yang ada pada entitas?

Sistem analis dapat menemukan semua dengan cara definisi kebutuhan logika dalam laporan dengan menggunakan *data-to-location-CRUD matrix* (gambar 6.10).

Data to Location CRUD Matrix

| Entity - Atribut | Location | Customer | Kansas City | Marketing | Advertising | Warehouse | Sales | A/R | Boston | Sales | Warehouse | San Francisco | Sales | San Diego | Warehouse |
|------------------------------|----------|----------|-------------|-----------|-------------|-----------|-------|-----|--------|-------|-----------|---------------|-------|-----------|-----------|
| Customer | INDV | | | | | ALL | ALL | | | SS | SS | | SS | | SS |
| . Customer Number | R | | | | | R | CRUD | R | | CRUD | R | | CRUD | | R |
| . Customer Name | RU | | | | | R | CRUD | R | | CRUD | R | | CRUD | | R |
| . Customer Address | RU | | | | | R | CRUD | R | | CRUD | R | | CRUD | | R |
| . Customer Credit Rating | X | | | | | R | RU | | | R | | | R | | |
| . Customer Balance Due | R | | | | | R | RU | | | R | | | R | | |
| Order | INDV | | ALL | | | SS | ALL | | | SS | SS | | SS | | SS |
| . Order Number | SRD | R | CRUD | R | CRUD | R | CRUD | R | | CRUD | R | | CRUD | | R |
| . Order Date | SRD | R | CRUD | R | CRUD | R | CRUD | R | | CRUD | R | | CRUD | | R |
| . Order amount | SRD | R | CRUD | | | R | CRUD | R | | CRUD | R | | CRUD | | R |
| Ordered Product | INDV | | ALL | | | SS | ALL | | | SS | SS | | SS | | SS |
| . Quantity Ordered | SUD | R | CRUD | R | CRUD | R | CRUD | R | | CRUD | | | CRUD | | |
| . Ordered Item Unit Price | SUD | R | CRUD | | | R | CRUD | R | | CRUD | | | CRUD | | |
| Product | ALL | | ALL | ALL | ALL | ALL | | | | ALL | ALL | | ALL | | ALL |
| . Product Number | R | CRUD | R | R | R | | | | | R | R | | R | | R |
| . Product Name | R | CRUD | R | R | R | | | | | R | R | | R | | R |
| . Product Description | R | | CRUD | RU | R | R | | | | R | R | | R | | R |
| . Product Unit of Measure | R | CRUD | R | R | R | R | | | | R | R | | R | | R |
| . Product Current Unit Price | R | | CRUD | R | | R | | | | R | R | | R | | R |
| . Product Quantity on Hand | X | | | | | RU | R | | | R | RU | | R | | RU |

INDV = Individual ALL = All SS = Subset X = No Access
 S= Submit C = Create R = Read U = Update D = Delete

Sumber : Whitten et. al. (2001)

Gambar 6.10. Data-to-location-CRUD matrix

Adapun yang dimaksud dengan *data-to-location matrix* adalah : tabel yang mana barisnya mengidentifikasikan entitas (mungkin

atribut), kolom mengidentifikasi lokasi dan sel (*cell*) (perpotongan baris dan kolom) dokumen tingkatan akses dimana C = *create*, R = *read*, U = *update* dan D = *delete* atau tidak aktif. (Whitten, 2001 : 197)

6.2. Denormalisasi Data

Seperti yang telah dibahas sebelumnya, normalisasi merupakan sebuah upaya untuk mendapatkan basis data dengan struktur yang baik, terutama sekali untuk efisiensi ruang penyimpanan (*storage*). Tetapi penggunaan normalisasi yang terlalu ketat dapat pula berakibat pada penurunan performa basis data. Pada dasarnya, normalisasi digunakan sebagai petunjuk yang berperan pada saat kita melakukan perancangan basis data. Normalisasi dapat saja kita langgar atau abaikan, dengan pertimbangan-pertimbangan yang lain. Sedangkan sistem manajemen basis data (DBMS) digunakan untuk mengimplementasikan basis data secara fisik, dan tidak digunakan sebagai pembatas para pengguna untuk selalu memenuhi aturan normalisasi basis data. Atau dengan kata lain, normalisasi digunakan untuk standar dalam perancangan basis data dan bukan keharusan dalam perancangan basis data.

Pelanggaran-pelanggaran normalisasi disebut dengan denormalisasi. Mengapa digunakan normalisasi?, karena dalam perancangan basis data harus memperhatikan tingkat performa yang baik atau terbaik. Apabila tingkat performa dapat ditingkatkan dengan denormalisasi, maka penggunaan normalisasi dapat ditiadakan atau diabaikan.

Pada relasi basis data dapat terjadi hubungan yang berlebihan dan semuanya tidak dapat dihilangkan karena ketergantungan antara tabel yang satu dengan tabel yang lain, terutama untuk atribut-atribut yang digunakan sebagai kunci primer. Akan tetapi relasi yang berlebihan dapat diminimalkan, karena dapat berdampak pada gangguan integritas dari basis data, khusus adanya perubahan data yang terjadi pada tabel yang lain yang tidak dibahas sebelumnya. Untuk mengatasinya, seorang analis sistem sudah terbiasa dengan penggunaan normalisasi dengan meniadakan atribut yang berlebihan

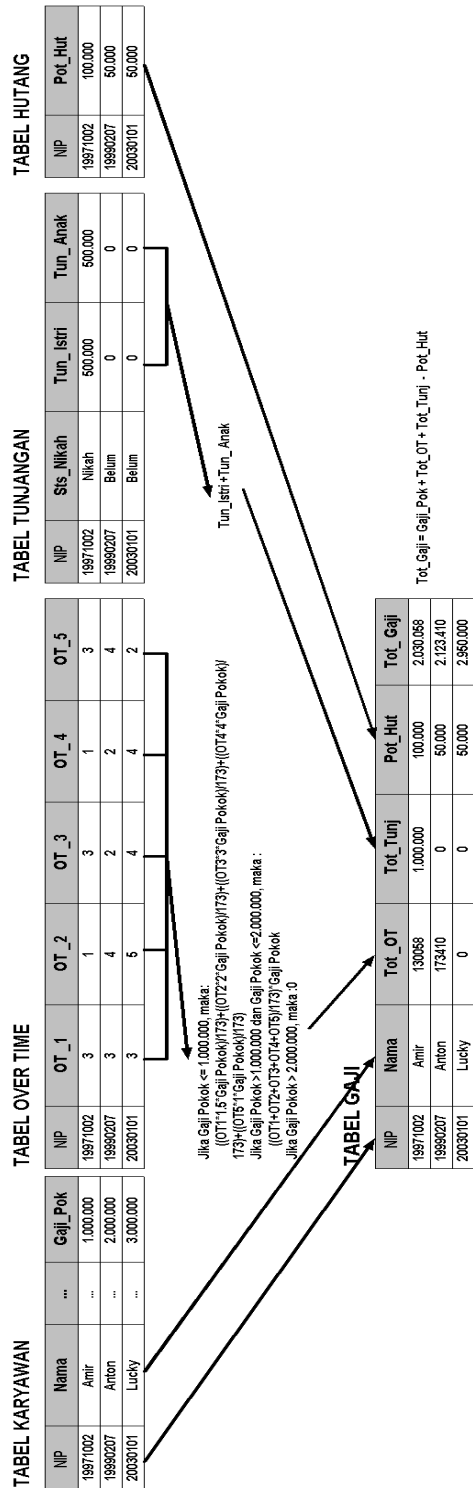
dan tidak perlu. Pembuatan normalisasi biasanya akan terjadi pada atribut-atribut yang digunakan sebagai kunci. Sedangkan pembuatan relasi yang berlebihan akan berakibat pada :

- a. Ruang penyimpanan yang lebih besar dari semestinya.
- b. Dapat melemahkan integrasi basis data.
- c. Waktu yang digunakan untuk mengelola dan memelihara integritas basis data dapat bertambah pada saat proses perubahan (*update*) data.

Adapun bentuk denormalisasi yang dapat terjadi antara lain :

- a. Terdapatnya atribut turunan

Pembuatan atribut dalam basis data seperti yang dibahas sebelumnya, sebenarnya dapat pula dibuat atribut yang merupakan hasil dari kalkulasi dari atribut yang lain. Atau atribut tersebut merupakan atribut berlebihan (*redundant*). Atribut semacam dapat pula digunakan untuk menghindari proses kalkulasi yang berulang dan memakan waktu. Namun hal ini harus menjadi perhatian analis sistem, terutama pakar basis data apakah proses dapat mengakibatkan dari normalisasi atau proses lebih baik dengan menggunakan denormalisasi.



Gambar 6.11 Denormalisasi dengan atribut turunan

Proses denormalisasi dengan turunan dapat dilihat pada gambar 6.11 diatas. Pada gambar tersebut jika digunakan bentuk normalisasi, akan membutuhkan waktu yang lama, karena akan memakan waktu untuk normalisasi dan perhitungan matematis, yaitu perhitungan total *over time* dan total tunjangan.

Dikarenakan banyak tabel yang dilibatkan dan adanya perhitungan matematis, maka proses pengolahan data yang harus menampilkan total gaji tersebut akan memakan waktu cukup lama. Bagaimana jika jumlah karyawannya banyak? Maka sudah dapat dipastikan, waktu yang dibutuhkan akan lama setiap akan menampilkan TOT_GAJI. Pada gambar 9.71, TOT_GAJI hanya akan ditampilkan pada *report* saja.

Untuk mengatasinya, bentuk denormalisasi akan lebih cocok untuk kasus diatas dengan penambahan *field* TOT_OT dan TOT_TUNJ pada tabel GAJI. Dengan keberadaan 2 (dua) *field* tersebut, maka informasi TOT_GAJI yang diambil pada semua karyawan dapat diperoleh dengan mudah dan cepat diakses langsung hanya dari tabel KARYAWAN.

Namun cara diatas mempunyai konsekuensi yang mungkin memberatkan, karena diperlukan tambahan ruang penyimpanan dan proses manipulasi dapat lebih lambat. Untuk mengatasinya, diperlukan perangkat keras yang lebih tinggi atau cepat tingkatannya dan ruang penyimpanan yang lebih besar.

b. Atribut yang digunakan berlebihan

Atribut yang berlebihan dapat melanggar bentuk normal tingkat pertama (1NF), karena tidak mempunyai domain yang unik (tidak atomik). Akan tetapi penggunaan atribut semacam ini pada beberapa kasus atau keadaan akan dapat mengefisienkan pemakaian ruang penyimpanan untuk tabel basis data.

Jenis-jenis atribut yang berlebihan antara lain: (Fatansyah et. al., 2005:137)

- Atribut terkodekan (*encoded attribute*)

Yaitu suatu atribut yang memiliki kode tambahan yang menunjukkan beberapa kondisi lainnya.

Contoh :

Pada gambar 6.11, pada tabel HUTANG dapat ditambahkan atribut baru berupa kode hutang. Pada kode hutang tersebut dapat ditambahkan kode yang menunjukkan batasan peminjaman dan pengembalian sesuai dengan tingkatan manajemen.

- Penggabungan atribut (*concatenated attribute*)

Yaitu atribut dalam domain komposit.

Contoh :

Atribut NIP pada tabel gambar 9.71 terdiri dari angka tahun 4 digit, bulan 2 digit dan nomor urut 2 digit. Dengan demikian atribut tersebut bukan atribut atomik, karena masih dibagi lagi. Misalnya 19971002 = tahun 1997, bulan 10 dan tanggal 02. Adapun contoh yang lebih kompleks lagi misalnya NIM (nomor induk mahasiswa), 9222500523 terdiri dari tahun = 92, jurusan MI = 22, jenjang S1 = 5 dan nomor urut = 00523, dan masih banyak lagi contoh yang lain.

- Atribut tumpang tindih (*overlapping attribute*)

Yaitu atribut dengan nilai yang tidak eksklusif.

Contoh :

Pada gambar 6.11, tabel tunjangan dapat ditambahkan kode tunjangan. Dengan kode N=nikah, S=belum nikah (*single*) dan NS=duda atau janda. Kode dengan NS ini disebut dengan *overlapping*, karena nilainya mencakup N dan S sekaligus (tidak eksklusif), yang menunjukkan bahwa adanya perhitungan tunjangan (tunjangan anak), namun ia tidak mempunyai pasangan.

- Atribut bermakna ganda (*alternate attribute*)

Yaitu atribut yang memiliki arti yang berbeda tergantung sub entitasnya (kelompok entitasnya).

Contoh :

Pada tabel gambar 6.11, gaji pokok mempunyai status ganda, yaitu bila karyawan yang bekerja merupakan pegawai tetap, maka diisi dengan jumlah gaji yang sesuai dengan perjanjian saat masuk. Namun jika karyawan tersebut merupakan pegawai kontrak atau honorer akan diisi dengan gaji perjam. Jika tidak masuk 1 hari, maka karyawan tetap tidak dipotong, namun untuk karyawan kontrak akan dipotong dengan jumlah jam tidak masuk.

Karena sifat-sifat tersebut diatas, atribut-atribut semacam ini jelas melanggar aturan normalisasi. Untuk menormalisasikannya, diperlukan langkah dekomposisi (diuraikan) atribut menjadi beberapa atribut atomik. Namun hal ini akan dirasakan aneh bila dilakukan dekomposisi. Misalnya NIP dipisah menjadi 2 (dua) atribut *thn_masuk* dan *no_urut*. Padahal kita jarang membutuhkan tahun masuknya atau nomor urutnya saja, kita membutuhkan NIP secara utuh (gabungan antara tahun dan nomor urut).

c. Adanya tabel ringkasan (*summary*)

Tabel-tabel yang dikelola dalam basis data merupakan tabel detail. Oleh karena itu laporan yang bentuknya ringkasan akan membutuhkan gabungan dari beberapa tabel atau tabel *query* sekaligus. Dikarenakan semakin besar *volume* dan semakin banyaknya tabel-tabel yang terlibat, maka semakin lama waktu yang dibutuhkan untuk mendapatkan sebuah laporan. Tentu saja hal ini menjadi tidak praktis. Untuk solusinya kita simpan data hasil kalkulasi gabungan kedalam sebuah tabel khusus. Lihat gambar 9.71 pada tabel GAJI yang merupakan gabungan dari 4 buah tabel, yaitu tabel KARYAWAN, OVERTIME, TUNJANGAN dan HUTANG.

Karena pengolahan data tersebut untuk menghasilkan laporan ringkasan, maka disebut pula dengan tabel ringkasan. Sudah barang tentu tabel ini memiliki atribut yang berlebihan dan melanggar aturan normalisasi (khususnya tingkat ke 5 / 5NF).

SOAL

PT. ABC

- PT. ABC merupakan sebuah perusahaan yang sangat berkembang. Untuk meningkatkan kinerja perusahaannya, PT. ABC akan membuat sistem informasi penggajian karyawan agar karyawan dapat menerima gaji tepat pada waktunya.
- Dalam pemberian gaji karyawan, PT. ABC mengharuskan semua karyawannya menjadi nasabah Bank Ma'un. Karena nantinya semua karyawan PT. ABC akan menerima gaji melalui transfer Bank Ma'un. Selain dari pada itu PT. ABC merupakan perusahaan yang taat pajak dengan selalu menyetorkan pajak penghasilan (PPh 21) setiap bulannya kepada pemerintah, serta PT. ABC selalu menyetorkan sebagian penghasilan karyawan ke JAMSOSTEK sebagai simpanan untuk pensiun kelak.
- Sistem penggajian PT. ABC nantinya akan menghasilkan laporan periodik kebagian pembukuan / Accounting, Manajer Sumber Daya Manusia dan Pimpinan atas (Top Management). Adapun sistem penggajian PT. ABC ini merupakan suatu sistem yang berada di dalam Sistem Informasi Sumber Daya Manusia (SI-SDM) / Human Resources Information System (HRIS).
- Buatlah ER-D, Database, dan tabel-tabel dengan bahasa Query komersial.

DAFTAR PUSTAKA

- Fathansyah (2002), **Buku Teks Ilmu Komputer : Basis Data**, Bandung, Informatika.
- FitzGerald, Jerry, FitzGerald, Andra F., Stalling, Jr., Warren D. (1981), **Fundamentals of Systems Analysis**, edisi kedua, New York, John Wiley & Sons.
- Jogiyanto. HM, **Analisis & Disain Sistem Informasi**, edisi pertama, Yogyakarta, Andi Offset, 1995.
- Korth, Henry F., Silberschatz, Abraham (1991), **Database System Concepts**, edisi kedua, United States of America, McGraw Hill.
- Martina, Inge (2003), **36 Jam belajar Microsoft SQL Server 2000**, edisi pertama, Jakarta, Elexmedia Komputindo.
- Santoso, Harip (2003), **Pemrograman Client-Server Menggunakan SQL Server 2000 dan Visual Basic 6.0**, edisi pertama, Jakarta, Elexmedia Komputindo.
- Ullman, J. D. (1989), **Principles of Database and Knowledge Base System**, volume 2, Rockville, MD, Computer Science Press..
- Whitten, Jeffrey L., Bentley, Lonnie D., Dittman, Kevin C. (2001), **Systems Analysis And Design Methods**, edisi kelima, New York, McGraw- Hill Irwin.
- Yourdon, E. (1989), **Modern Structured Analysis**, New Jersey, Prentice Hall.

TENTANG PENULIS



Adyanata Lubis lahir di Pasir Pengaraian pada tanggal 24 Desember 1979. Pendidikan SD sampai SLTA (SMK) ditempuh di kota Pasir Pengaraian sampai tahun 1998. Pada tahun 2002 melanjutkan studi di Program Studi Agribisnis Politeknik Pasir Pengaraian yang diselesaikan pada tahun awal 2005. Setelah lulus program diploma melanjutkan studi pada program Studi Teknik Informatika pada Sekolah Tinggi Manajemen Ilmu Komputer Riau (STMIK-AMIK Riau) selesai pada tahun 2008, pada tahun bersamaan diterima sebagai staf pengajar di Politeknik Pasir Pengaraian yang kemudian berubah bentuk menjadi Universitas

Pasir Pengaraian, pada tahun 2012 melanjutkan studi pada Magister Ilmu Komputer Universitas Putra Indonesia YPTK Padang, pada tahun 2013 berhasil menyelesaikan S2 dengan gelar M.Kom. Pada Fakultas Ilmu Komputer mengasuh beberapa mata kuliah antara lain Aplikasi Komputer (SPSS), Dasar Rekayasa Desain I dan II, Basis Data, Kecerdasan Buatan, Pengantar Teknologi Informasi.

Penelitian yang pernah/sedang dilakukan :

1. Pendataan Potensi Desa wilayah Kecamatan Rambah dan Tambusai Utara Pada Tahun 2005. (Bappeda Kabupaten Rokan Hulu)
2. Pembuatan Film Animasi 3 Dimensi Perkuliahan Mahasiswa STMIK-AMIK Riau (2008)
3. Pemetaan Mina Agro Kabupaten Rokan Hulu. Tahun 2013 (Bappeda Kabupaten Rokan Hulu)
4. Evaluasi Tingkat Penerimaan Sistem Informasi Layanan Pengadaan Secara Elektronik Menggunakan Metode Technology Acceptance Model (TAM) Oleh Pengusaha (2013).

5. Pengaruh Media Pembelajaran Menggunakan Macromedia Flash Terhadap Prestasi Belajar Siswa SMKN 4 Rambah Kabupaten Rokan Hulu (2015)
6. Perancangan Sistem Informasi Usaha Ekonomi Kelurahan Simpan Pinjam (UEK-SP) Mekar Sari Pada Lembaga Pemberdayaan Kelurahan Rejosari Pekanbaru Berbasis Web (2015)
7. Penerapan Cloud Computing pada Infrastruktur as Service Penyimpanan data Akademik Universitas pasir pengaraian (2016)

Buku

1. Dasar Rekayasa dan Desain
2. Buku Ajar Basis Data 1