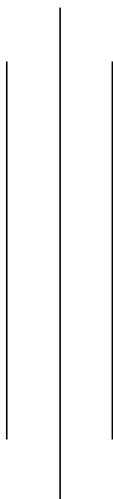


# **MODUL MATA KULIAH**

## **BASIS DATA**



**Disusun Oleh:**

Fauzi Adi Rafrastara, M.CS

Slamet Sudaryanto N., ST, M.Kom

Danang Wahyu Utomo, M.Kom

Dr. Aripin, M.Kom

**FAKULTAS ILMU KOMPUTER**  
**UNIVERSITAS DIAN NUSWANTORO**

## **DAFTAR ISI**

<b>BAB 1 PENGANTAR BASIS DATA .....</b>	<b>5</b>
SEJARAH KEMUNCULAN BASIS DATA .....	5
DEFINISI BASIS DATA .....	10
TUJUAN BASIS DATA.....	12
BASIS DATA VS SPREADSHEET .....	13
PENERAPAN BASIS DATA .....	13
HIRARKI DATA.....	14
<b>BAB 2 LINGKUNGAN DAN SISTEM BASIS DATA .....</b>	<b>17</b>
SISTEM BASIS DATA .....	17
KOMPONEN SISTEM BASIS DATA .....	18
SISTEM MANAJEMEN BASIS DATA.....	21
MENGENAL ARSITEKTUR DBMS.....	22
<b>BAB 3 MODEL DATA RELATIONAL .....</b>	<b>37</b>
DESAIN BASIS DATA .....	38
<i>Desain Konseptual (Conceptual Design)</i> .....	39
<i>Desain Logis (Logical Design)</i> .....	40
<i>Desain fisik (Physical Design)</i> .....	41
PEMODELAN DATA .....	42
MODEL E-R.....	44
MODEL RELASIONAL.....	45
<b>BAB 4 DIAGRAM E-R .....</b>	<b>47</b>
KELEBIHAN DAN KEKURANGAN ERD .....	47
SIMBOLISASI PADA ERD.....	48
ENTITAS .....	50
HIMPUNAN ENTITAS .....	50
ATRIBUT .....	51
RELASI DAN KARDINALITAS .....	55
<i>Batasan Kardinalitas (Cardinality Constraints) ....Error! Bookmark not defined.</i>	
HIMPUNAN RELASI .....	61
RELATION KEYS .....	62
<i>Jenis-Jenis Key di DBMS .....</i>	63
ENHANCED ENTITY-RELATIONSHIP MODEL .....	64

Spesialisasi (↓) .....	65
Generalisasi (↑) .....	68
Agregasi .....	71
PARTICIPATION CONSTRAINT .....	72
<b>BAB 5 TRANSFORMASI MODEL DATA KE BASIS DATA FISIK.....</b>	<b>74</b>
SUB ENTITAS.....	74
RELASI TUNGGAL .....	75
RELASI MULTI ENTITAS .....	ERROR! BOOKMARK NOT DEFINED.
RELASI GANDA .....	75
RELATIONAL INTEGRITY RULES.....	ERROR! BOOKMARK NOT DEFINED.
<b>BAB 6 IMPLEMENTASI BAHASA PADA RELATIONAL DATABASE.....</b>	<b>ERROR! BOOKMARK NOT DEFINED.</b>
BAHASA QUERY NORMAL PROCEDURAL .....	ERROR! BOOKMARK NOT DEFINED.
BAHASA QUERY NORMAL NON-PROCEDURAL..	ERROR! BOOKMARK NOT DEFINED.
BAHASA QUERY KOMERSIAL .....	ERROR! BOOKMARK NOT DEFINED.
<b>BAB 7 DESAIN BASIS DATA .....</b>	<b>88</b>
PROSES PERANCANGAN BASIS DATA .....	127
PENGEMBANGAN SISTEM .....	ERROR! BOOKMARK NOT DEFINED.
CONTOH APLIKASI ER .....	ERROR! BOOKMARK NOT DEFINED.
BAHASA QUERY KOMERSIAL .....	ERROR! BOOKMARK NOT DEFINED.
<b>BAB 8 NORMALISASI DATA DAN KETERGANTUNGAN FUNGSIONAL.....</b>	<b>141</b>
PENGERTIAN DAN TUJUAN NORMALISASI.....	141
PENTINGNYA NORMALISASI .....	142
JENIS-JENIS NORMALISASI .....	143
TAHAPAN NORMALISASI.....	144
<i>Tabel Universal</i> .....	144
INF – Bentuk Normal Pertama .....	145
2NF – Bentuk Normal Kedua.....	148
3NF – Bentuk Normal Ketiga.....	149
BCNF – Bentuk Normal Boyce-Codd .....	149
CLOSURE DAN KETERGANTUNGAN FUNGSIONAL .....	ERROR! BOOKMARK NOT DEFINED.
ANOMALI DAN DEPENDENSI .....	ERROR! BOOKMARK NOT DEFINED.
DIAGRAM DEPENDENSI FUNGSIONAL.....	ERROR! BOOKMARK NOT DEFINED.
DEKOMPOSISI TAK HILANG .....	ERROR! BOOKMARK NOT DEFINED.
CONTOH KASUS BENTUK NORMAL DAN TIDAK NORMAL.	ERROR! BOOKMARK NOT DEFINED.

---

<b>BAB 9 PENGENALAN SQL .....</b>	<b>150</b>
PENGANTAR SQL .....	150
PERINTAH SQL DASAR .....	151
<i>SQL: Data Definition</i> .....	151
<b>BAB 10 PEMROSESAN QUERY .....</b>	<b>158</b>
MANIPULASI DATA.....	158
<code>INSERT</code> .....	160
<code>SELECT</code> .....	161
<code>UPDATE</code> .....	172
<code>DELETE</code> .....	173
<code>ROLLBACK</code> .....	<i>Error! Bookmark not defined.</i>
<code>COMMIT</code> .....	<i>Error! Bookmark not defined.</i>
FUNGSI AGREGASI .....	173
NILAI NULL .....	175
<b>BAB 11 BEKERJA DENGAN SQL .....</b>	<b>176</b>
CREATE.....	<i>ERROR! BOOKMARK NOT DEFINED.</i>
DROP.....	<i>ERROR! BOOKMARK NOT DEFINED.</i>
ALTER .....	<i>ERROR! BOOKMARK NOT DEFINED.</i>
INSERT .....	<i>ERROR! BOOKMARK NOT DEFINED.</i>
UPDATE .....	<i>ERROR! BOOKMARK NOT DEFINED.</i>
DELETE .....	<i>ERROR! BOOKMARK NOT DEFINED.</i>
<b>BAB 12 RDBMS.....</b>	<b>ERROR! BOOKMARK NOT DEFINED.</b>
APLIKASI SQL DAN PL/SQL .....	<b>ERROR! BOOKMARK NOT DEFINED.</b>
<b>BAB 13 MANAJEMEN BASIS DATA: PROTEKSI DATA.....</b>	<b>ERROR! BOOKMARK NOT DEFINED.</b>
PEMULIHAN .....	<i>ERROR! BOOKMARK NOT DEFINED.</i>
PENGAMANAN .....	<i>ERROR! BOOKMARK NOT DEFINED.</i>
INTEGRITAS .....	<i>ERROR! BOOKMARK NOT DEFINED.</i>
KONKURENSI.....	<i>ERROR! BOOKMARK NOT DEFINED.</i>
<b>BAB 14 APLIKASI BASIS DATA.....</b>	<b>ERROR! BOOKMARK NOT DEFINED.</b>
DBMS .....	<i>ERROR! BOOKMARK NOT DEFINED.</i>
ARSITEKTUR SISTEM .....	<i>ERROR! BOOKMARK NOT DEFINED.</i>
PEMILIHAN DEVELOPMENT TOOLS .....	<i>ERROR! BOOKMARK NOT DEFINED.</i>
<b>REFERENSI.....</b>	<b>182</b>

# Bab 1

## PENGANTAR BASIS DATA

---

### Bab ini membahas:

- Sejarah Kemunculan Basis Data
  - Definisi Basis Data
  - Tujuan Basis Data
  - Basis Data vs Spreasheet
  - Penerapan Sistem Basis Data
  - Hirarki Data
- 

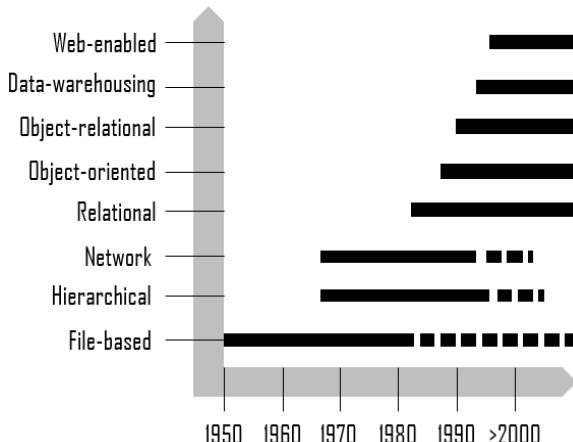
Basis data (*Database*) telah menjadi bagian tak terpisahkan pada kehidupan modern kita sekarang ini. Hampir setiap hari kita berada pada situasi yang membutuhkan interaksi dengan database, seperti: pemesanan online tiket kereta api hingga bus, pemesanan tiket bioskop, penggunaan mesin ATM, transaksi di kasir minimarket maupun supermarket, dll.

### Sejarah Kemunculan Basis Data

Sejak awal mula komputer dikembangkan, kegiatan menyimpan dan memanipulasi data merupakan salah satu fokus utama yang harus dipecahkan oleh para ilmuwan dan *engineer*. Aplikasi komputer, pada awalnya dibuat untuk menyelesaikan tugas-tugas administrasi, seperti perhitungan gaji karyawan, penjadwalan kerja pada industri manufaktur, dan lain sebagainya. Sesuai dengan permintaan dan kebutuhan para user, aplikasi-aplikasi tersebut harus mampu mengakses data yang (kala itu) tersimpan pada suatu file di komputer, mengubah data-data yang tersimpan menjadi suatu informasi, hingga menghasilkan beragam

laporan yang berguna untuk organisasi. Model yang demikian ini, kelak dikenal dengan istilah *File-Based System* atau *File-Based Method*.

Setelah sekian dekade berlalu, teknologi komputer mengalami perkembangan yang cukup signifikan, diantaranya yaitu dalam hal *data processing* dan *information management*. Adanya evolusi pada kedua hal tersebut, turut membawa perubahan pula pada metode penyimpanan data, hingga pada akhirnya dikembangkanlah suatu sistem database yang canggih dan modern. *File-based method* yang dibuat pada tahun 1950-an dirasa tidak mampu memenuhi kebutuhan, maka dibangunlah sistem database yang terintegrasi hingga seperti yang ada sekarang ini. Dengan adanya perkembangan ini, *file-based system* bukan berarti sudah tidak digunakan lagi, pada area-area tertentu ia masih tetap eksis. Evolusi teknologi sistem basis data, dapat dilihat pada gambar *Gambar 1*.



**Gambar 1: Evolusi teknologi sistem basis data**

Pada awal dekade 60an, presiden Amerika Serikat saat itu, John. F. Kennedy, menginisiasi sebuah projek besar yang diberi nama “*Apollo Moon Landing*”, dengan sebuah target yaitu mendaratkan manusia di bulan selambat-lambatnya pada akhir dekade 1960an. Projek tersebut pada prosesnya tentu akan menghasilkan data dengan volume yang super besar, dan pada saat itu belum ada sistem atau teknologi yang bisa menanganinya. Teknologi tercanggih saat itu, yakni *file-based system*,

tidaklah mampu menangani data masive tersebut. Pada masa-masa ini lah *database system* pertama kali diperkenalkan, untuk mengatasi kebutuhan tersebut.

*North American Aviation (NAA)* yang sekarang dikenal dengan nama *Rockwell International* merupakan kontraktor utama dari projek *Apollo Moon Landing*. Perusahaan ini berhasil membuat software yang dikenal dengan nama *Generalized Update Access Method (GAUM)* yang berhasil mengatasi tantangan terkait dengan data bervolume besar. Software GAUM ini dikembangkan dengan menggunakan suatu konsep bahwa komponen-komponen kecil merupakan bagian dari komponen-komponen yang lebih besar. Komponen-komponen yang lebih besar tersebut, merupakan bagian dari komponen-komponen yang lebih besar lagi. Begitu seterusnya, hingga produk akhirnya berhasil dirangkai dari susunan komponen-komponen di bawahnya. Struktur *up-down tree* tersebut kemudian diberi nama *Hierarchical Structure*. Sejak saat itu, database mengalami evolusi dan terus berkembang pada dekade-dekade berikutnya.

Pada pertengahan era 1960an, *Data-Base Management System (DBMS)* sebenarnya sudah didesain oleh Charles Bachman yang merupakan seorang engineer di General Electric, Amerika Serikat, dan karyanya tersebut dikenal dengan nama *Integrated Data Store (IDS)*. IDS merupakan awal mula dari satu model yang tidak kalah terkenal pada saat itu, yaitu *Network Model*. Berbeda dari model yang dikembangkan sebelumnya, *Network Model* telah distandarisasi oleh *Conference of Data System Language (CODASYL)*. CODASYL ini terdiri dari perwakilan pemerintah Amerika Serikat serta kalangan dari dunia bisnis dan perdagangan. Paska standarisasi ini, Network Model menjadi sangat berpengaruh pada perkembangan sistem basis data di era 60an. Pada saat itu, CODASYL membentuk satuan tugas yang bernama *List Processing Task Force (LPTF)* pada tahun 1965, dan kemudian berganti nama menjadi *Data Base Task Group (DBTG)*, yang bertanggung jawab dalam menentukan spesifikasi standar yang memungkinkan pembuatan dan manipulasi data. DBTG juga berperan dalam pembuatan standarisasi bahasa pemrograman COBOL yang kemudian menjadi tren pada saat itu. Atas karyanya dalam bidang basis data ini, pada tahun 1973,

Bachman dianugerahi sebuah penghargaan (layaknya *Nobel Prize*) dari *Association of Computing Machinery (ACM) Turing Award*. Penghargaan ini sekaligus menjadikannya sebagai orang pertama yang menerima penghargaan dari organisasi tersebut, yang kini sangat populer di kalangan para *Computer Scientist*.

Di saat yang lain, IBM bergabung dengan NAA dalam mengembangkan GAUM hingga kemudian dikenal dengan nama Information Management System (IMS), dan dirilis pada tahun 1968. Mengingat pada era tersebut *serial storage device*, seperti *magnetic tape*, merupakan media penyimpanan yang paling diminati oleh pasar, IBM membatasi IMS untuk berjalan di magnetic tape saja. Beberapa tahun berikutnya, IBM mulai mengembangkan IMS sehingga bisa digunakan oleh media penyimpanan jenis lain. IMS merajai pasar hierarchical DBMS pada saat itu dan terinstal pada banyak komputer mainframe. IMS inilah yang kemudian menjadi dasar dari framework alternatif untuk membuat representasi data, yaitu *Hierarchical Data Model*.

*Hierarchical model* menggambarkan data dengan model pohon (tree) dimana akarnya di atas, dan daun-daunnya ada di bagian bawah. Sedangkan pada *network model*, keterkaitan data digambarkan dengan menggunakan konsep graf (tanpa sirkuit) sehingga menjadikan proses pembacaan *real-world data* menjadi lebih mudah. CODASYL bersama dengan *hierarchical structure* merupakan generasi pertama dari DBMS. Dan sejak tahun 1970an, model hierarki dan jaringan (*network*) ini dikembangkan secara besar-besaran untuk mengatasi data yang semakin kompleks dan tidak mampu diatasi dengan menggunakan metode pemrosesan file yang konvensional. Meskipun demikian, hingga saat ini masih ada beberapa organisasi yang tetap setia menggunakan dua DBMS tersebut, dan sistem tersebut kini dikenal dengan istilah *Legacy Systems*.

Pada tahun 1970an, Edgar Codd, seorang peneliti di IBM (San Jose Research Laboratory), menulis sebuah *paper* dimana ia mengusulkan sebuah framework baru untuk representasi data (*data representation framework*), yang dinamakan dengan *Relational Data Model*, sekaligus memperkenalkan cara non-prosedural dalam meng-query data di dalam model relasional tersebut. Model ini kemudian disebut sebagai DBMS

generasi ke-2, dan memperoleh pengakuan sekaligus digunakan secara luas di dunia pada tahun 1980an. Pada masa-masa berikutnya, dikembangkanlah beberapa model DBMS yang terinspirasi dari model relasional ini.

Pada model relasional, semua data direpresentasikan dalam bentuk tabel. Di sini, *Structure Query Language (SQL)* mulai diperkenalkan dan digunakan untuk *data retrieval*. SQL dikenal sebagai bahasa query yang sederhana dan dikategorikan sebagai bahasa generasi ke-4. Model relasional ini mampu mengatasi beberapa kekurangan yang ada pada model-model yang dikembangkan sebelumnya, seperti dalam hal kesederhanaan, salah satunya membuat orang-orang awam (non-programmer) bisa memahaminya dengan mudah termasuk untuk urusan akses ke data. SQL juga memungkinkan untuk menyembunyikan detail implementasi, sehingga lebih mudah diterapkan oleh para programmer. Atas kontribusinya ini, Codd dianugerahi penghargaan ACM's Turing Award pada tahun 1981.

Pada awalnya, model relasional tidak begitu menarik minat user mengingat masih sebatas konsep yang belum dapat secara penuh diimplementasikan. Secara performa, model relasional belum mampu mengimbangi 2 model yang telah lebih dulu hadir, yaitu model network dan hierarki. Kondisi ini terus berlangsung hingga pada tahun 1980an IBM menginisiasi suatu projek yang bernama *System R*. Projek dari IBM ini berfokus pada bagaimana membuat *Relational Database System* yang efisien. Dari projek ini, maka lahirlah *Structured Query Language / Database System (SQL/DS)*, yang merupakan produk database relasional yang berfungsi secara penuh. Sejak saat itu, model relasional menjadi semakin dikenal dan digunakan secara luas, hingga akhirnya berhasil menggeser popularitas dari database berbasis *network* dan hierarki. Saat ini, ada beberapa relasional DBMS yang dapat digunakan di aplikasi komersial, baik di komputer mainframe maupun PC, seperti Ingress (dari Computer Associates), Informix (dari Informix Software Inc.), ORACLE, IBM DB2, Digital Equipment Corporation's Relational Database (DEC Rdb), Access dan FoxPro (dari Microsoft), Paradox (dari Corel Corporation), InterBase dan BDE (dari Borland), serta R-Base (dari R-Base Technologies). Di akhir 1980an (atau awal 1990an), SQL

distanدارisasi sekaligus diadopsi oleh American National Standards Institute (ANSI) dan International Standard Organizations (ISO).

Pada tahun 1990an, sebuah era baru muncul dalam dunia komputer. Saat itu, komputasi *client/server*, *data warehouse*, *world wide web* (*www*) mulai dikenal oleh publik. Dengan adanya pembaharuan-pembaharuan dalam dunia komputer ini, pengembangan di lingkup sistem basis data serta data multimedia (seperti grafik, suara, gambar, dan video) menjadi suatu keharusan. Di masa ini, Object-Oriented Data-Base (OODBMS) dan Objected-Relational Data-Bases (ORDBMS) mulai diperkenalkan untuk mengatasi data-data yang semakin kompleks. Object Oriented Databases selanjutnya dinobatkan sebagai database generasi ke-3.

Kemunculan paket *Enterprise Resource Planning (ERP)* dan *Management Resource Planning (MRP)* menambah fitur penting di layer *application-oriented* pada DBMS. Paket-paket tersebut merupakan hasil identifikasi pada fungsi atau tugas umum yang ada di berbagai organisasi, seperti manajemen inventori, analisa finansial, perencanaan SDM, perencanaan produksi, manajemen order dan lain sebagainya. Hal-hal tersebut merupakan sesuatu yang lumrah yang dimiliki oleh organisasi atau institusi pada umumnya, dan ERP juga MRP menyediakan fitur-fitur yang bisa digunakan untuk menyelesaikan tugas-tugas umum tersebut.

DBMS terus mengalami perkembangan dan memiliki peran semakin penting, mengingat semakin banyaknya data yang di-*online*-kan sehingga lebih mudah diakses melalui jaringan komputer (termasuk internet). Sekarang, bidang basis data merupakan salah satu bidang yang paling menarik dan menjanjikan berkat adanya konsep-konsep baru seperti *multimedia database*, *interactive video*, *digital library*, *data mining*, *big data*, dan lain sebagainya.

## Definisi Basis Data

Secara etimologi, Basis Data terdiri dari 2 suku kata, yaitu Basis dan Data. Menurut KBBI, Basis diartikan sebagai pangkalan. Dalam definisi

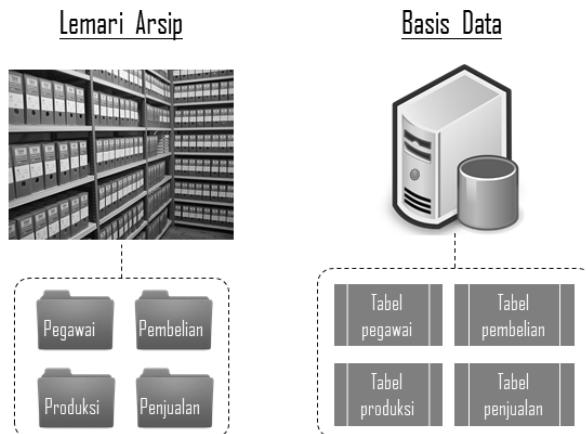
lain, basis juga dapat diartikan sebagai markas, gudang, juga tempat bersarang/berkumpul.

Sedangkan data, memiliki arti yaitu informasi dalam bentuk yang dapat diproses oleh komputer, seperti representasi digital dari teks, angka, gambar grafis, atau suara (KBBI). Data juga dikenal sebagai representasi dari fakta dunia nyata yang mewakili suatu objek seperti manusia (pegawai, siswa, pembeli, pelanggan), barang, hewan, peristiwa, konsep, keadaan dan sebagainya.

Dengan demikian, basis data dapat didefinisikan sebagai himpunan kelompok data (arsip) yang saling berhubungan dan diorganisasi sedemikian rupa agar kelak dapat dimanfaatkan kembali dengan cepat dan mudah. Definisi lain terkait basis data adalah kumpulan informasi yang terorganisir sehingga dapat dengan mudah diakses, di-manage, dan di-update.

Pada dasarnya, basis data memiliki prinsip kerja yang sama seperti lemari arsip, yaitu pengaturan dan penyimpanan data/arsip. Tujuan dasar dari keduanya pun sama, yaitu untuk kemudahan dan kecepatan dalam pengambilan kembali suatu data/arsip.

Perbedaan antara basis data dan lemari arsip hanyalah pada sisi pengelola dan media penyimpanan yang digunakan. Dari sisi pengelola, lemari arsip dikelola oleh manusia. Proses penyimpanan, pilah-memilah, serta pengurutan dokumen dilakukan secara manual oleh manusia. Sedangkan pada basis data, manusia hanya berperan sebagai operator, dan mesin/komputer lah yang bertindak dalam tataran teknisnya. Berkaitan dengan media penyimpanan, pada lemari arsip, semua dokumen disimpan dan ditata rapi pada lemari arsip konvensional, berupa lemari besi atau kayu. Pada basis data, data-data tersebut disimpan secara elektronik, yaitu pada media penyimpanan seperti HDD, SSD, dll.



Gambar 2: Ilustrasi lemari arsip dan basis data

Pada penerapan di komputer, yang membedakan aplikasi basis data dengan aplikasi lain seperti spreadsheet, adalah bahwa aplikasi basis data memiliki fitur pemilahan dan pengelompokan data sesuai dengan jenis data, serta yang tidak kalah penting adalah fitur anti duplikasi. Jika hanya untuk menyimpan data saja, maka file-file berekstensi \*.xls, \*.rtf, bahkan \*.doc dan \*.txt pun bisa digunakan untuk merekam data. Hanya saja mereka tidak bisa mencegah adanya duplikasi data atau inputan yang masuk.

## Tujuan Basis Data

Selain untuk kemudahan dalam pengelolaan data, tujuan Basis Data diantaranya adalah untuk meningkatkan:

- **Speed:** kecepatan pengaksesan serta pengelolaan data.
- **Space:** efisiensi ruang penyimpanan, berkat adanya anti redundancy.
- **Accuracy:** akurasi dalam pengelolaan data, berkat adanya struktur & relasi tabel.
- **Availability:** ketersediaan data.
- **Completeness:** kelengkapan data.
- **Security:** keamanan data.

- **Sharability:** kebersamaan pemakaian.

## Basis Data Vs Spreadsheet

Basis data dan spreadsheet sebetulnya memiliki kesamaan, yaitu adanya kolom-kolom dan baris-baris yang bisa digunakan untuk menyimpan data. Salah satu contoh aplikasi spreadsheet yang melegenda yaitu Lotus 1-2-3, yang merupakan hasil karya dari IBM dan sempat mencapai masa kejayaan pada tahun 1980-1990an.

Diluar kesamaan antara keduanya tersebut, antara basis data dan spreadsheet sebetulnya memiliki perbedaan yang cukup mendasar. Berikut adalah beberapa hal yang membedakannya:

- Spreadsheet memiliki fungsi utama untuk menganalisa data, sedangkan basis data berfokus pada penyimpanan data.
- Spreadsheet tersusun dari kolom dan baris, sedangkan pada basis data terdiri dari fields dan records. Pada basis data, kolom disebut sebagai fields, sedangkan baris disebut sebagai records.
- Terkait kemampuan menyimpan data, spreadsheet tergolong kecil, sedangkan database memiliki kemampuan menyimpan data yang besar.
- Pada spreadsheet, duplikasi data sangat mungkin terjadi. Sedangkan pada basis data, peluang terjadinya duplikasi data dapat diminimalisir.
- Contoh Spreadsheet: Ms. Excel, Google Sheets, OpenOffice Calc, Zoho Sheet, dll. Sedangkan contoh database adalah MySQL, Postgre, Ms. Access, dll.

## Penerapan Basis Data

Basis data merupakan salah satu elemen penting dalam Sistem Informasi. Tidak ada sistem informasi yang bisa dibuat dan dijalankan tanpa adanya basis data. Berikut ini merupakan beberapa contoh organisasi/instansi yang memanfaatkan basis data sebagai komponen manajemen informasi:

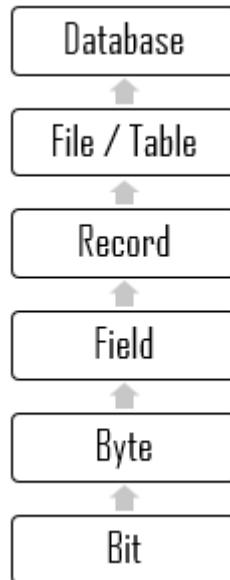
- Perbankan: basis data digunakan dalam pengelolaan data nasabah, data tabungan, data pinjaman, pembuatan laporan-laporan akuntasi hingga pelayanan informasi pada nasabah.
- Asuransi: basis data digunakan dalam pengelolaan data nasabah, data pembayaran premi, pemrosesan pengajuan klaim asuransi, dll.
- Rumah sakit: basis data digunakan dalam pengelolaan histori penyakit pasien, pengobatan pasien, hingga pembayaran perawatan pasien.
- Pendidikan: basis data digunakan dalam pengelolaan data siswa/mahasiswa, guru/dosen, karyawan, keuangan, hingga penjadualan mata kuliah.
- Dll.

## Hirarki Data

Dalam era informatika seperti saat ini, segala sesuatu melibatkan data dan data. Sangat penting bagi kita untuk memahami lebih dulu apa itu data, dan apa yang membentuk suatu data. Sebelum kita memahami sesuatu yang utuh, maka kita perlu terlebih dahulu memahami bagian-bagiannya.

Ibarat mainan lego, data pada awalnya seperti *bricks* yang belum tersusun. Ketika mereka masih tercerai-berai maka sulit untuk mendapatkan manfaat atau pengetahuan darinya. Namun, ketika *bricks* mulai disusun, maka akan dapat menghasilkan objek yang menarik dan penuh makna. Data juga demikian, yaitu perlu dikumpulkan lebih dahulu untuk kemudian disusun sedemikian rupa sehingga bisa menghasilkan informasi yang bermakna.

Sekumpulan data dalam tingkatan tertinggi disebut sebagai database. Sebuah database merupakan kumpulan dari beberapa file atau tabel. Sebuah file atau tabel dibentuk dari beberapa record (baris). Dan sebuah record berisi beberapa nilai pada field (kolom). Sebuah field, tersusun dari beberapa bytes dan byte inilah yang dibentuk dari sekumpulan bit-bit data. Demikianlah hierarki data, berawal dari bit hingga menjadi database (lihat *Gambar 3*).



Gambar 3: Hierarki data

- **Bit:** merupakan unit terkecil dari representasi data, yang berupa angka 0 dan 1. Jika 8 buah angka bit dikumpulkan, maka akan menghasilkan 1 byte, dan dapat membentuk suatu karakter atau simbol tertentu (sesuai character code). Sistem angka biner ini merupakan dasar-dasar untuk komunikasi antara manusia dengan komputer, mengingat komputer hanya mengenal kondisi 1 dan 0 atau on dan off saja.
- **Byte:** terbentuk dari kumpulan 8 angka biner (bit). Byte bisa berupa karakter, angka, maupun simbol-simbol lain sesuai dengan character code. Contoh: huruf “A” (kapital) memiliki kode biner 01000001, sedangkan huruf “a” (huruf kecil) memiliki kode biner 01100001. Dari kumpulan byte inilah, nantinya karakter bisa membentuk kata, dan kata membentuk kalimat.
- **Field:** dikenal juga dengan istilah kolom. Merupakan unit terkecil dari sesuatu yang disebut sebagai data. Field merupakan sekumpulan byte yang memiliki makna. Sebagai contoh, data “Ana”

merupakan nama seseorang dan terekam dalam field nama. “Ana” disusun dari beberapa bytes, yaitu: 01000001 01101110 01100001.

- **Record:** jika field dikenal sebagai kolom, maka record dikenal sebagai baris. Record merupakan kumpulan item data yang saling berhubungan secara lojik. Record merepresentasikan kumpulan atribut data yang menjelaskan suatu entitas. Sebuah record tersusun dari beberapa field, dimana masing-masing field menjelaskan atribut dari satu entitas.
- **File/Tabel:** merupakan kumpulan dari record-record yang terhubung. File atau tabel umumnya memiliki satu atribut yang bertindak sebagai *primary key*. Kunci ini berguna untuk mengidentifikasi dan membedakan antara satu record dengan record lain pada data file atau tabel.
- **Database:** merupakan kumpulan file atau tabel yang terintegrasi dan terhubung secara lojik. Data yang ada pada database dapat dikelola oleh software yang disebut sebagai Data-Base Management System (DBMS) atau Sistem Manajemen Basis Data (SMBD).

Setelah terkumpul pada database, suatu data menjadi mudah untuk diakses dan dimanipulasi. Bentuk database yang paling umum digunakan sekarang adalah Relational Database. Sedangkan yang semakin booming akhir-akhir ini adalah NoSQL Database.

## Bab 2

# LINGKUNGAN DAN SISTEM BASIS DATA

---

### Bab ini membahas:

- Sistem Basis Data
  - Komponen Sistem Basis Data
  - Mengenal Arsitektur Basis Data
  - Sistem Manajemen Basis Data
  - Abstraksi Data
  - Bahasa Basis Data
- 

## Sistem Basis Data

Setelah mengetahui definisi dan kegunaan basis data, berikutnya ada yang dinamakan dengan Sistem Manajemen Basis Data (SMBD) atau biasa dikenal dengan istilah Data-Base Management System (DBMS). DBMS merupakan suatu software yang memungkinkan kita untuk mengakses data pada database. Seringkali istilah basis data atau database digunakan juga untuk merujuk kepada DBMS. Adapun contoh software atau aplikasi populer yang berhubungan dengan database dan DBMS, diantaranya yaitu: MySQL, Postgre, Ms. Access, dll. (*Penjelasan lebih detail tentang DBMS dibahas di subbab lain*)

Suatu database ketika digabungkan dengan DBMS, maka akan membentuk satu istilah baru yang dikenal dengan nama Sistem Basis Data, atau Database System. Menurut KBBI, definisi sistem adalah perangkat atau unsur yang secara teratur saling berkaitan sehingga membentuk suatu totalitas.

Sebagai gambaran, sistem kendaraan bermotor terdiri dari beberapa komponen-komponen yang masing-masing memiliki fungsi sendiri-sendiri, seperti komponen pemantik, komponen kelistrikan, komponen penggerak, dan komponen pengereman. Jika masing-masing komponen tersebut dirangkai menjadi satu, maka jadilah satu sistem yang dinamakan sistem kendaraan.

Pada basis data juga demikian. Sistem Basis Data merupakan sistem yang memiliki tujuan utama untuk menyimpan dan mengelola informasi, yang setidaknya terdiri dari basis data dan DBMS. Sistem basis data setidaknya terdiri dari 4 komponen utama, yaitu data, perangkat keras (*hardware*), perangkat lunak (*software*), dan pengguna (*user*).

$$\text{Database} + \text{DBMS} = \text{Database System}$$

## Komponen Sistem Basis Data

Secara lengkap, sistem basis data terdiri dari 4 komponen utama, sebagaimana tampak pada *Gambar 4*. Berikut ini adalah penjelasan dari masing-masing komponen tersebut.

1. **Data:** dari sudut pandang user, barangkali data merupakan komponen paling penting dalam sistem basis data. Keseluruhan data pada sistem basis data tersimpan pada sebuah basis data tunggal. Data yang ada pada database tersebut dapat *di-share* dan diintegrasikan. Maksud dari berbagi data (*sharing of data*) ialah setiap data pada database *di-share* kepada beberapa user yang berbeda, dan setiap user boleh jadi memiliki akses pada setiap data tersebut namun dengan tujuan yang berbeda. Sedangkan yang dimaksud dengan integrasi data yaitu database dapat berfungsi layaknya beberapa file berbeda yang memiliki kemampuan mengontrol duplikasi (*redundancy*) antar file tersebut.
2. **Perangkat Keras (*Hardware*):** semua komponen fisik pada komputer disebut sebagai perangkat keras atau hardware. Yang dimaksud komputer di sini, bisa berupa sebuah *personal computer (PC)* saja, *minicomputer*, komputer mainframe, hingga sekumpulan komputer yang berada dalam jaringan luas, dan itu semua tergantung dari kebutuhan suatu organisasi/institusi atau ukuran database. Pada sistem basis data, perangkat keras terdiri dari dua komponen utama, yaitu:

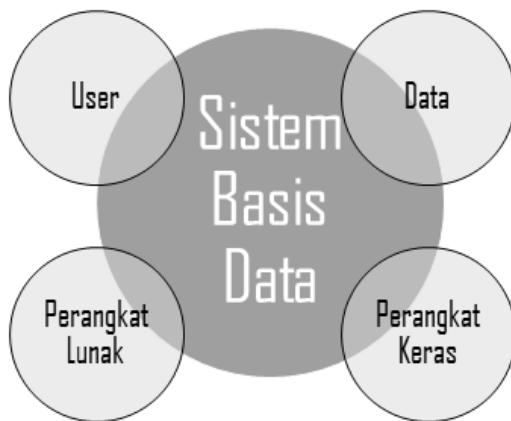
- Prosesor dan memori utama (RAM) untuk mendukung proses dan eksekusi pada DBMS.
- Media penyimpanan sekunder, seperti harddisk, magnetic disk, compact disc, dll, dimana berfungsi untuk menyimpan data dan digunakan secara bersama-sama dengan perangkat lain, seperti: input/output devices (keyboard, mouse, printer, dll), device controller, input/output channel, dll.

Sistem basis data membutuhkan memori utama serta sekunder dengan ukuran minimal tertentu agar bisa dijalankan. Dengan jumlah pengguna (*user*) yang banyak, maka ukuran RAM dan kapasitas harddisk yang besar sangatlah dibutuhkan. Hal ini berguna untuk menjaga dan mengontrol data dengan jumlah sangat banyak yang tersimpan di database. Di sisi lain, komputer dan jaringan berkecepatan tinggi dibutuhkan untuk mengeksekusi sekaligus mendapatkan data jumlah besar dalam waktu yang singkat. Perkembangan teknologi komputer (termasuk *hardware*) yang sangat pesat yang mana komputer semakin bertenaga namun justru semakin terjangkau, turut berkontribusi dalam mendorong perkembangan teknologi basis data beserta aplikasi yang mendukungnya.

3. **Perangkat Lunak (*Software*):** merupakan lapisan (*layer*) atau antar muka yang menjembatani antara *physical database* dengan user. Salah satu software yang berada pada layer ini disebut dengan DBMS. Segala aksi atau permintaan (*request*) yang dilakukan oleh user dalam kaitan dengan akses pada database, ditangani oleh DBMS. DBMS menyediakan beragam fasilitas, seperti penambahan dan penghapusan file, menarik dan mengupdate data pada suatu file, dlsb. Selain DBMS, ada juga software lain yang dibutuhkan untuk menyusun sistem basis data, yang dikenal dengan istilah *application program* atau program aplikasi. Program aplikasi merupakan software yang secara umum telah tersedia atau yang dibuat secara khusus untuk suatu perusahaan guna menyelesaikan pekerjaan-pekerjaannya. Program aplikasi umumnya dibangun dengan menggunakan bahasa pemrograman generasi ke-3 (3<sup>rd</sup> generation programming language / 3GL), seperti: C, C++, Java, Visual Basic, COBOL, Fortran, Pascal, dll, atau dengan menggunakan bahasa pemrograman generasi ke-4 (4GL), seperti SQL. Program aplikasi menggunakan fasilitas yang disediakan oleh DBMS untuk mengakses dan memanipulasi data pada database. Selain itu, software tersebut juga berperan dalam menyediakan dokumen atau laporan yang dibutuhkan oleh suatu perusahaan, sebagai suatu informasi yang berguna untuk pengambilan keputusan. Selain kedua software tersebut, piranti lunak

terakhir yang terlibat dalam sistem basis data adalah Sistem Operasi (*Operating System*). Di sini OS memainkan peranan yang sangat penting, yaitu mengelola semua komponen hardware agar software-software dapat berjalan dengan baik di komputer.

4. **Pengguna (User):** *user* merupakan orang atau sekumpulan orang yang berinteraksi dengan sistem basis data dengan cara langsung maupun tidak langsung. Sedikitnya ada 4 jenis user yang berinteraksi dengan database, yaitu: programmer aplikasi, *end user*, *database administrator* (DBA), dan desainer basis data (*database designer*). Penjelasannya adalah sebagai berikut:
  - **Database designer:** orang atau tim yang bertugas mendesain struktur basis data.
  - **Programer aplikasi (application programmer):** orang atau tim yang bertugas menulis kode program atau membangun aplikasi yang terhubung ke database, sekaligus menyesuaikannya dengan desain yang telah dibuat oleh *database designer*.
  - **End user:** orang atau tim yang berinteraksi secara langsung dengan sistem atau aplikasi dari komputer yang terhubung ke database baik secara online maupun offline. User jenis ini mengakses database melalui program aplikasi yang dibangun oleh *application programmer*. Contoh: kasir, petugas administrasi, dll.
  - **Database administrator (DBA):** orang atau tim yang bertanggung jawab terhadap semua kendali pada sistem pada tataran teknis. DBA harus merupakan orang yang benar-benar paham tentang dunia IT (*IT Professional*). DBA merupakan pengendali utama dari suatu sistem basis data yang senantiasa memantau dan mengelola segala sumber daya, seperti database, DBMS, dan program aplikasi terkait.



Gambar 4: Komponen sistem basis data

## Sistem Manajemen Basis Data

Ketika menggunakan prinsip kerja lemari arsip, maka pengelolaan dokumen-dokumen dilakukan oleh petugas terkait, dalam hal ini adalah manusia. Namun, dalam dunia elektronik ini, penyimpanan dan pengelolaan data dilakukan oleh mesin atau komputer. Di sini peran manusia hanya sebagai operator saja.

Agar manusia bisa mengakses database dan melakukan pengelolaan data di dalamnya, maka dibutuhkanlah suatu software khusus yang dikenal dengan istilah Sistem Manajemen Basis Data (SMBD) atau Data-Base Management System (DBMS).

DBMS ini berfungsi untuk menentukan bagaimana data diorganisasi, disimpan, diubah, dan diambil kembali. Selain itu, DBMS juga memiliki beberapa fitur penting lain seperti untuk manajemen keamanan hingga pemakaian data secara bersama. Berikut adalah beberapa contoh DBMS yang populer:

- Ms. Access
- MySQL
- Oracle DB
- PostgreSQL, dll

Suatu DBMS dapat digunakan untuk membentuk beragam tipe database yang sesuai dengan tujuan tertentu, seperti:

1. **Single User Database:** hanya dapat digunakan oleh 1 orang pada satu waktu. Jika user X sedang menggunakannya, maka user Y dan Z tidak dapat menggunakan DB tersebut hingga user X selesai menggunakannya.
2. **Desktop Database:** sebuah Single-User Database yang berjalan di sebuah computer personal.
3. **Multi-User Database:** suatu database yang dapat digunakan oleh beberapa user sekaligus dalam 1 waktu.
4. **Workgroup Database:** merupakan Multi-User Database dengan jumlah user yang tidak terlalu banyak, seperti 40-50 user saja, atau suatu database yang khusus untuk bagian tertentu di suatu perusahaan.
5. **Enterprise Database:** merupakan jenis database yang digunakan untuk satu perusahaan penuh dengan semua departemen-departemennya.
6. **Centralized Database:** database yang diletakkan pada satu tempat tertentu dengan user yang berada di beragam lokasi yang terpisah.
7. **Distributed Database:** database yang memiliki kemampuan untuk disebar di beragam lokasi terpisah, dan pada saat-saat tertentu dapat melakukan sinkronisasi dengan database pusat.

## Mengenal Arsitektur DBMS

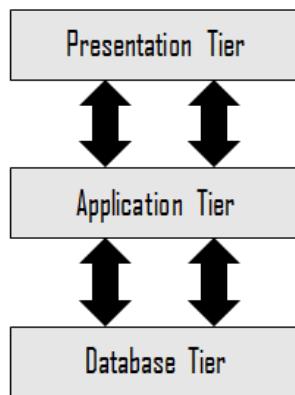
Desain dari DBMS sangat tergantung dengan arsitekturnya. Ia dapat berupa basis data yang *centralized*, *decentralized*, ataupun *hierarchical*. Arsitektur suatu DBMS bisa berupa *1-tier*, *2-tier*, ataupun *3-tier*. Sebuah basis data dengan arsitektur *n-tier* membagi keseluruhan sistem menjadi sejumlah *n* modul yang independen namun tetap saling terkait.

Pada arsitektur *1-tier*, DBMS merupakan satu-satunya entitas dimana pengguna berinteraksi sekaligus menggunakannya. Segala macam bentuk perubahan atau modifikasi langsung dilakukan melalui DBMS itu sendiri. Arsitektur ini tidak menyediakan *tool* yang khusus digunakan oleh *end-user* untuk memudahkan penggunaan karena keawaman

mereka tentang hal-hal teknis terkait basis data. Namun, arsitektur ini justru cocok dan disukai oleh deainer basis data dan juga programmer.

Sedangkan pada arsitektur *2-tier*, DBMS harus menyediakan aplikasi untuk mempermudah akses data ke dalam basis data. Dalam pengembangan suatu sistem, programmer akan menggunakan arsitektur *2-tier* apabila dibutuhkan media aplikasi guna mengakses basis data. Di sini, aplikasi yang dibangun tersebut sepenuhnya bersifat independen dari suatu basis data, khususnya dalam hal pegaseran, desain, dan pemrogramannya.

Selanjutnya terkait dengan arsitektur *3-tier*, ia memisahkan masing-masing tingkatannya berdasarkan kompleksitas dari pengguna serta bagaimana mereka menggunakan data yang ada di dalam basis data. Arsitektur ini merupakan salah satu arsitektur yang paling umum digunakan untuk mendesain suatu DBMS. Gambaran arsitektur *3-tier* tampak pada *Gambar 5*.



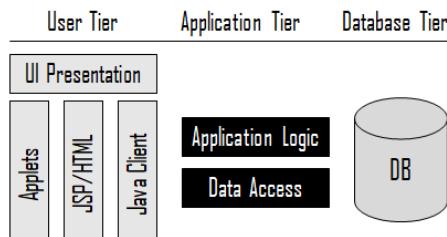
**Gambar 5: Ilustrasi arsitektur 3-tier**

Penjelasan dari ilustrasi pada *Gambar 5* adalah sebagai berikut:

- **Database (Data) Tier.** Ini merupakan tingkatan yang pertama dimana basis data terletak bersama dengan bahasa pemrosesan querinya. Pada tahap ini juga ada relasi yang mendefinisikan suatu data beserta aturan-aturannya (*constraints*). Database tier terdiri dari basis data dan suatu program untuk mengatur akses baca-tulis ke basis data. *Layer* ini juga dikenal dengan nama *Storage Tier*, dan dapat berada baik di lokasi *on-*

*premises* (contoh: PC) ataupun di *Cloud*. Beberapa *tool* populer yang berguna untuk mengelola akses *read/write* ke basis data, diantaranya yaitu: MySQL, PostgreSQL, Microsoft SQL Server dan MongoDB.

- **Application (Middle) Tier.** *Application Tier* dikenal pula dengan nama lain yaitu *Logic Tier*. Pada level ini terdapat program yang berfungsi untuk mengakses basis data, yang umumnya dibuat dengan menggunakan bahasa pemrograman, seperti: python, java, hingga PHP. Di sini, aplikasi memiliki peran penting karena memuat *business logic* yang disesuaikan dengan kebutuhan *stake-holder*. Aplikasi tersebut dapat diletakkan di server-server yang tersebar di *cloud* ataupun server-server biasa seperti pada umumnya yang tersimpan di kantor, tergantung pada seberapa besar kekuatan pemrosesan (*processing power*) yang dibutuhkan oleh aplikasi tersebut. *Layer* aplikasi ini umumnya berada di tengah-tengah sekaligus menjadi perantara antara pengguna akhir (*end-user*) dengan database. Terkadang *end-user* bahkan merasa tidak sadar akan keberadaan database dibalik aplikasi yang mereka gunakan.
- **User (Presentation) Tier.** Pengguna akhir (*end-user*) umumnya beroperasi pada layer ini dan kebanyakan dari mereka justru tidak menyadari akan keberadaan database dibalik program aplikasi yang mereka gunakan. Di level ini, antarmuka untuk pengguna akhir umumnya dibuat dengan HTML5, Cascading Style Sheets (CSS) hingga JavaScript. Antarmuka pada layer ini dibuat agar bisa dibuka di beragam *computing devices* (seperti komputer atau handphone) tentunya melalui web browser, sehingga aplikasi tersebut dikenal dengan nama aplikasi berbasis web (*web-based application*). Namun tidak menutup kemungkinan aplikasi yang dibuat di sini masuk ke dalam kategori *desktop-based application* yaitu aplikasi desktop yang perlu diinstal terlebih dahulu pada *device* yang dituju. *Presentation tier* ini dapat berkomunikasi dengan *tier-tier* yang lain dengan memanfaatkan API (*Application Programming Interface*).



Gambar 6: Lebih rinci tentang arsitektur 3-tier

Arsitektur *3-tier* ini merupakan salah satu arsitektur yang paling umum digunakan, mengingat adanya beberapa keunggulan dibanding arsitektur-arsitektur lain, khususnya yang berkaitan dengan kemudahan dalam melakukan perluasan sistem secara horizontal (*horizontal scalability*), serta peningkatan performa dan ketersediaan data (*availability*). Dengan arsitektur jenis ini, masing-masing bagian (*tier*) memungkinkan untuk dibangun secara bersama-sama, dalam waktu yang sama (*concurrently*) oleh tim programmer yang berbeda, sekalipun dengan menggunakan bahasa pemrograman yang berbeda pula. Dengan keunggulan ini, maka apabila ada sebuah *tier* yang hendak dimodifikasi, maka hal tersebut dapat dilakukan tanpa berpengaruh pada *tier-tier* yang lain. Arsitektur *3-tier* menjadikan segalanya lebih mudah khususnya untuk perusahaan besar atau *software packager* untuk terus membenahi suatu aplikasi seiring dengan meningkatnya kebutuhan dan tantangan suatu perusahaan/instansi. Aplikasi-aplikasi yang sudah ada tersebut (atau bisa juga sebagian dari aplikasi tersebut) dapat secara permanen atau sementara tetap digunakan kemudian dibungkus hingga menjadi komponen pada *tier* baru.

## Abstraksi Data

Abstraksi data (*data abstraction*) adalah upaya penggambaran suatu proses/data dengan cara menampilkan beberapa karakteristik yang dibutuhkan, namun di sisi lain juga menyembunyikan beberapa detil atau bagian yang tidak dibutuhkan. Kebutuhan akan abstraksi data sangat tergantung dari sudut pandang seseorang yang disesuaikan dengan peran dari orang tersebut. Singkatnya, seseorang pada peran tertentu hanya perlu fokus pada suatu abstraksi pada level tertentu saja. Pengguna dari sistem basis data hanya menggunakan data sesuai porsinya. Mereka tidak perlu untuk mengetahui struktur data yang kompleks dan bagaimana suatu data tersebut disimpan. Oleh karena itu, hal-hal yang sekiranya tidak penting atau tidak dibutuhkan oleh pengguna dapat disembunyikan (tidak ditampilkan). Inilah yang dinamakan dengan abstraksi data.

Ada tiga level pada abstraksi data, yaitu:

1. **Level Eksternal (*External Level*)**. Nama lain dari level ini yaitu level pandangan/penampakan (*view level*). Abstraksi data pada level ini fokus pada sesuatu yang terlihat oleh pengguna sistem informasi (pada kasus berikut ini terdiri dari peminjam dan petugas perpustakaan). Jenis pengguna yang berbeda seringkali memiliki tampilan *user interface* yang berbeda pula. Tampilan yang disediakan tersebut mewakili porsi

data yang dibutuhkan oleh pengguna, sesuai dengan tingkatannya. Oleh karena itu, peran dari level eksternal ini ialah memberikan gambaran tentang bagaimana pengguna melihat suatu data. Beberapa bentuk umum yang termasuk dalam *user view* ini adalah: form input data, halaman website, laporan, dll. Sedikitnya ada 3 alasan mengapa level eksternal ini sangat dibutuhkan, yaitu:

- a. Dengan level eksternal ini, pengguna yang berbeda hanya akan memperoleh porsi data yang mereka butuhkan saja (sesuai perannya). Hal demikian akan menyederhanakan interaksi antara pengguna dan sistem basis data.
  - b. Adanya level ini dapat mencegah pengguna awam untuk mengakses bagian-bagian lain dari database yang tidak seharusnya bisa diakses oleh pengguna awam. Dengan demikian, tingkat keamanan sistem dapat ditingkatkan.
  - c. Level ini menyembunyikan struktur basis data yang rumit dari pengguna awam mengingat mereka tidak ada kebutuhan untuk melihat struktur tersebut. Hal ini dapat memudahkan pengguna awam untuk memahami tugas mereka secara lebih sederhana.
2. **Level Konseptual (*Conceptual Level*)**. Level yang biasa disebut juga dengan level logis (*logical level*) ini diperankan oleh desainer basis data (*database designer*) dan administrator basis data (*database administrator/DBA*). Level konseptual terdiri dari struktur logis (*logical structure*) secara lengkap dari suatu basis data. Salah satu *tool* yang sering digunakan untuk mendesain struktur logis pada level konseptual ini, yaitu: Entity-Relationship (ER) Diagram (ERD). Sebagai contoh, pada basis data relasional, struktur logis ialah gambaran dari semua tabel yang terlibat beserta relasi-relasi antar mereka. Orang yang bekerja pada level ini tidak perlu tahu bagaimana struktur logis tersebut secara fisik disimpan di dalam media penyimpanan. Sedikitnya ada dua alasan kenapa level konseptual sangat penting, yaitu:
    - a. Level ini memfasilitasi desainer basis data dan DBA dalam mendesain dan melihat struktur logis dari suatu basis data.
    - b. Level konseptual menyembunyikan rincian-rincian tentang bagaimana data dan struktur basis data direpresentasikan dan disimpan di media penyimpanan.
  3. **Level Fisik (*Physical Level*)**. Level ini merupakan level terendah pada konsep abstraksi data, dan biasa disebut juga dengan istilah level *internals*. Level fisik fokus pada bagaimana data dan struktur basis data

secara fisik disimpan di dalam media penyimpanan (misal: harddisk). Dengan kata lain, level fisik berhubungan langsung dengan implementasi fisik dari suatu basis data, dan level ini menjelaskan tentang bagaimana DBMS dan Sistem Operasi melihat suatu data. Sebagai pengingat, DBMS merupakan software yang memungkinkan seseorang untuk membuat, mengelola dan mengakses basis data. Untuk basis data relasional, level fisik terkait dengan bagaimana tabel-tabel dan *relationship* antar tabel tersebut disimpan di media penyimpanan. Implementasi dari level fisik ini tergantung dari DBMS yang digunakan. Beberapa hal yang perlu diperhatikan pada level fisik ini, diantaranya yaitu: alokasi kapasitas media penyimpanan, organisasi file pada basis data, kemampuan membaca dan menulis data pada media penyimpanan, teknik kompresi dan enkripsi data, dll. Ada setidaknya dua alasan mengapa level fisik ini sangat penting:

- a. Level fisik dibutuhkan dalam melakukan implementasi dari sistem basis data.
- b. Level ini berkaitan erat dengan bagaimana struktur logis dari suatu basis data secara fisik diimplementasikan pada media penyimpanan.

Secara lebih singkat, manfaat dari penggunaan abstraksi data ini dapat disimpulkan sebagai berikut:

1. Abstraksi data memungkinkan adanya *user view* yang independen. Jenis pengguna yang berbeda akan melihat porsi data yang berbeda dengan bentuk tampilan data yang berbeda pula.
2. Abstraksi data menyembunyikan struktur logis basis data yang rumit sekaligus detail penyimpanan fisik basis data di media penyimpanan, sehingga pengguna awam dan DBA (Database Administrator/DBA) tidak perlu melihatnya.
3. Adanya abstraksi data memungkinkan DBA untuk mengubah struktur basis data tanpa mempengaruhi tampilan pada pengguna/*user*. Hal ini dikenal dengan istilah *logical data independence*.
4. Abstraksi data memungkinkan adanya perubahan pada media penyimpanan tanpa mempengaruhi struktur basis data. Kelebihan ini dikenal dengan istilah *physical data independence*.
5. Dari poin nomor 3 dan 4 di atas, maka dapat disimpulkan bahwa aspek fisik dari suatu basis data dirubah tanpa mempengaruhi pengguna pada level eksternal.

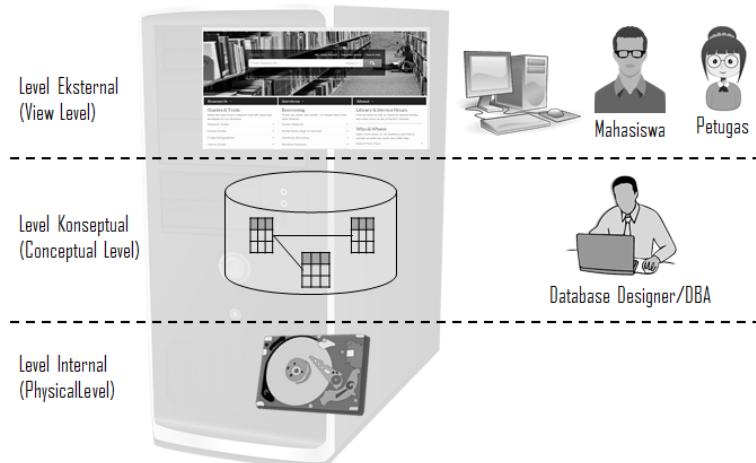
Untuk memudahkan dalam memahami karakteristik masing-masing level di atas, kita akan melakukannya dengan menggunakan studi kasus pada sistem informasi perpustakaan kampus. Skenarionya adalah sebagai berikut:

Sistem informasi perpustakaan sekolah memiliki sebuah basis data yang menyimpan data-data terkait dengan informasi buku, pengguna/peminjam, dan data peminjaman. Para pengguna perpustakaan dapat mengakses sistem informasi perpustakaan melalui *web browser* dan jaringan internet di kampus.

Ada beberapa orang yang bekerja pada beberapa bagian yang berbeda di sistem informasi perpustakaan tersebut, di antaranya yaitu:

1. Peminjam (*users*)
2. Petugas perpustakaan (*librarians*)
3. Desainer basis data (*database designers*)
4. Administrator basis data (*database administrators*)
5. Dll.

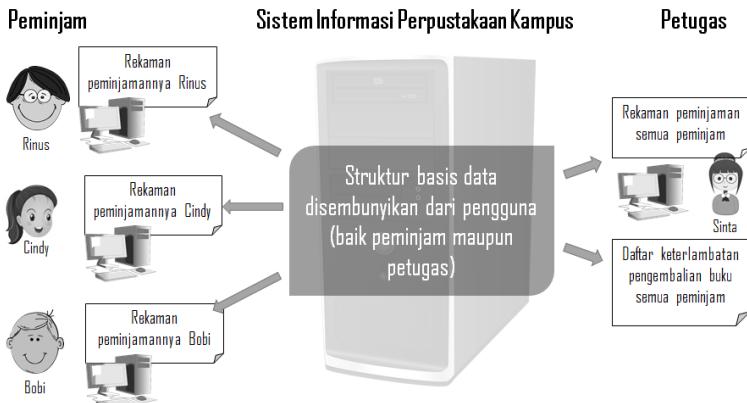
Masing-masing kelompok orang di atas memiliki peran yang berbeda-beda, termasuk dalam setiap levelnya. Berikut akan dijelaskan peran masing-masing pihak di setiap level.



Gambar 7: Ilustrasi tiga level abstraksi data

### Level Eksternal (*View Level*)

Pada tingkatan ini, setidaknya ada dua pihak yang terlibat, dan peran dari masing-masing pihak tersebut adalah sebagai berikut:



**Gambar 8: Ilustrasi cara pandang pengguna (peminjam dan petugas) terhadap basis data sistem informasi perpustakaan kampus.**

1. **Peminjam:** mereka hanya membutuhkan informasi tentang rekaman peminjaman mereka dan buku-buku yang dikoleksi oleh perpustakaan. Mereka tidak membutuhkan rekaman peminjaman orang lain, apalagi data keterlambatannya.

Data Peminjaman						
Perpanjang	Kode Buku	Judul	Penulis	Tgl Pinjam	Batas Pengembalian	
<input type="checkbox"/>	X12.2009.1293	Mencegah Virus Dengan Tangan Kosong	Fauzi Adi Rafastara	20 Juli 2019	3 Agustus 2019	<a href="#">Perpanjang</a> Beri tanda cek pada buku yang akan diperpanjang, kemudian klik tombol Perpanjang
<input type="checkbox"/>	X12.2014.1321	Basis Data	Fathansyah	20 Juli 2019	3 Agustus 2019	
<input type="checkbox"/>	X13.2017.1001	Network Security Through Data Analysis	Michael Collins	20 Juli 2019	3 Agustus 2019	

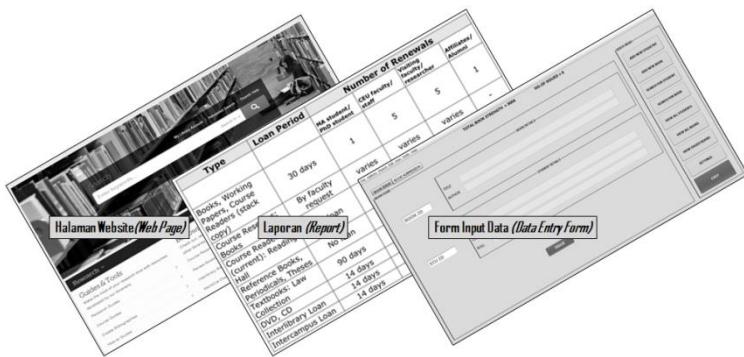
**Gambar 9: Contoh user interface dan data-data yang bisa dilihat oleh peminjam**

2. **Petugas perpustakaan:** mereka butuh untuk dapat mengakses rekaman peminjaman dari semua pengguna/peminjam dengan cara memasukkan kode peminjam. Selain itu, mereka juga perlu untuk dapat melihat sekaligus mencetak data keterlambatan berdasarkan tanggal.

<b>Perpustakaan Kampus Biru</b>						
Laporan Keterlambatan Pengembalian Buku						
Tanggal: 25 Juli 2019						
No.	Kode Buku	Judul	Kode Peminjam	Nama Peminjam	Batas Pengembalian	Keterlambatan
1.	X12.2005.0085	MySQL Untuk Pemula	A0012	David Agung	18 Juli 2019	7 Hari
2.	X13.2016.0012	Database Concept	A0230	Reza Adi	19 Juli 2019	6 Hari
3.	X13.2014.0251	Cyberworld China Embraces the Internet	A0087	Munif Uwais	21 Juli 2019	4 Hari
4.	X11.2007.0011	Matematika Terapan untuk Bisnis dan Ekonomi	A0081	Mecca Arzabania	21 Juli 2019	4 Hari
5.	X11.2001.0047	Teori Bahasa dan Otomata	A0081	Mecca Arzabania	21 Juli 2019	4 Hari
6.	X13.2017.0061	Practical Statistics for Data Scientists	A0081	Mecca Arzabania	21 Juli 2019	4 Hari

**Gambar 10:** Contoh laporan dengan data-data yang bisa dilihat oleh petugas perpustakaan

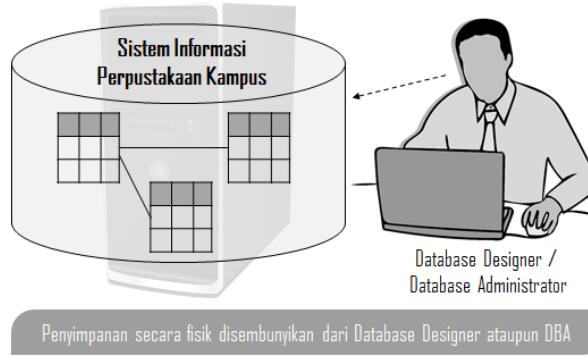
3. Baik peminjam maupun petugas perpustakaan tidak membutuhkan informasi terkait dengan struktur basis data dan bagaimana data disimpan.



**Gambar 11:** Contoh ragam bentuk *view* dari pengguna (peminjam dan petugas)

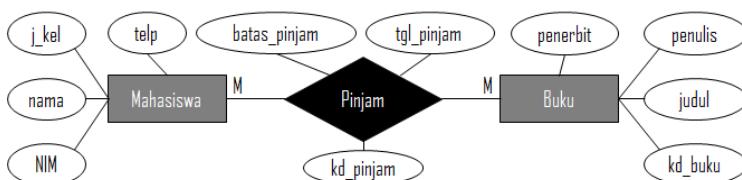
### Level Konsep (*Conceptual Level*)

Berikut merupakan penjelasan terkait dengan pihak-pihak yang terlibat di dalam level ini:



**Gambar 12: Ilustrasi cara pandangan seorang designer basis data atau DBA terhadap database sistem informasi perpustakaan kampus**

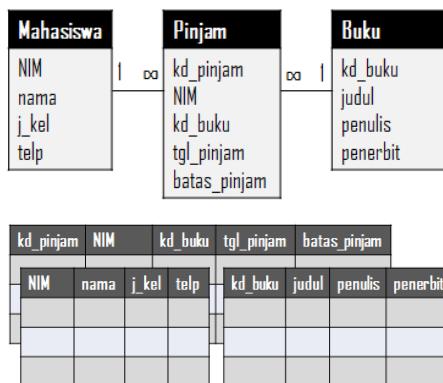
1. Di level ini melibatkan dua jenis pihak lain yang terlibat dalam sistem basis data, yaitu desainer basis data (*database designer*) dan administrator basis data (*database administrator*).
2. **Desainer basis data:** merupakan orang yang mendesain dan membuat sistem basis data.
3. **Administrator basis data:** merupakan orang yang bertanggung jawab dalam memelihara dan mengawasi/memantau sistem basis data. Peran seperti ini seringkali disebut sebagai DBA (*Database Administrator*).
4. Terkadang peran sebagai database designer dan DBA diperankan oleh orang yang sama.
5. Baik desainer maupun DBA harus memahami dan mengerti tentang struktur basis data. Keduanya pun harus mengetahui data apa yang disimpan dan bagaimana data direlasikan ke basis data. Sebagai contoh, dalam basis data relasional, yang perlu mereka kuasai yaitu struktur tabel dan hubungan antar tabel-tabelnya.
6. Meskipun keduanya harus menguasai struktur dan relasi data yang ada, namun keduanya tidak perlu mengetahui lebih dalam dari itu, seperti bagaimana tabel-tabel tersebut secara fisik berada dan disimpan di dalam media penyimpanan (misal harddisk).



**Gambar 13: Contoh ERD sistem informasi perpustakaan kampus (level konseptual)**

Mahasiswa (NIM, nama, j_kel, telp)
Buku (kd_buku, judul, penulis, penerbit)
Pinjam (kd_pinjam, NIM, kd_buku, tgl_pinjam, batas_pinjam)

**Gambar 14: Contoh desain logis (*logical design*) dari sistem informasi perpustakaan kampus**



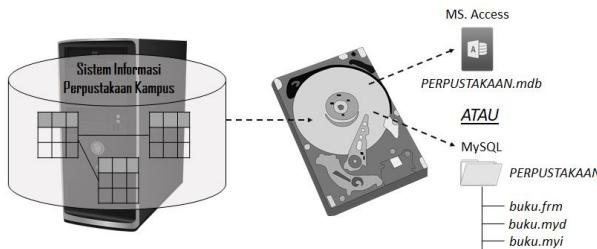
**Gambar 15: Struktur tabel dan relasinya**

### Level Internal (*Physical Level*)

Terkait dengan skenario tentang sistem informasi perpustakaan, penjelasan dari abstraksi data pada level fisik adalah sebagai berikut:

1. Level ini membahas tentang bagaimana suatu basis data relasional dalam sistem informasi perpustakaan direpresentasikan dan disimpan di dalam media penyimpanan.
2. Apa yang dikerjakan di dalam level fisik ini sangat tergantung dengan DBMS (*Database Management System*) yang digunakan.
3. Sebagai contoh, apabila suatu basis data dibangun menggunakan aplikasi Microsoft Access, maka basis datanya berupa suatu file yang berekstensi \*.mdb (contoh: *PERPUSTAKAAN.mdb*). File \*.mdb tersebut menyimpan semua informasi dan tabel-tabel yang terlibat (contoh: tabel *pengguna*, *buku*, dan *data\_peminjaman*).
4. Contoh lain, apabila basis data dibuat di dalam MySQL, maka basis datanya ialah berupa folder (misal folder PERPUSTAKAAN). Di dalam folder ini tersimpan tabel-tabel dan segala informasi yang terkait

dengan basis data. Untuk tabel *buku* di dalam basis data, aka noda sekumpulan file dengan beberapa ekstensi, seperti: *buku frm*, *buku myd*, dan *buku myi*.



**Gambar 16: Ilustrasi keberadaan (secara fisik) database *PERPUSTAKAAN* di media penyimpanan**

## Bahasa Basis Data

Guna menyediakan beragam fasilitas untuk jenis *user* yang berbeda-beda, suatu DBMS umumnya menyediakan satu atau lebih bahasa pemrograman khusus yang disebut dengan bahasa Basis Data atau DBMS. Basis data atau DBMS secara umum mendukung beberapa jenis bahasa berikut ini:

1. **Data Definition Language (DDL).** Bahasa jenis ini digunakan untuk membuat atau menghapus basis data. Selain itu, perintah-perintah DDL juga berfungsi untuk membangun, memodifikasi struktur hingga menghapus tabel juga objek-objek lain di dalam basis data. Berikut merupakan perintah-perintah yang termasuk ke dalam DDL.

No.	Perintah	Fungsi
1.	CREATE	Membuat tabel baru, tabel virtual ( <i>view</i> ) atau objek-objek lain di dalam basis data.
2.	ALTER	Memodifikasi objek-objek basis data, termasuk tabel.
3.	DROP	Menghapus keseluruhan tabel, tabel virtual ( <i>view</i> ) atau objek-objek lain di dalam basis data.

**Tabel 1: Daftar perintah DDL.**

2. **Data Manipulation Language (DML).** DML digunakan untuk memanggil, menambah, juga memodifikasi informasi pada suatu basis

data. Singkatnya, DML berguna untuk kepentingan memanipulasi data di suatu tabel. Contoh perintah-perintah yang masuk kategori DML, bisa dilihat pada *Tabel 2*.

No.	Perintah	Fungsi
1.	INSERT	Membuat baris ( <i>record</i> ) baru pada tabel.
2.	UPDATE	Memodifikasi baris ( <i>record</i> ) pada tabel.
3.	DELETE	Menghapus baris ( <i>record</i> ) pada tabel.

**Tabel 2: Daftar perintah DML.**

DML sendiri dibedakan menjadi dua, yaitu Procedural DML dan Non-Procedural DML (perhatikan *Tabel 3*).

- a. **Procedural DML** membutuhkan seorang *user* untuk menentukan data apa yang dibutuhkan serta bagaimana mendapatkannya. DML jenis ini biasa disebut *low level DML*. *Procedural DML* mengambil data *record* satu-per-satu dan memproses masing-masing secara terpisah. *Relational Algebra (RA)* adalah contoh dari procedural DML.
- b. **Non-Procedural DML** membutuhkan seorang *user* hanya untuk menentukan data apa yang dibutuhkan saja. DML jenis ini tidak perlu diberitahu tentang bagaimana cara mendapatkan data. Istilah lain dari DML jenis ini ialah *high level DML*. Procedural PostgreSQL ialah salah satu contoh dari *non-procedural DML*.

No.	Procedural DML	Non-Procedural DML
1.	Menentukan bagaimana output/hasil dari perintah DML diperoleh.	Menentukan output apa yang hendak diperoleh.
2.	<i>Embedded</i> pada Bahasa tingkat tinggi ( <i>high level language</i> )	Secara langsung melakukan <i>query</i> pada DBMS untuk menemukan data.
3.	Memperlakukan baris atau <i>record</i> secara terpisah.	Dapat beroperasi pada sekumpulan <i>records</i> .

---

4.	Tipe basis data <i>Network</i> dan <i>Hierarchical</i> umumnya bersifat procedural.	Relational DBMS menggunakan DML jenis ini untuk memperoleh data.
5.	Sulit untuk digunakan dan dipelajari.	Secara umum lebih mudah untuk dipelajari dan digunakan karena <i>user</i> hanya melakukan sedikit pekerjaan saja, sisanya dilakukan oleh DBMS.
6.	Contohnya termasuk PL/SQL.	Contohnya termasuk SQL.

**Tabel 3: Perbedaan DML Prosedural dan Non-Prosedural**

3. **Data Control Language (DCL).** DCL merupakan pelengkap dari DDL yang berperan dalam menambah dan mengurangi objek pada basis data, serta DML yang berperan dalam menambah, memodifikasi, dan menghapus data pada tabel. Perintah-perintah yang termasuk ke dalam DCL adalah sebagai berikut:

No.	Perintah	Fungsi
1.	GRANT	Memberikan hak khusus ( <i>privilege</i> ) kepada pengguna ( <i>user</i> ).
2.	REVOKE	Membatalkan hak khusus ( <i>privilege</i> ) yang telah diberikan kepada <i>user</i> .

**Tabel 4: Daftar perintah DML.**

4. **Data Query Language (DQL).** Perintah ini digunakan untuk menampilkan data dari sebuah tabel ataupun beberapa tabel. Perintah yang tergolong sebagai DQL adalah sebagai berikut:

No.	Perintah	Fungsi
1.	SELECT	Mengambil beberapa baris ( <i>records</i> ) dari satu atau beberapa tabel.

**Tabel 5:** Daftar perintah DQL.

## Bab 3

# MODEL DATA RELASIONAL

---

### Bab ini membahas:

- Desain Basis Data
  - Pemodelan Data
  - Model E-R
  - Model Relasional
- 

Database hadir karena adanya kebutuhan untuk merubah data menjadi informasi. Data merupakan fakta mentah yang belum diproses. Sedangkan informasi diperoleh dari hasil memproses data hingga menjadi sesuatu yang bermakna. Sebagai contoh, pada beberapa tahun lalu sebelum dunia telekomunikasi sedemikian canggih, buku telepon yang dikenal dengan nama *Yellow Pages* menjadi sesuatu yang wajib dimiliki oleh orang-orang. Ia bagaikan basis data yang menyimpan semua nomor telepon beserta nama yang mudah untuk dicari.

Pada *Yellow Pages* ini, ribuan atau bahkan jutaan nama dan nomor telepon tersebut merupakan suatu data. Di sini, data tersebut belum menjadi informasi apa-apa, hanya tersedia begitu saja dan menunggu untuk dimanfaatkan. Apabila terjadi kebakaran di suatu tempat, kemudian dari buku daftar telefon tersebut diperoleh nomor telepon dinas pemadam kebakaran, maka nomor telepon itulah yang dinamakan sebagai informasi atau sesuatu yang bermakna.

Basis data merupakan tempat penampungan fakta-fakta yang sangat besar yang didesain sedemikian rupa sehingga mampu dengan mudah memproses fakta-fakta tersebut menjadi informasi. Pada kasus pemadam kebakaran yang disampaikan sebelumnya, apabila buku telepon dibuat dengan struktur yang kurang nyaman, dimana nama dan nomor telepon disusun berdasarkan tanggal nomor telepon dikeluarkan oleh pihak Telkom (misal), maka proses konversi data menjadi informasi barangkali menjadi sangat sulit. Tentu saja jarang dari kita mengetahui dengan pasti kapan dinas pemadam kebakaran memperoleh

nomor teleponnya. Akibatnya kita bisa saja mencarinya halaman demi halaman hingga kita menemukannya. Dan pada saat menemukan nomor yang dituju, barang kali rumah yang terbakar sudah tinggal menjadi abu, akibat lamanya proses pencarian satu nama dalam satu buku saja. Proses pencarian tersebut terjadi sangat lama akibat desain susunan nama dan nomor telepon berbasiskan tanggal nomor telepon dibuat. Singkatnya, di sini terjadi permasalahan desain buku daftar telepon. Beruntungnya, buku telepon yang ada selama ini, pengurutannya berbasiskan huruf abjad pada nama pemilik nomor telepon, bukan berdasarkan kapan nomor telepon dikeluarkan.

Pada zaman sekarang, dimana informasi sudah mudah diakses secara digital, kesalahan desain pada ilustrasi keadaan di atas bisa lebih diminimalisir dengan melibatkan basis data. Bab ini akan mempelajari tentang desain basis data, yang diantaranya adalah Model E-R dan Model Data Relational.

## Desain Basis Data

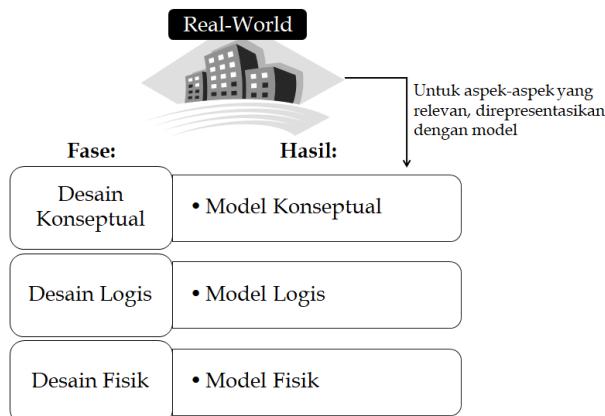
Desain basis data merupakan suatu kegiatan atau proses dalam memproduksi model data yang mendetail pada suatu basis data. Kegiatan ini memungkinkan kita untuk menggambarkan entitas-entitas nyata (*real-world entities*) ke dalam bentuk yang dapat dimengerti dan diproses oleh komputer. Dalam mendesain basis data, dibutuhkan pemahaman yang baik tentang suatu organisasi, baik terkait dengan kebutuhan operasional maupun kebutuhan bisnis. Disamping itu, kemampuan dalam memodelkan dan merealisasikan kebutuhan-kebutuhan tersebut ke dalam bentuk basis data juga wajib dimiliki oleh seorang yang berprofesi sebagai *database designer*.

Ada beberapa tujuan utama dalam mendesain basis data, diantaranya yaitu:

- **Comprehensive.** Suatu database harus didesain secara komprehensif, artinya memuat semua data dan hubungan-hubungan yang dibutuhkan.
- **Understandable.** Struktur basis data harus dibuat sejelas mungkin agar mudah, fleksibel, dan cepat dalam membaca serta memperbarui (*update*) data.
- **Expandable.** Memungkinkan untuk merubah struktur basis data dengan efek perubahan seminimal mungkin terhadap software atau aplikasi yang terkait.
- **Dapat digunakan di beragam organisasi.** Basis data harus dapat diadaptasikan ke dalam beragam jenis lingkungan dan pengguna, tanpa perlu melakukan perubahan pada struktur basis data yang ada.

- **Integritas data.** Data harus benar, tepat, dan konsisten.

Proses mendesain basis data secara umum dibagi ke dalam 3 tahap, yaitu: tahap desain konseptual, desain logis, dan desain fisik.



Gambar 17: Tahapan-tahapan dalam mendesain basis data

## Desain Konseptual (*Conceptual Design*)

Tahap pertama dalam desain basis data adalah *desain basis data konseptual* atau biasa disebut dengan *desain konseptual* saja. Desain konseptual merupakan proses membuat model data yang digunakan dalam desain suatu basis data. Pada tahap ini, pemodelan yang dilakukan adalah pemodelan konseptual yang mana merupakan langkah penting dalam mendesain basis data yang berkualitas. Pemodelan konseptual biasa disebut juga dengan pemodelan semantik (*semantic modelling*). Fokus dari pemodelan ini bukan pada ‘operasi apa yang harus dilakukan pada suatu data’, melainkan pada ‘data apa yang dibutuhkan’ dan ‘bagaimana data tersebut diorganisir’. Tujuan dari tahap desain ini diantaranya adalah untuk mengidentifikasi entitas, hubungan (*relationship*), dan atribut penting yang terlibat.

Model data dibangun dengan menggunakan informasi-informasi yang didokumentasikan dalam spesifikasi kebutuhan pengguna (*users' requirements specification*). Desain konseptual ini sepenuhnya terbebas dari detail-detail terkait implementasi, seperti software DBMS, program aplikasi, bahasa pemrograman, hardware, atau kebutuhan-kebutuhan teknis lainnya.

Dalam membangun model data konseptual ini, model harus diuji dan divalidasi terhadap kebutuhan pengguna (*users' requirements*) untuk memastikan ketepatannya. Model ini merupakan sumber informasi untuk model pada tahap berikutnya, yaitu desain basis data logis (*logical database design*).

Langkah-langkah yang perlu dilakukan dalam tahap desain konseptual ini adalah sebagai berikut:

1. Mengidentifikasi tipe entitas.
2. Mengidentifikasi tipe hubungan (*relationships*).
3. Mengidentifikasi serta mengasosiasikan atribut dengan tipe entitas maupun hubungan.
4. Menentukan domain atribut.
5. Menentukan atribut yang bertindak sebagai kunci (*key*) kandidat (*candidate key*), primary (*primary key*), maupun alternatif (*alternate key*).
6. Mempertimbangkan penggunaan konsep pemodelan tingkat lanjut (opsional).
7. Memeriksa keberadaan redundansi pada model.
8. Memvalidasi model konseptual terhadap transaksi pengguna.
9. Mereview model konseptual bersama dengan pengguna.

## Desain Logis (*Logical Design*)

Langkah ke-dua dalam mendesain basis data adalah mendesain basis data logis (*logical database design*). Pada tahap ini akan menghasilkan model data logis (*logical data model*), yang mana model data konseptual yang telah dibuat sebelumnya mengalami perbaikan dan dikonversi menjadi model data logis. Model logis ini masih terbebas dari hal-hal bersifat fisik, seperti struktur dan index penyimpanan data. Contoh dari model data logis ini, diantaranya adalah: model Relasional, Network, Hierarchical, Object Oriented, dan juga Entity-Relationship (E-R).

Dalam membangun model data logis, model harus diuji dan divalidasi terhadap kebutuhan pengguna (*users' requirements*). Metode normalisasi (dibahas di Bab 8) digunakan untuk menguji kebenaran atau ketepatan model data logis. Normalisasi berguna untuk memastikan bahwa suatu relasi pada model data tidak mengandung redundansi, yang notabene dapat menyebabkan anomali, salah satunya pada saat proses update.

Selain itu, model data logis harus diperiksa pula untuk memastikan bahwa ia sesuai dengan scenario atau transaksi yang ditetapkan oleh *user*. Model ini merupakan sumber informasi penting untuk menuju ke tahap berikutnya, yaitu desain basis data fisik (*physical database design*), yang menyediakan amunisi dan keperluan yang dibutuhkan oleh perancang basis data (*database designer*) dalam membuat suatu rancangan yang baik, tepat dan efisien. Model ini juga memainkan peranan penting di dalam kegiatan pemeliharaan rutin atau kegiatan operasional lain, sebagaimana terdapat pada *Database System Development Lifecycle*. Dengan adanya model data yang baik, kelak jika dibutuhkan adanya perubahan pada program aplikasi maupun pada data di masa mendatang, maka perubahan tersebut dapat diakomodir dengan sangat cepat dan tepat.

Langkah-langkah yang perlu di lakukan dalam tahap desain logis ini adalah sebagai berikut:

1. Mendapatkan relasi dari model data logis.
2. Memvalidasi relasi menggunakan normalisasi.
3. Memvalidasi relasi terhadap transaksi pengguna.
4. Memeriksa aturan integritas (*integrity constraints*).
5. Memeriksa model data logis bersama pengguna (*users*).
6. Menggabungkan beberapa model data logis menjadi model global (*optional*).
7. Memeriksa kehandalan model logis ini untuk beradaptasi terhadap kemungkinan-kemungkinan perubahan di masa mendatang.

## **Desain fisik (*Physical Design*)**

Pada tahap ini, rancangan atau hasil dari tahap desain sebelumnya mulai diaplikasikan pada media penyimpanan sekunder (cth: harddisk). Tahap desain fisik ini lebih banyak berfokus pada relasi-relasi dasar, organisasi file, dan pembuatan indeks untuk memperoleh akses terhadap data secara efisien. Selain itu, hal yang tidak kalah penting untuk diperhatikan ialah masalah aturan integritas (*integrity constraints*) dan kemanan.

Tahap ini merupakan tahap ketiga sekaligus tahap akhir dalam proses mendesain basis data. Di sini desain basis data memikirkan tentang bagaimana suatu basis data diimplementasikan, dengan mengacu pada hasil yang diperoleh dari tahap sebelumnya, yaitu model data logis (seperti model data relasional dan E-R).

Hal pertama kali yang perlu dilakukan dalam membangun desain basis data fisik ialah mengidentifikasi DBMS yang akan digunakan. Selanjutnya, baru desain fisik bisa disesuaikan dengan DBMS tersebut. Desain logis dan fisik memiliki keterkaitan yang sangat erat, bahkan bisa saling menyesuaikan. Hal ini terjadi karena keputusan akhir dibuat pada saat tahap desain fisik guna meningkatkan performa suatu system, yang tidak jarang akan berefek pada penyesuaian struktur pada model data logis.

Langkah-langkah yang perlu di lakukan dalam tahap desain fisik ini adalah sebagai berikut:

1. Menterjemahkan model data logis pada DBMS, yaitu: mendesain relasi, representasi data, hingga aturan umum pada DBMS.
2. Mendesain index dan organisasi file, yang meliputi: analisa transaksi, memilih organisasi file, memilih pengindeksan, dan mengestimasi media penyimpanan yang dibutuhkan.
3. Mendesain *user view*.
4. Mendesain mekanisme pengamanan basis data.
5. Mempertimbangkan adanya redundansi yang terkontrol dengan melonggarkan aturan-aturan normalisasi, yang terkadang malah bisa meningkatkan performa sistem.
6. Memantau dan menjaga sistem yang beroperasi agar tetap berjalan dengan baik, bahkan bila perlu melakukan penyesuaian-penesuaian agar lebih baik lagi.

## **Pemodelan Data**

Suatu DBMS memungkinkan kita untuk menyimpan dan menarik/mengambil data. Data sendiri berisikan suatu fakta yang mana akan berubah menjadi informasi yang berguna dan mudah dipahami setelah diproses lebih lanjut. Informasi tersebutlah yang pada akhirnya akan disampaikan kepada manusia atau dalam hal ini adalah pengguna (*user*).

Saat mendesain suatu basis data, desainer harus terlebih dahulu memutuskan data apa saja yang akan disimpan di database. Struktur dan relasi antar data harus ditentukan. Oleh karena itu, seorang desainer membutuhkan metode untuk membuat suatu basis data yang tepat untuk kebutuhan pengguna. Metode inilah yang disebut sebagai Model atau Model Data.

Model sendiri merupakan abstraksi dari suatu proses yang mana menyembunyikan detail-detail tertentu yang tidak dibutuhkan, dan sebaliknya

menekankan pada hal-hal lain yang dibutuhkan dalam membangun sistem. Model data merupakan kumpulan dari tool-tool konseptual untuk mendeskripsikan data, hubungan antar data, semantik hingga *constraint* atau aturan yang berlaku. Di sini, model data juga menjelaskan tentang metode penyimpanan dan pengambilan data dari dan ke basis data.

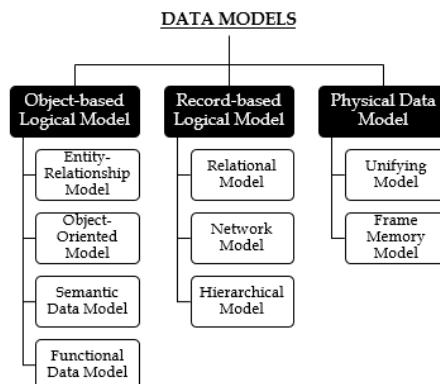
Pemodelan data berguna untuk membantu kita dalam memahami maksud dari suatu data. Data model menunjukkan apa yang dipahami oleh seorang desainer tentang kebutuhan informasi pada suatu organisasi. Dan secara umum, data model menyediakan abstraksi dari suatu aplikasi basis data.

Model data terdiri dari:

1. Unit logis (*logical unit*), seperti: baris dan item data (*data item*).
2. Relasi antara unit-unit logis.
3. Item data merupakan unit logis dari data, sedangkan baris merupakan kumpulan dari item data.

Mengingat model data digunakan untuk merepresentasikan entitas dan relasi-relasinya di dalam suatu basis data, maka model data dapat dibagi menjadi 3 jenis, yaitu:

1. Model logis berbasis objek (*object based logical models*)
2. Model logis berbasi baris (*record based logical models*)
3. Model data fisik (*physical data models*)



**Gambar 18: Jenis-jenis model data**

## Model E-R

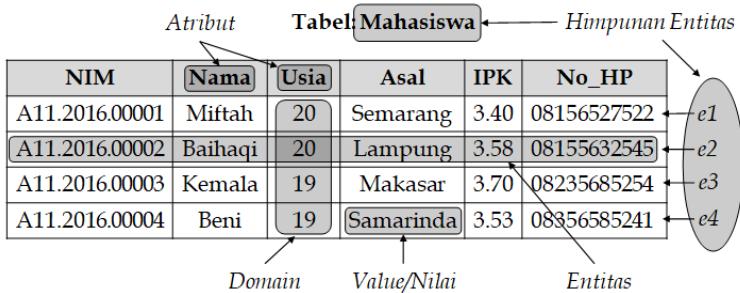
Entity-Relationship Model umumnya diekspresikan dengan menggunakan E-R Diagram (dibahas di Bab 4). Model Entity Relational (E-R) pertama kali diperkenalkan pada 1976, oleh Peter Pin-Shan Chen (陈品山). E-R Model merupakan *high-level conceptual data model* yang paling populer sekarang ini. Model ini merupakan cara yang sistematis dalam mendeskripsikan dan mendefinisikan suatu *conceptual database design process*. Di sini, E-R Model memandang dunia nyata (real world) sebagai kumpulan dari objek-objek dasar (dikenal dengan entitas) beserta karakteristiknya (atribut), juga hubungan antar objeknya (relationship/relasi). Entitas, atribut, dan relasi merupakan 3 komponen utama dalam E-R Model

Setidaknya ada empat kelebihan dari E-R Model ini, yaitu:

- E-R Model mampu menggambarkan kebutuhan-kebutuhan data pada dunia nyata dengan cara yang sederhana, logis, dan informatif.
- Entitas dan atribut pada E-R Model dapat dengan mudah dikonversi menjadi Tabel dan kolom (atau field) dalam Relational Model.
- Sebagai Design Plan, E-R Model dapat diimplementasikan di DBMS apapun.
- Sederhana dan mudah untuk dipahami, menjadikan E-R Model sangat efektif digunakan untuk berkomunikasi antara desainer dan *end user*, terkait rancangan yang hendak dibuat.

Ada beberapa istilah yang menjadi dasar dari E-R Model:

- **Enterprise:** merujuk pada suatu organisasi, seperti sekolah, universitas, bank, dll.
- **Entity:** merujuk pada objek pada dunia nyata, seperti mahasiswa, siswa, karyawan bank, dll.
- **Attributes:** merupakan karakteristik dari suatu entitas, contoh: entitas mahasiswa memiliki atribut seperti NIM, nama, alamat, tgl lahir, dll.
- **Value:** merupakan informasi atau data yang disimpan pada setiap atribut masing-masing data.
- **Entity Sets:** entitas-entitas yang memiliki atribut yang sama menghasilkan himpunan entitas.
- **Domain (Value Set):** merupakan himpunan dari semua nilai atau informasi tentang suatu atribut.



Gambar 19: Komponen-komponen dalam E-R Model

## Model Relasional

Sebelum model relasional dikembangkan, kebanyakan dari software basis data menggunakan struktur pohon (*tree*) dalam menyimpan data ke database. Sebenarnya metode ini sangatlah kompleks, khususnya dalam mengelola relasi antara data atau database. Akibat dari kerumitan-kerumitan tersebut, maka diusulkanlah metode yang lebih sederhana sekaligus elegan, yang dinamakan dengan Model Data Relasional atau *Relational Data Model (RDM)*.

RDM menyimpan data dalam bentuk tabel. Dan salah satu fitur penting yang dibawa oleh RDM adalah sebuah database tunggal yang dapat dipecah ke dalam beberapa tabel. Dalam hal ini, kumpulan dari tabel-tabel tersebut merepresentasikan data dan juga relasinya. Masing-masing tabel terdiri dari beberapa kolom, dan masing-masing kolom ditandai dengan nama-nama yang unik. Keterhubungan (*relationship*) antara dua tabel tersebut ditunjukkan oleh tabel ketiga, dimana salah satu kolom/field dari kedua tabel sama-sama diambil dan dijadikan kolom/field pada tabel ketiga tersebut. Dengan demikian, sebuah tabel bernama Customer-Amt telah berhasil terbentuk.

Beberapa kelebihan Model Data Relasional, diantaranya adalah:

1. Database RDM dapat digunakan pada komputer dengan memori dan kemampuan pemrosesan yang terbatas.
2. RDM sangat mudah untuk digunakan.
3. RDM memungkinkan sistem komputer untuk menangani beragam permintaan dengan cara yang efisien.
4. RDM sangat berguna untuk basis data skala kecil.
5. RDM mampu meningkatkan performa sistem karena ia fokus pada data, bukan pada struktur.

Disamping beberapa kelebihan di atas, RDM memiliki beberapa kelemahan, diantaranya yaitu:

1. RDM menyembunyikan detail informasi tentang penyimpanan data fisik kepada pengguna, dan ia juga menyembunyikan kompleksitas implementasi.
2. Karena tidak diketahui bagaimana suatu data itu disimpan, maka hal ini dapat membuat seorang desainer basis data terlena, hingga pada akhirnya ia menghasilkan desain yang buruk.
3. Mengingat kemudahan dalam penggunaan dan implementasi, banyak orang atau departemen dengan penuh percaya diri membuat database sendiri.. Hal-hal seperti itu dapat memunculkan ketidak konsistensi data (*data inconsistency*), hingga duplikasi dan redudansi data. Apabila ini tidak dikendalikan, makabisa mengacaukan keseluruhan sistem yang ada.

## Bab 4

# DIAGRAM E-R

---

### Bab ini membahas:

- Kelebihan dan kekurangan ERD
  - Simbolisasi pada ERD
  - Entitas, Himpunan Entitas, Atribut, Relasi dan Kardinalitas, Himpunan Relasi, *Relation Keys*
  - *Enhanced ERD*
  - *Participation Constraint*
- 

Struktur logis dari suatu basis data dapat diekspresikan secara grafis dengan menggunakan bantuan diagram, salah satunya yang dikenal dengan nama Diagram E-R (*Entity-Relationship Diagram*). *Entity-Relationship Diagram* atau yang sering disingkat dengan ERD merupakan representasi visual dari data yang menjelaskan bagaimana data-data saling terhubung satu sama lain. ERD pertama kali dikembangkan pada tahun 1970 oleh Dr. Peter Chen bersama timnya. ERD dianalogikan sebagai fotokopi dari struktur data. Diagram ini mampu menyederhanakan konsep penyimpanan data yang besar dan kompleks. Diagram E-R dibangun berbasiskan persepsi dari kondisi nyata (*real-world*) yang terdiri dari kumpulan objek-objek dasar yang bernama entitas dan relasi antar objek-objek tersebut. ERD memiliki beberapa manfaat dan tujuan, diantaranya yaitu:

1. Digunakan untuk mengkomunikasikan struktur logis suatu basis data kepada *end users* atau pengguna akhir.
2. Membantu desainer basis data dalam memahami informasi-informasi apa saja yang perlu disimpan di suatu basis data.
3. Sebagai salah satu *tool* dokumentasi basis data.

## Kelebihan dan Kekurangan ERD

Kelebihan dari diagram E-R, diantaranya adalah:

1. Mudah untuk dipahami dan sederhana dalam pembuatannya.
2. Menyediakan representasi visual/grafis dari desain basis data.
3. Dengan menggunakan ERD, redundansi data bisa lebih mudah untuk dihindari.

Selain beberapa kelebihan yang dijelaskan di atas, ERD memiliki beberapa kekurangan atau kelemahan, diantaranya yaitu:

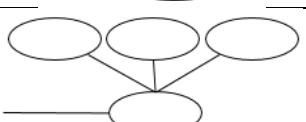
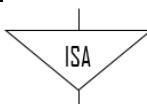
1. Tidak memiliki standar universal.
2. Terbatasnya representasi *constraint* maupun relasi.
3. Tidak memiliki representasi pada manipulasi data.

## **Simbolisasi pada ERD**

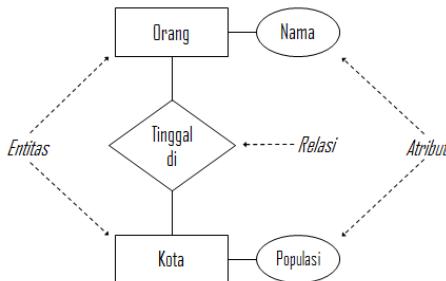
Simbol-simbol yang digunakan dalam E-R Diagram, diantaranya tampak pada *Tabel 6*.

**Tabel 6: Simbol-simbol pada ERD**

No.	Nama Simbol	Simbol	Keterangan
1.	Persegi panjang		Entitas/himpunan entitas – kuat ( <i>strong</i> )
2.	Persegi panjang ganda		Entitas/himpunan entitas – lemah ( <i>weak</i> )
3.	Elips		Attribut ( <i>attribute</i> )
4.	Diamond/belah ketupat		Relasi atau himpunan relasi
5.	Diamond ganda		Mengidentifikasi tipe relasi
6.	Elips ganda		Atribut multi-nilai ( <i>multi-valued attributes</i> )

7.	Elips garis putus-putus		Atribut turunan (derived attributes)
8.	Elips dengan garis di dalam		Atribut kunci (key attributes)
9.	Elips yang terhubung dengan elips lain		Atribut komposit (composite attributes)
10.	Garis ganda		Partisipasi penuh (total participation)
11.	Garis tunggal		Partisipasi sebagian (partial participation)
12.	Segitiga		Spesialisasi atau generalisasi

Contoh penggunaan simbol-simbol pada Tabel 6, adalah sebagai berikut:



**Gambar 20: Contoh penggunaan simbol-simbol ERD**

Gambar 20 menunjukkan bahwa diagram tersebut memiliki dua buah entitas, yaitu ‘Orang’ dan ‘Kota’. Dua entitas itu dihubungkan oleh sebuah relasi yang bernama ‘Tinggal di’. Cara membacanya adalah, setiap satu orang tinggal di hanya satu kota, namun satu kota bisa ditinggali oleh banyak orang. Kata ‘satu’ dan ‘banyak’ ditentukan oleh symbol khusus yang akan dibahas pada bagian kardinalitas.

## Elemen-Elemen pada ERD

Dalam subbab ini, akan dibahas elemen-elemen yang ada pada Diagram E-R, diantaranya yaitu:

1. Entitas (*entities*)
2. Himpunan entitas (*entity sets*)
3. Atribut (*attributes*)
4. Relasi (*relationships*)
5. Himpunan relasi (*relationship sets*)

### Entitas

Entitas (*entity*) merupakan sebuah objek nyata (*real world object*) yang memiliki ciri-ciri khusus yang disebut dengan *properties*. Suatu objek harus bisa dibedakan dari objek-objek yang lain, sebagai contoh: mahasiswa, karyawan, dokter, dan politisi. Masing-masing entitas tersebut tentu bisa dibedakan dari entitas-entitas yang lain, karena memiliki ciri-ciri yang berbeda.

Pada diagram E-R, entitas digambarkan dengan symbol kotak, sebagai contoh:

Mahasiswa

→ *Ini merupakan entitas dengan nama 'Mahasiswa'*

**Gambar 21: Ilustrasi simbolisasi entitas pada diagram E-R.**

Kumpulan ciri atau *properties* pada suatu entitas disebut dengan ‘atribut’ (*attributes*). Sebuah entitas wajib memiliki beberapa atribut yang mana dengan atribut-atribut ini, suatu entitas menjadi bersifat unik sehingga mudah dikenali atau dibedakan dari entitas-entitas lainnya. Sebagai contoh, NIM merupakan identitas dari mahasiswa, sedangkan NIK merupakan identitas dari karyawan. NIM dan NIK merupakan salah satu contoh atribut dari entitas mahasiswa dan karyawan.

Apabila kita berbicara tentang ‘mahasiswa’, maka yang dinamakan sebagai entitas adalah orang-orang di dalamnya, seperti Miftah, Baihaqi, Kemala dan Beni. Keempat mahasiswa tersebut tentu memiliki NIM dan atribut-atribut lain yang terkait.

### Himpunan Entitas

Entitas merupakan objek dasar dari model E-R. Entitas merupakan sebuah objek di dunia nyata (*real world*) yang memiliki ciri-ciri (*properties/attributes*)

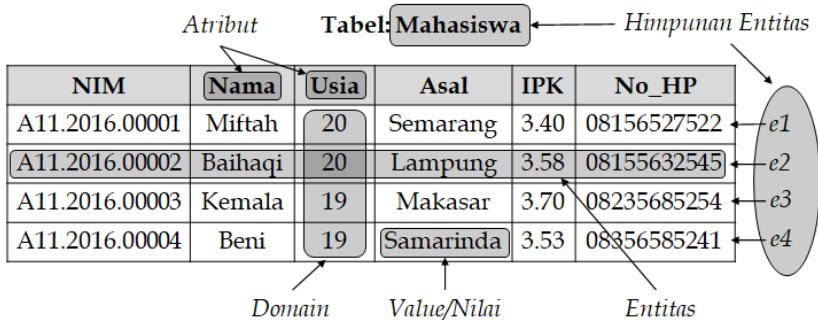
sehingga bisa dibedakan dari entitas-entitas lain. Kumpulan entitas-entitas yang sejenis atau memiliki atribut yang sama disebut sebagai himpunan entitas (*Entity Set*). Sebagai contoh:

1. Sekumpulan orang yang bekerja di perusahaan, disebut sebagai himpunan entitas ‘karyawan’.
2. Sekumpulan orang yang belajar di universitas, disebut sebagai himpunan entitas ‘mahasiswa’.
3. Jika sebuah bank memiliki beberapa cabang, maka ini juga bisa disebut sebagai *entity set*. Sehingga pada himpunan entitas bernama ‘cabang’, akan memiliki beberapa entitas yang dibedakan dari atribut-atributnya, seperti: nama\_cabang, kota\_cabang, dan aset.

Himpunan entitas seringkali disebut sebagai ‘entitas’ saja. Apabila ‘Mahasiswa’ merupakan himpunan entitas, maka Miftah, Baihaqi, Kemala dan Beni adalah entitasnya. Dalam diagram E-R, baik entitas maupun himpunan entitas disimbolkan dengan bentuk kotak (lihat *Gambar 22*).

## Atribut

Atribut merupakan suatu deskripsi yang digunakan untuk mengidentifikasi, mengklasifikasi atau bahkan menjelaskan terbentuknya suatu entitas atau relasi-relasinya. Pada setiap atribut, kumpulan dari nilai-nilainya (*values*) disebut dengan **Domain**. Dan suatu atribut yang membentuk suatu entitas dengan nilai-nilai yang unik disebut sebagai **Primary Key**. Jika pada suatu tabel tidak ada atribut yang bertindak sebagai primary key (dengan nilai-nilai yang unik), maka satu buah atribut baru ditambahkan dengan tugas menjadi Primary Key. NIM (Nomor Induk Mahasiswa) dan NIK (Nomor Induk Karyawan) merupakan Primary Key dari masing-masing tabel Mahasiswa maupun Karyawan, karena keduanya bersifat unik, alias tidak ada mahasiswa atau karyawan yang memiliki nomor identitas yang sama.



Gambar 22: Komponen-komponen suatu tabel

Pada contoh tabel Mahasiswa, tampak keberadaan NIM yang memiliki nilai unik dan bisa menjadi identitas dari setiap entitas. Sekarang perhatikan tabel Penyakit di bawah ini (*Tabel 23*), yang mana tidak memiliki identitas unik untuk masing-masing entitasnya. Pada tabel (a) tidak terdapat Primary Key, karena setiap atributnya memungkinkan nilai yang sama (tidak unik). Oleh karena itu, perlu menambahkan satu buah atribut lagi yang berperan sebagai pemberi identitas unik untuk masing-masing entitas, dan bisa diberi nama ‘ID\_Penyakit’ (tabel b).

Nama_Penyakit	Organisme
Typhoid	Salmonella Typhi
Dysentry	Shigella Dysenteriae

(a)

ID_Penyakit	Nama_Penyakit	Organisme
0001	Typhoid	Salmonella Typhi
0002	Dysentry	Shigella Dysenteriae

(b)

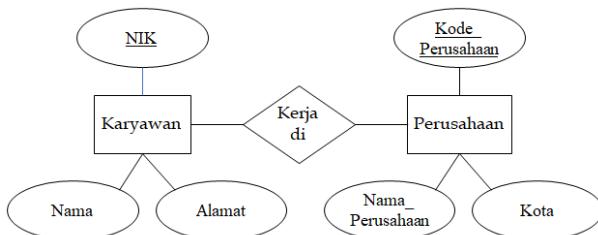
Gambar 23: (a) tabel penyakit tanpa primary key; (b) tabel penyakit dengan primary key

Secara aturan, atribut hanya boleh muncul dalam 1 tempat saja di suatu model, baik ketika atribut tersebut masuk dalam kategori dibutuhkan (*required*) ataupun opsional. Jika suatu atribut tergolong sebagai atribut yang dibutuhkan, maka ia wajib memiliki nilai (tidak boleh kosong). Namun jika ia tergolong sebagai atribut yang opsional, maka ia boleh bernilai kosong atau tidak memiliki nilai. Sebagai contoh, entitas ‘Tanaman’ memiliki 4 atribut, yaitu:

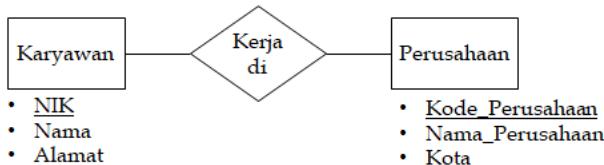
nama\_tanaman, jenis\_tanaman, tanggal\_akuisisi, dan ukuran\_pot. Dalam kasus ini, atribut ‘ukuran\_pot’ tergolong sebagai atribut opsional, karena ada beberapa tanaman yang datang tidak dengan pot, melainkan dengan plastik biasa. Sedangkan atribut-atribut lain berjenis *required*.

Dalam diagram E-R, suatu atribut dapat direpresentasikan melalui 2 cara, yaitu:

1. Bentuk elips yang menempel di entitas (*Gambar 24*)
2. Daftar atribut sebagai teks biasa (*Gambar 25*)



**Gambar 24: Contoh diagram E-R dengan atribut dalam bentuk elips**



**Gambar 25: Contoh diagram E-R dengan atribut berbentuk list**

Garis bawah yang ada pada NIK dan Kode\_Perusahaan menunjukkan bahwa atribut tersebut merupakan Primary Key.

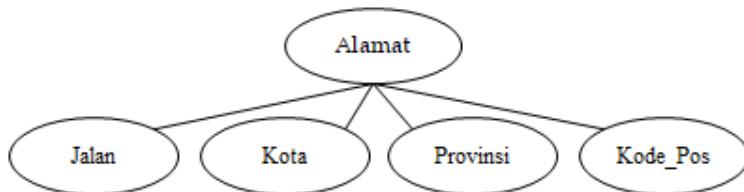
Ada 6 jenis atribut yang bisa digunakan sesuai dengan kebutuhannya, yaitu:

1. **Simple attributes atau atribut sederhana.** Merupakan atribut yang mewakili satu ciri saja dan tidak dapat dipecah menjadi beberapa. Atribut jenis ini juga biasa disebut juga dengan atribut atomik (*atomic attributes*). Contoh:



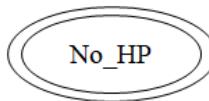
**Gambar 26: Contoh atribut sederhana atau atomik**

2. **Composite attributes** atau **atribut komposit**. Merupakan atribut yang dapat dipecah menjadi beberapa sub-atribut (atribut sederhana). Atribut komposit membantu kita untuk mengelompokkan beberapa atribut terkait. Penggambarannya menggunakan konsep hirarki. Sebagai contoh, ‘Alamat’ merupakan atribut komposit karena bisa dipecah menjadi beberapa sub-atribut, yaitu: jalan, kota, provinsi, dan kode\_pos.



Gambar 27: Contoh atribut komposit

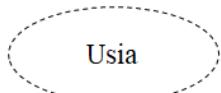
3. **Single-Valued Attribute**. Atribut jenis ini hanya mengizinkan satu nilai saja di kolomnya. Satu entitas tidak diperkenankan memasukkan lebih dari satu nilai pada atribut ini. Contoh: NIM, nama, usia, hanya memiliki satu nilai saja.
4. **Multi-Valued Attribute**. Berkebalikan dari jenis atribut sebelumnya, pada atribut ini diperkenankan mengisikan lebih dari satu nilai untuk entitas tertentu. Sebagai contoh: kolom alamat\_email, nomor\_HP, dan tanggungan memungkinkan untuk diisi lebih dari satu nilai. Berikut ini adalah contoh simbol yang merepresentasikan *Multi-Valued Attribute*, yaitu menggunakan garis ganda.



Gambar 28: Contoh simbolisasi Multi-Valued Attribute

5. **Null Attribute**. Atribut jenis ini tidak mengizinkan adanya nilai kosong atau tidak di isi (*Null*). Nilai null (*null value*) digunakan ketika suatu entitas tidak memiliki nilai untuk suatu atribut. Ada dua asumsi yang dilakukan untuk atribut jenis ini, yaitu: “*not applicable*” dan “*value is missing*”. Jika seorang karyawan tidak memiliki tanggungan atau tanggungan bernilai null, ini bermakna “*not applicable*”. Namun jika NIK seorang karyawan tidak ada, atau tidak mengisi nama, artinya “*value is missing*”.

6. ***Derived Attribute.*** Nilai dari atribut ini dapat diperoleh dari atribut atau entitas lain. Sebagai contoh, atribut ‘usia’ bisa diperoleh dari menghitung tahun sekarang dikurangi tahun lahir yang ada di atribut ‘tanggal\_lahir’ (jika ada). Jika demikian, maka atribut ‘usia’ disebut sebagai *derived attribute*, dan atribut ‘tanggal\_lahir’ disebut sebagai *stored attribute*. Penyimbolannya *derived attribute* dapat digambarkan dengan menggunakan elips bergaris putus-putus.

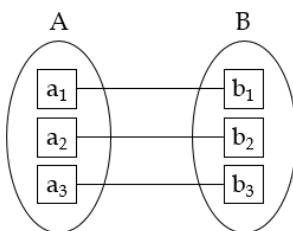


Gambar 29: Contoh derived attribute

## Relasi dan Kardinalitas

Dalam dunia basis data, relasi merujuk pada hubungan antara data, entitas, dan bahkan tabel. Sebagai contoh, misal hubungan antara karyawan dan departemen, atau hubungan antara mahasiswa dan dosen. Kardinalitas suatu relasi merujuk pada pemetaan entitas-entitas yang terhubung pada suatu relasi. Pemetaan sendiri merupakan proses pemilihan model logis dari model konseptual untuk kemudian diubah ke dalam database fisik. Ada 4 jenis pemetaan kardinalitas yang seringkali digunakan, yaitu:

1. **One-to-one (1 : 1).** Setiap entitas di A dihubungkan dengan paling banyak satu entitas di B, dan setiap entitas di B dihubungkan dengan paling banyak satu entitas di A (*Gambar 30*). Sebagai contoh, 1 departemen hanya memiliki 1 manajer atau 1 universitas hanya memiliki 1 rektor.

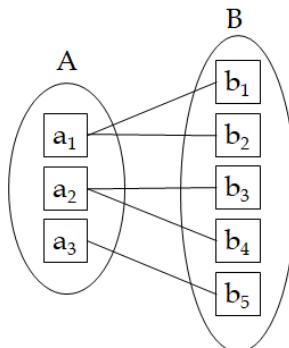


Gambar 30: Ilustrasi one-to-one



Gambar 31: Representasi relasi 1:1 pada ERD

2. **One-to-many (1 : M).** Sebuah entitas di A dihubungkan dengan berapapun entitas di B, dan setiap entitas di B dihubungkan dengan paling banyak satu entitas di A. Hubungan 1:M direpresentasikan dengan konsep “ibu-anak”. Salah satu sisi (sisi yang bernilai 1) disebut sebagai *ibu*, dan sisi yang lain (sisi yang bernilai M) disebut sebagai *anak*. Satu *ibu* bisa punya berapapun *anak*. Namun setiap *anak* tidak bisa memiliki lebih dari satu *ibu*. Sebagai contoh, satu mahasiswa dapat mengambil beberapa mata kuliah. Hal yang sama juga berlaku pada kasus lain seperti, setiap manajer membawahi beberapa karyawan. Adapun representasi relasi one-to-many pada diagram E-R dapat dilihat pada Gambar 33 dan Gambar 34. Anda dapat memilih salah satu cara tersebut dalam merepresentasikan one-to-many relationship.



Gambar 32: Ilustrasi one-to-many

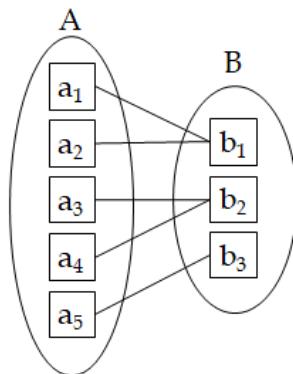


Gambar 33: Representasi relasi 1:M pada ERD



Gambar 34: Alternatif relasi 1:M pada ERD

3. **Many-to-one (M : 1).** Sebuah entitas A dihubungkan dengan paling banyak sebuah entitas di B, dan setiap entitas di B dihubungkan dengan berapapun entitas di A. Sebagai contoh, beberapa politisi hadir di sebuah pesta. Contoh lain, beberapa karyawan bekerja di sebuah departemen. Hal ini berarti bahwa satu departemen dapat memiliki beberapa atau banyak karyawan, namun satu karyawan hanya ada di satu departemen.



Gambar 35: Ilustrasi many-to-one

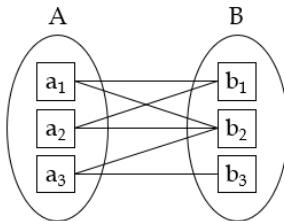


Gambar 36: Representasi relasi M:1 pada ERD



Gambar 37: Alternatif relasi M:1 pada ERD

4. **Many-to-many (M : N).** Entitas di A dapat dihubungkan dengan berapapun entitas di B, demikian juga sebaliknya. Sebagai contoh: beberapa dokter bekerja di beberapa rumah sakit.



Gambar 38: Ilustrasi many-to-many



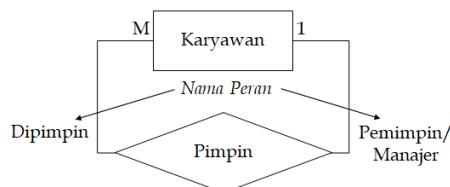
Gambar 39: Representasi relasi M:N pada ERD



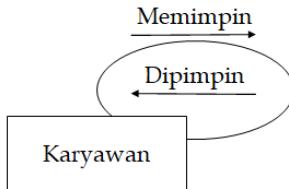
Gambar 40: Alternatif relasi M:N pada ERD

Jumlah entitas yang terlibat dalam satu relasi menunjukkan tingkat atau derajat suatu relasi (*degree of the relationship*). Derajat suatu relasi didefinisikan sebagai jumlah entitas yang diasosiasikan dengan suatu relasi. Secara umum, beberapa jenis *degree of relationship*, diantaranya yaitu: *unary*, *binary* dan *ternary*.

1. **Unary Relationship.** Relasi jenis ini biasa disebut juga dengan *Recursive Relationship*, karena hanya melibatkan satu entitas saja. Dengan kata lain, *unary relationship* adalah relasi berderajat 1. Dalam jenis ini, entitas akan terhubung dengan dirinya sendiri. Perhatikan contoh berikut ini:



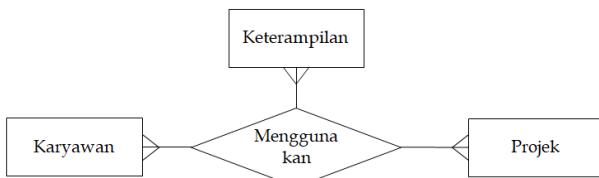
Gambar 41: Representasi unary relationship



**Gambar 42: Contoh lain representasi *unary relationship***

Pada contoh di atas hanya ada satu entitas tapi memainkan dua peran (*role*) yang berbeda. Yang pertama yaitu manajer/pemimpin, dan berikutnya adalah karyawan yang dipimpin. Satu pemimpin bisa memimpin/memiliki banyak karyawan/bawahan, sementara beberapa karyawan/bawahan dipimpin oleh satu manajer saja. Dalam kasus yang bersifat *unary*, pemberian peran akan sangat penting guna menekankan tujuan dari masing-masing entitas dengan peran berbeda itu dalam suatu hubungan atau relationship.

2. **Binary Relationship.** Relasi jenis ini melibatkan dua buah entitas yang saling terhubung, sehingga disebut sebagai relasi berderajat dua. Relasi jenis binary ini merupakan jenis yang paling umum, sebagaimana telah dibahas di bagian sebelumnya (lihat *Gambar 31* hingga *Gambar 40*). Sebuah entitas bisa direlasikan ke entitas lain menggunakan kardinalitas yang ada, seperti: 1 : 1, 1 : M, M : 1, M : N.
3. **Ternary Relationship.** Relasi jenis ini memiliki derajat tiga karena melibatkan tiga entitas, dan digunakan ketika *binary relationship* dianggap tidak cukup akurat dalam menjelaskan makna antar 3 entitas. Perhatikan pada *Gambar 43*. Pada ilustrasi tersebut dapat dibaca bahwa seorang karyawan bisa menggunakan beberapa keterampilan pada beberapa projek. Demikian pula keterampilan, bisa digunakan oleh beberapa karyawan pada beberapa projek.



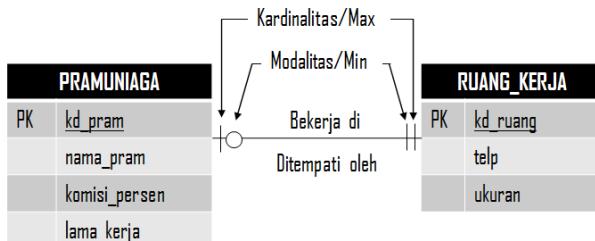
**Gambar 43: Contoh *ternary relationship***

Selanjutnya, untuk relasi yang melibatkan entitas sejumlah  $n$ , maka nama relasinya adalah  $n$ -ary *Relationship*.

## Constraint

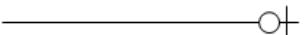
Bahasan tentang *constraint* masih terkait dengan subbab sebelumnya, yaitu Relasi dan Kardinalitas. Konsep *constraint* pada relasi menjelaskan tentang kekangan atau batasan jumlah relasi antara satu entitas dengan entitas lainnya. Batasan yang dimaksud ialah batasan terkait jumlah minimum maupun maksimum dari relasi yang menghubungkan suatu entitas. *Constraint* terdiri dari dua, yaitu kardinalitas dan modalitas.

Cardinalitas (*cardinality*) digunakan untuk menentukan jumlah maksimal dari suatu relasi. Jumlah maksimal tersebut bisa saja berjumlah 1 atau bahkan banyak. Di sisi lain, modalitas (*modality*) ialah jumlah minimal yang boleh digunakan pada suatu relasi antar entitas. Dengan adanya modalitas ini, maka suatu hubungan antar entitas dapat diberi tanda apakah ia boleh memiliki 1 saja atau bahkan nol buah relasi. Gambar 44 menjelaskan perbedaan lokasi peletakan simbol pada kardinalitas dan modalitas, sedangkan macam-macam simbol beserta penjelasannya ada pada tabel 7.



Gambar 44: Ilustrasi keberadaan kardinalitas dan modalitas pada suatu relasi

No.	Simbol	Keterangan
1.	—+—	Satu (kardinalitas)
2.	—<—	Banyak (kardinalitas)
3.	—  —	Modalitas = 1, kardinalitas = 1, artinya hanya akan terbentuk satu buah relasi saja.

4.		Modalitas = 0, kardinalitas = 1. Minimal 0 dan maksimal 1 buah relasi.
5.		Modalitas = 1, kardinalitas = banyak. Minimal 1 dan tidak dibatasi jumlah maksimal relasi.
6.		Modalitas = 0, kardinalitas = banyak. Minimal 0 dan tidak dibatasi jumlah maksimal relasi.

**Tabel 7: Representasi kekangan kardinalitas dan modalitas**

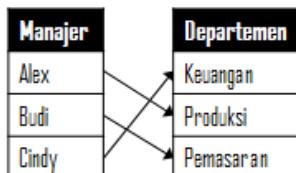
## Himpunan Relasi

Sebagaimana himpunan entitas yang seringkali disebut sebagai entitas saja, himpunan relasi pun demikian. Dalam banyak kasus, himpunan relasi hanya disebut sebagai relasi saja. Sebenarnya, himpunan relasi ialah kumpulan hubungan yang memiliki makna antara beberapa entitas. Sebelumnya telah dibahas relasi antara manajer dan departemen, sebagaimana tampak pada gambar berikut ini:



**Gambar 45: Hubungan antara Manajer dan Departemen.**

Pada diagram di atas, sebenarnya itu adalah himpunan relasi antara himpunan entitas. Apabila dibuat lebih detail untuk menampilkan masing-masing relasi beserta entitasnya, maka hasilnya adalah sebagai berikut:



**Gambar 46: Detail relasi antar entitas**

Pada ilustrasi *Gambar 45*, tampak adanya relasi antara entitas Alex dengan entitas Produksi, yang berarti bahwa Alex adalah manajer yang memimpin di

departemen Produksi. Demikian pula Budi yang memimpin di bagian Pemasaran, dan Cindy yang menjadi manajer di departemen Keuangan. Dengan demikian, himpunan relasi yang bisa digambarkan dari relasi-relasi di atas adalah sebagaimana tampak pada *Gambar 45*. Contoh lain:

ID_Dokter	Nama		No_Pasien	Nama	Alamat
A01	Salma		101	Mutia	Semarang
A02	Sofia		102	Hafiz	Ambarawa
A03	Rania		103	Dian	Madiun

**Gambar 47:** Detail relas antara dokter dan pasien

Apabila digambarkan himpunan relasi sekaligus himpunan entitasnya dalam diagram E-R, maka hasilnya akan seperti berikut ini:



**Gambar 48:** Tampak himpunan relasi antara himpunan entitas Dokter dan Pasien

## Relation Keys

Suatu basis data dikatakan memiliki integritas apabila data-data di dalamnya mengikuti aturan-aturan tertentu. Masing-masing aturan tersebut dinamakan *integrity constraints*. Dalam basis data, *constraint* merupakan suatu batasan, baik pada konten maupun pada operasi basis data. Oleh karena itu, entitas, himpunan entitas, relasi, maupun atribut juga disebut sebagai *constraints*. Selain yang disebut sebelumnya, masih ada lagi beberapa *constraints* yang tidak kalah pentingnya dan dibutuhkan dalam diagram E-R, salah satunya yaitu: kunci atau disebut juga dengan *Key Constraints*.

Kunci atau *key* merupakan salah satu komponen fundamental dalam basis data. Sebuah *key* merupakan suatu atribut atau kumpulan atribut yang digunakan untuk mengidentifikasi data dalam himpunan entitas. Atribut-atribut yang digunakan sebagai *key* dikenal dengan sebutan *key attributes* atau atribut kunci, sedangkan atrirubut-atribut selain itu disebut dengan atribut non kunci (*non-key attributes*).

## Jenis-Jenis *Key* di DBMS

Ada beragam jenis *key* yang umum digunakan di DBMS, diantaranya yaitu:

1. **Super Key.** Merupakan satu atau sekumpulan atribut yang digunakan untuk mengidentifikasi entitas-entitas secara unik dari sebuah himpunan entitas. Terkadang pada sebuah himpunan entitas terdapat lebih dari satu *super key*. Sebagai contoh:

**Karyawan**

No_Registrasi	NIK	Nama	Gaji	Kode_Dept
A01	001	Agung	5.000.000	2
B02	002	Ganda	6.000.000	3
C03	003	Ariyadi	5.000.000	2
E04	004	Arga	5.500.000	1
F05	005	Fendi	6.000.000	3

**Departemen**

Kode_Dept	Nama_Dept
1	Penjualan
2	Pemasaran
3	Produksi

**Gambar 49:** Tampak himpunan entitas ‘Karyawan’ dan ‘Departemen’

Pada himpunan entitas bernama ‘Karyawan’ seperti tampak pada Gambar 48, beberapa *super key*-nya diantaranya adalah sebagai berikut:

- a. (No\_Registrasi, NIK, Nama, Gaji)
- b. (No\_Registrasi, NIK, Nama)
- c. (No\_Registrasi, NIK)

Masing-masing kombinasi di atas dapat mengidentifikasi data secara unik.

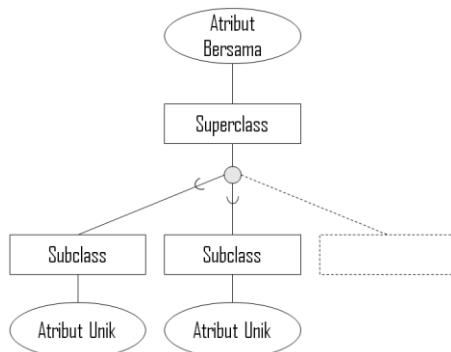
2. **Candidate Key.** *Super key* yang paling sedikit atau minimum disebut sebagai *Candidate Key*. Sebagai contoh, pada himpunan entitas ‘Karyawan’ seperti tampak pada Gambar 48, *candidate key*-nya adalah *NIK* dan *No\_Registrasi*.

3. **Composite Key.** Suatu *key* yang terdiri dari dua atau lebih kolom/atribut disebut dengan *composite key*.
4. **Primary Key.** Suatu kolom/atribut yang dapat digunakan untuk mengidentifikasi data secara unik dikenal dengan nama *primary key*. Sebuah himpunan entitas boleh memiliki lebih dari satu *candidate key*, namun hanya boleh memiliki satu *primary key*. Sebagai contoh, pada himpunan entitas ‘Karyawan’ di atas, *primary key*-nya hanya boleh salah satu saja, yaitu antara *No\_Registrasi* atau *NIK*.
5. **Foreign Key.** *Foreign key* merupakan sebuah atribut pada himpunan entitas manapun yang mana juga menjadi *primary key* pada himpunan entitas lain. Sebagai contoh, atribut *Kode\_Dept* merupakan atribut biasa di dalam himpunan entitas ‘Karyawan’, namun ia merupakan *primary key* pada himpunan entitas ‘Departemen’. Oleh karena itu, *Kode\_Dept* merupakan *foreign key* di himpunan entitas ‘Karyawan’.
6. **Alternate Key.** Semua *candidate key* selain yang menjadi *primary key* pada suatu himpunan entitas dinamakan sebagai *alternate key*.
7. **Secondary Key.** Sebuah atribut atau sekumpulan atribut yang tidak dapat digunakan untuk mengidentifikasi data secara unik, namun dapat digunakan untuk mengidentifikasi suatu kumpulan data, disebut dengan *secondary key*. Sebagai contoh, *nama* dan *gaji* dapat dianggap sebagai *secondary key*. Tujuan dari *secondary key* ialah untuk kepentingan pengambilan data saja.

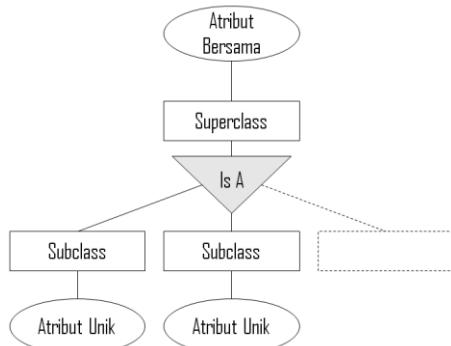
## Enhanced Entity-Relationship Model

Dengan memanfaatkan fitur-fitur dasar dalam Model E-R yang telah dibahas sebelumnya, sebenarnya sudah dapat digunakan untuk memodelkan hampir keseluruhan basis data secara baik. Namun, konsep dasar pemodelan E-R tersebut masih memiliki keterbatasan, khususnya dalam merepresentasikan kebutuhan dari aplikasi-aplikasi yang lebih baru dan lebih kompleks. Oleh karena itu, kebutuhan akan konsep pemodelan semantik yang lebih *powerful* mulai dirasakan oleh para desainer basis data dalam mengembangkan sistem yang semakin hari semakin kompleks. Di sini, E-R Model juga tidak selesai sampai di titik itu saja, sehingga muncul istilah baru yang dikenal dengan nama *Enhanced Entity-Relationship (EER)*. Model EER ini dikembangkan untuk mampu mengakomodir dukungan terhadap beberapa konsep tambahan, seperti: spesialisasi, generalisasi dan agregasi.

Hal baru dan penting yang ditawarkan oleh model EER ini ialah konsep *superclass* dan *subclass* pada entitas. Dalam literatur lain, *superclass* disebut juga sebagai *supertype*, sedangkan *subclass* disebut dengan *subtype*. Perhatikan ilustrasi pada Gambar 49. Penggambaran *superclass* dan *subclass* tidak hanya terpaku pada satu cara saja. Gambar 50 menunjukkan alternatif lain dalam mengilustrasikan *superclass* dan *subclass*.



Gambar 50: Notasi dasar relasi *superclass/subclass*

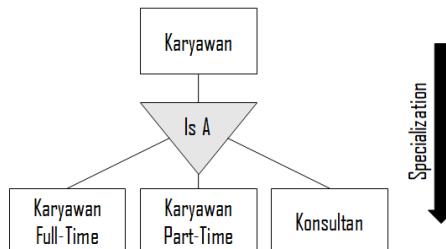


Gambar 51: Alternatif penggambaran *superclass/subclass*

## Spesialisasi (↓)

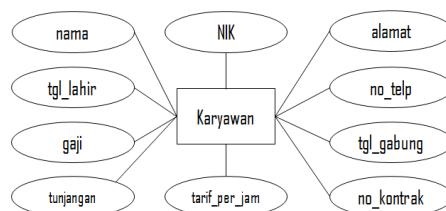
Spesialisasi (*specialization*) ditandai dengan simbol panah ke bawah. Konsep ini menggambarkan suatu proses pembentukan karakteristik baru dari suatu objek, yaitu berupa satu atau lebih kelas baru. Dengan kata lain, spesialisasi merupakan hasil pembentukan *subset* dari himpunan entitas dengan level yang lebih tinggi menjadi himpunan entitas dengan level yang lebih rendah. Definisi

lain tentang spesialisasi adalah proses identifikasi subset dari suatu himpunan entitas (*superclass* atau *supertype*) dengan melakukan pemecahan berdasarkan karakteristik yang membedakannya. Diagram pada *Gambar 51* menjelaskan bahwa himpunan entitas ‘Karyawan’ diperluas dengan spesialisasi sehingga memiliki tiga buah *subset* yaitu ‘Karyawan Full-Time’, ‘Karyawan Part-Time’ dan ‘Konsultan’. Adapun teks ‘Is A’ di dalam simbol segitiga merupakan istilah dalam Bahasa Inggris yang apabila diterjemahkan ke dalam Bahasa Indonesia memiliki arti ‘adalah’.

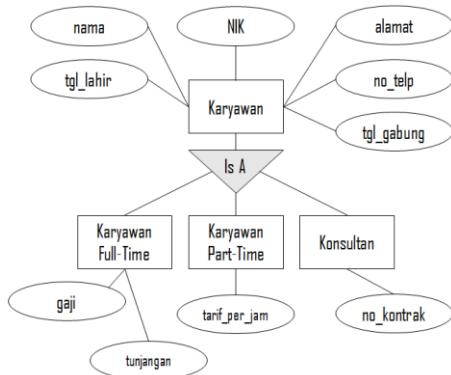


**Gambar 52: Spesialisasi**

Contoh yang lebih lengkap, bisa dilihat pada *Gambar 52* dan *Gambar 53*. Pada *Gambar 52* terlihat himpunan entitas ‘Karyawan’ yang mana memiliki 10 atribut. Himpunan entitas tersebut butuh dispesialisasi mengingat tidak semua atribut berlaku untuk semua karyawan. Misal, perlu dibedakan antara karyawan paruh waktu (*part-time*) dengan karyawan *full-time* terkait dengan cara penggajian, apakah dengan gaji dan tunjangan, ataukah berdasarkan tarif per jam. Karyawan *part-time* digaji dengan menggunakan tarif-per-jam, sedangkan karyawan *full-time* menggunakan gaji dan tunjangan bulanan.

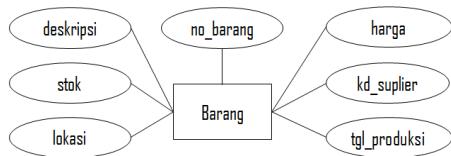


**Gambar 53: Himpunan entitas ‘Karyawan’ sebelum spesialisasi**

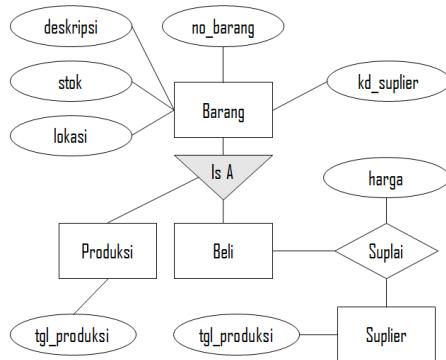


**Gambar 54: Himpunan entitas ‘Karyawan’ setelah spesialisasi**

Contoh yang lain tampak pada *Gambar 54* yang menjelaskan tentang informasi suatu barang. Pada awalnya, himpunan entitas bernama ‘Barang’ memiliki tujuh atribut. Setelah melalui proses spesialisasi (*Gambar 55*), himpunan entitas ‘Barang’ akan memiliki subclass yaitu ‘Produksi’ dan ‘Beli’. ‘Produksi’ merupakan barang yang diproduksi sendiri, sedangkan ‘Beli’ merupakan item barang yang tidak diproduksi sendiri alias beli dari pihak ke tiga. Oleh karena itu, terbentuk himpunan entitas baru lagi bernama ‘Supplier’ yang merupakan pihak penyalur dari barang yang dibeli. Selanjutnya, perhatikan penempatan atribut-atribut yang kini telah disesuaikan dengan himpunan entitas yang seharusnya.



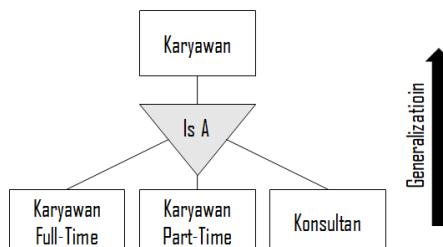
**Gambar 55: Himpunan entitas ‘Barang’ sebelum spesialisasi**



**Gambar 56: Himpunan entitas ‘Batang’ setelah spesialisasi**

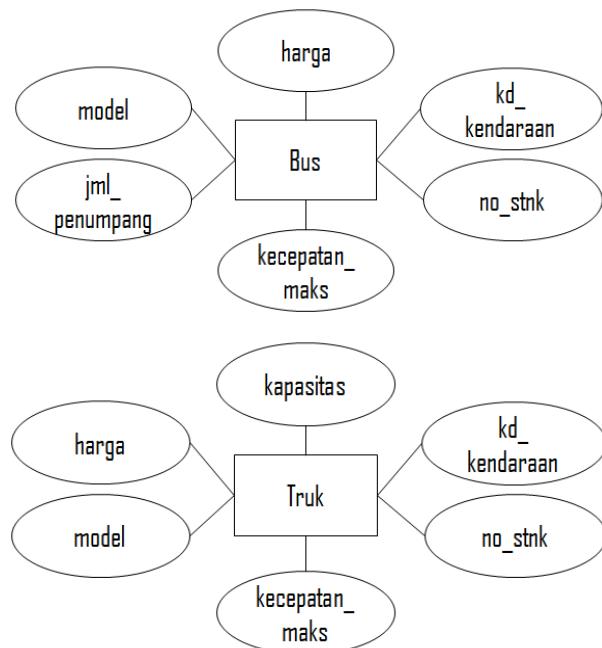
### Generalisasi ( $\uparrow$ )

Generalisasi (*generalization*) merupakan abstraksi dari suatu cara bagaimana melihat beberapa objek berbeda menjadi satu kelas yang lebih umum. Definisi lain dari generalisasi ialah proses mengidentifikasi beberapa karakteristik umum dari beberapa himpunan entitas dan membentuk sebuah himpunan entitas baru yang mengandung beberapa karakteristik umum tersebut. Generalisasi hanya fokus pada karakteristik-karakteristik yang umum saja dari beberapa himpunan entitas, sehingga mengabaikan perbedaan diantara mereka. Dengan kata lain, generalisasi merupakan hasil penggabungan dari dua atau lebih himpunan entitas untuk menghasilkan himpunan entitas baru dengan level yang lebih tinggi. Dalam kasus generalisasi ini, *subclass* akan didefinisikan terlebih dahulu dan diikuti oleh dibentuknya *superclass*. Selanjutnya, himpunan relasi yang melibatkan *superclass* dan *subclass* baru dibuat. Contoh pada *Gambar 56* menjelaskan bahwa kriteria seorang karyawan yaitu harus seorang ‘Karyawan Full-Time’, ‘Karyawan Part-Time’ atau ‘Konsultan’ (bukan yang lain).

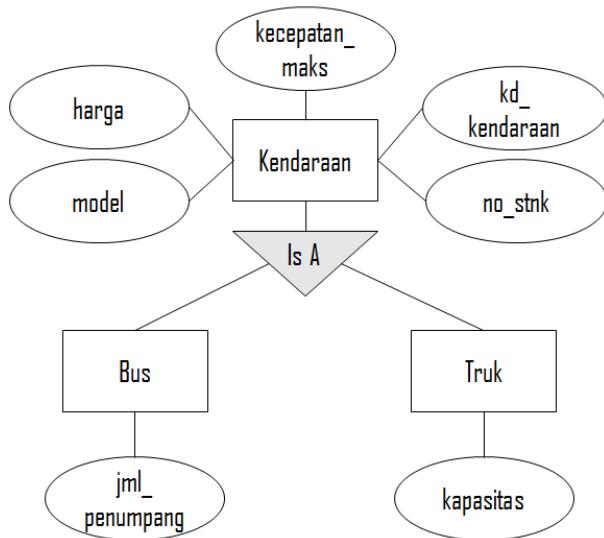


**Gambar 57: Generalisasi**

*Gambar 57* menunjukkan dua himpunan entitas (beserta atribut-atributnya) sebelum melalui tahapan generalisasi. Selanjutnya perhatikan *Gambar 58* dimana kedua himpunan entitas tersebut telah melalui proses generalisasi. Perhatikan, atribut ‘jml\_penumpang’ hanya ada di himpunan entitas ‘Bus’, dan atribut ‘kapasitas’ hanya ada di himpunan entitas ‘Truk’. Atribut-atribut selain itu dimiliki oleh keduanya. Oleh karena itu, kita bisa membentuk satu buah himpunan entitas baru (bernama ‘Kendaraan’) yang memiliki semua atribut selain atribut ‘jml\_penumpang’ dan ‘kapasitas’.



**Gambar 58:** Dua himpunan entitas sebelum dilakukan generalisasi



**Gambar 59: Himpunan entitas ‘Kendaraan’ terbentuk setelah proses generalisasi**

Secara lebih spesifik, perbedaan antara spesialisasi dan generalisasi dapat dilihat pada *Tabel 8*.

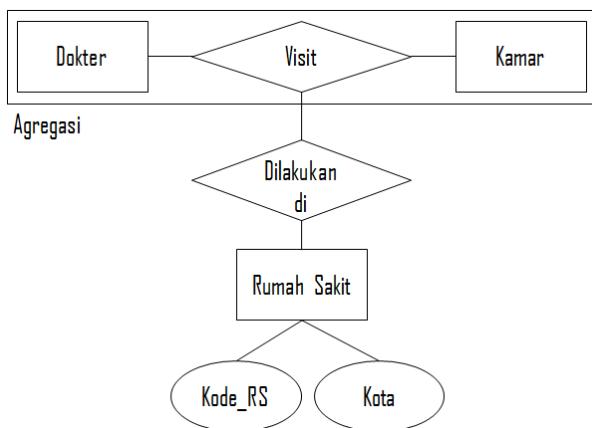
**Tabel 8: Perbedaan antara spesialisasi dan generalisasi**

No.	Spesialisasi	Generalisasi
1.	Proses desain dari atas ke bawah ( <i>top-down</i> ).	Proses desain dari bawah ke atas ( <i>bottom-up</i> ).
2.	Merupakan himpunan bagian ( <i>subset</i> ) dari suatu himpunan entitas, dan membentuk beberapa himpunan entitas baru pada level yang lebih rendah.	Merupakan penggabungan dari dua atau lebih himpunan entitas pada level yang lebih rendah untuk menghasilkan sebuah himpunan entitas yang lebih tinggi (levelnya).
3.	Himpunan entitas yang lebih rendah (levelnya) boleh jadi memiliki atribut yang berbeda dan boleh terlibat dalam suatu	Himpunan entitas pada level yang lebih rendah dideskripsikan oleh atribut maupun relasi pada himpunan entitas di atasnya.

	relasi tanpa melibatkan himpunan entitas di atasnya.	
4.	Digunakan untuk memberikan penekanan pada perbedaan antara himpunan entitas level lebih tinggi dengan yang lebih rendah.	Digunakan untuk menekankan kesamaan diantara himpunan entitas pada level yang lebih rendah.
5.	Mengizinkan adanya banyak himpunan entitas yang terbentuk.	Setiap entitas di level yang lebih tinggi juga harus menjadi entitas di bawahnya.

## Agregasi

Agregasi (*aggregation*) merupakan suatu proses penggabungan informasi pada objek sehingga objek yang lebih tinggi (levelnya) dapat diabstraksikan. Contoh pada *Gambar 59* menggambarkan bahwa dokter mengunjungi kamar pasien, dan segala sesuatunya dilakukan di rumah sakit yang sama.

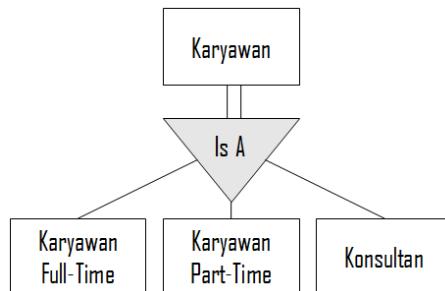


**Gambar 60:** Contoh agregasi terkait dengan kunjungan dokter

## Participation Constraint

Participation constraint bisa juga disebut dengan aturan atau batasan partisipasi, yang mana mengatur tentang partisipasi dari himpunan entitas pada himpunan relasi. Ada dua jenis partisipasi, yaitu:

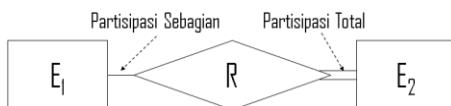
1. Partisipasi Total (*Total Participation* atau *Mandatory Participation*). Apabila semua entitas dari himpunan entitas E berpartisipasi dan minimal memiliki sebuah relasi ke himpunan relasi R, maka partisipasi tersebut disebut dengan partisipasi penuh/total. Partisipasi total direpresentasikan dengan simbol garis ganda, sebagaimana tampak pada *Gambar 60*. Dalam kasus *superclass* dan *subclass*, maka partisipasi total ialah apabila semua member atau entitas di *superclass* berpartisipasi/berperan sebagai member juga pada sebagian *subclass* dalam spesialisasi maupun generalisasi. Sebagai contoh, jika setiap karyawan harus salah satu dari *subclass* ‘karyawan full-time’, ‘karyawan part-time’, atau ‘konsultan’ di sebuah perusahaan dan tidak ada lagi jenis karyawan lain, maka hal demikian disebut sebagai *total participation constraint*.



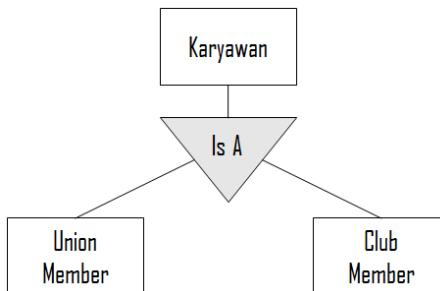
**Gambar 61: Total participation constraint**

2. Partisipasi Sebagian (*Partial Participation* atau *Optional Participation*). Pada partisipasi sebagian ini, tidak semua entitas pada suatu himpunan entitas terlibat dalam suatu relasi. Apabila hanya beberapa entitas dari himpunan entitas E yang berpartisipasi dalam himpunan relasi R, maka disebut sebagai partisipasi sebagian. Partisipasi sebagian ini direpresentasikan dengan symbol garis tunggal (*Gambal 61*). Sebagai contoh, setiap karyawan boleh mengambil pinjaman (artinya tidak semua karyawan wajib/harus mengambil pinjaman) maka disebut sebagai partisipasi sebagian. Dalam kasus

*superclass* dan *subclass*, yang dimaksud dengan partisipasi sebagian apabila anggota dari *superclass* tidak wajib menjadi anggota dari *subclass*-nya dalam spesialisasi maupun generalisasi. Sebagai contoh (*Gambar 62*), seorang karyawan tidak harus menjadi ‘Union Member’ ataupun ‘Club Member’. Karyawan boleh bergabung menjadi salah satu member ataupun keduanya, namun karyawan juga boleh untuk tidak menjadi salah satunya. Hal demikian disebut dengan *partial participation constraint*.



**Gambar 62: Partial participation constraint**



**Gambar 63: Ilustrasi partisipasi total dan sebagian**



**Gambar 64: Partisipasi antara himpunan entitas Fakultas dan Mobil**

Pada contoh *Gambar 63*, terlihat adanya dua buah himpunan entitas yang bernama ‘Fakultas’ dan ‘Mobil’. Keduanya dihubungkan oleh sebuah relasi yang bernama ‘Memiliki’. Di sana terlihat bahwa mobil memiliki partisipasi penuh terhadap relasi ‘Memiliki’. Sedangkan ‘Fakultas’ memiliki partisipasi sebagian terhadap relasi ‘Memiliki’. Hal ini berarti bahwa fakultas tidak wajib memiliki mobil, atau tidak semua fakultas memiliki mobil. Sedangkan entitas mobil wajib ada dalam database agar bisa dipilih/dimiliki oleh fakultas.

# Bab 5

## TRANSFORMASI MODEL DATA KE BASIS DATA FISIK

---

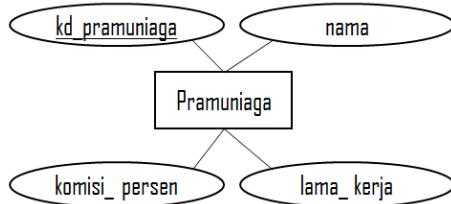
### Bab ini membahas:

- Konversi Pada Entitas Sederhana
  - Konversi Pada Relasi Unary
  - Konversi Pada Relasi Binary
  - Konversi Pada Relasi Ternary
  - Studi Kasus
- 

Transformasi model data menjadi basis data fisik bisa diterjemahkan sebagai konversi ERD menjadi tabel relasional. Konversi Entity-Relationship Diagram (ERD) menjadi tabel relasional tidaklah sulit, cukup dengan beberapa langkah sederhana saja. Pada dasarnya, masing-masing entitas akan dikonversi menjadi suatu tabel, kemudian relasi *many-to-many* atau entitas yang bersifat asosiatif juga akan dikonversi menjadi suatu tabel. Yang perlu diberi perhatian secara khusus yaitu pada saat konversi, dimana beberapa aturan harus diikuti untuk memastikan *foreign keys* muncul secara tepat di suatu tabel. Di dalam bab ini, akan didemonstrasikan konversi dari ERD menjadi tabel relasional.

### Konversi Entitas Sederhana

Gambar 65 merujuk pada satu entitas sederhana (entitas *Pramuniaga*) yang memiliki empat atribut, yaitu: kd\_pramuniaga, nama, komisi\_persen, dan lama\_kerja. Untuk menyederhanakan simbolisasi dalam ERD, berikutnya yang akan kita gunakan ialah model *entity box* seperti tampak pada gambar 66.



Gambar 65: Entitas *Pramuniaga*

PRAMUNIAGA	
PK	kd_pram
	nama_pram
	komisi_persen
	lama_kerja

Gambar 66: Simbolisasi entitas *PRAMUNIAGA* dalam bentuk kotak entitas (*entity box*)

Gambar 67 merupakan tabel relasional yang merupakan hasil konversi dari entitas sederhana bernama *Pramuniaga*. Secara sederhana, tabel yang terbentuk mengandung atribut-atribut yang sebelumnya telah ditentukan pada kotak entitas *Pramuniaga* (Gambar 66). Perhatikan, *kd\_pramuniaga* diberikan garis bawah yang berarti bahwa atribut tersebut memiliki nilai unik dan dijadikan sebagai *Primary Key* pada entitas *Pramuniaga*. Nantinya, permasalahan yang menjadi *issue* utama dan perlu dihadapi adalah kapan suatu entitas dilibatkan atau tidak dalam sebuah relasi dengan entitas lain.

Tabel PRAMUNIAGA

kd_pram	nama_pram	komisi_persen	lama_kerja

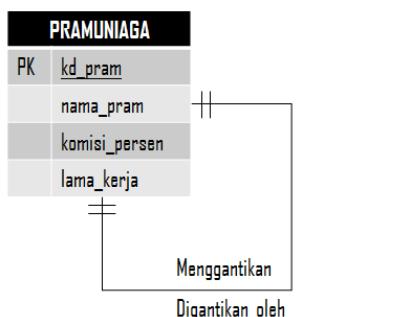
Gambar 67: Bentuk tabel relasional hasil konversi

## Relasi Unary

Dalam relasi tunggal atau *Unary Relationship*, setidaknya ada tiga macam relasi yang akan dibahas, yaitu relasi *One-to-One*, *One-to-Many*, dan *Many-to-Many*. Penjelasan lebih detail adalah sebagai berikut.

## Relasi Unary – *One-to-One*

Relasi tunggal *One-to-One* ialah suatu relasi yang melibatkan hanya satu entitas dengan jenis relasi *one-to-one* atau 1 : 1. Gambar 68 menunjukkan relasi tunggal *one-to-one*. Contoh tersebut menggambarkan kondisi dimana seorang pramuniaga bisa menggantikan atau digantikan oleh rekannya apabila ia sedang tidak bisa menjalankan bertugas. Oleh karena itu, dalam tabel yang terbentuk (Gambar 69) akan muncul sebuah atribut baru bernama *kd\_pengganti* yang merujuk pada kode pramuniaga baru yang menggantikan sang pramuniaga yang absen bertugas.



Gambar 68: Relasi tunggal *one-to-one*

Tabel PRAMUNIAGA

kd_pram	nama_pram	komisi_persen	lama_kerja	kd_pengganti

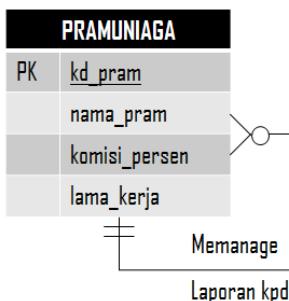
Gambar 69: Tabel relasional hasil konversi dari relasi tunggal *one-to-one*

## Relasi Unary – *One-to-Many*

Relasi tunggal jenis *one-to-many* memiliki situasi yang hampir mirip dengan situasi pada relasi tunggal jenis *one-to-one*, sebagaimana telah dibahas sebelumnya. Pada kasus *one-to-many* ini, ilustrasinya adalah seorang karyawan mengelola karyawan lain, atau menjadi manager dari beberapa karyawan lain. Dengan kata lain, sekelompok karyawan memiliki satu manager dari kelompoknya. Dalam kasus pramuniaga seperti contoh sebelumnya, artinya ada satu pramuniaga yang ditunjuk sebagai manager, yang bertugas untuk manage pramuniaga yang lain. Sebaliknya, para pramuniaga harus memberikan laporan kepada pramuniaga yang bertugas sebagai manager. Relasi seperti ini

disebut dengan relasi tunggal *one-to-many*. Dari Gambar 70 terlihat bahwa seorang pramuniaga bisa mengelola banyak pramuniaga lain atau bahkan tidak sama sekali.

Dalam relasi jenis ini, hasil konversi dari ERD menjadi tabel relasional (Gambar 71) akan membentuk satu atribut baru, yaitu *kd\_manager* yang merepresentasikan kode unik yang dimiliki oleh setiap manager. Isi dari atribut *kd\_manager* merupakan *kd\_pramuniaga* yang telah melekat pada dirinya.



Gambar 70: Contoh ERD dengan relasi tunggal *one-to-many*

Tabel PRAMUNIAGA

kd_pram	nama_pram	komisi_persen	lama_kerja	kd_manager

Gambar 71: Hasil konversi ERD menjadi tabel relasional dengan atribut baru bernama *kd\_manager*

## Relasi Unary – *Many-to-Many*

Selain jenis *one-to-one* dan *one-to-many*, relasi tunggal juga memiliki jenis *many-to-many*. Salah satu contoh klasik yang sering digunakan untuk menjelaskan jenis relasi ini dikenal dengan istilah “*Bill of Materials Problem*” (BOM Problem), dengan produk akhir seperti mobil, pesawat atau permesinan lainnya.

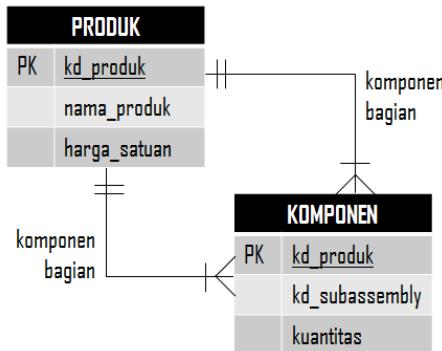
Kita ambil contoh pada produk mobil. Produk-produk tersebut tentu terdiri dari komponen-komponen dasar, seperti mur dan baut (*nuts and bolts*). Selain itu, mur dan baut tersebut juga digunakan untuk menyusun komponen-komponen utama di dalam mobil atau *sub-assemblies* dari objek akhir, misal seperti komponen transmisi. Bahkan apabila di-*break-down* lagi, masing-masing komponen tersebut juga bisa saja terdiri dari beberapa sub-komponen lain yang

juga memiliki mur dan baut. Begitu semua sub-komponen dan komponen tersebut disatukan, barulah bisa dirangkai menjadi sebuah mobil.

Dari ilustrasi di atas, maka komponen dasar dan komponen utama bisa dianggap sebagai bagian (*part*) dari suatu objek atau produk. Oleh karena itu, himpunan entitas komponen mobil (*parts*) tersebut memiliki relasi tunggal jenis *many-to-many* terhadap entitas-entitas lain. Sebuah komponen di mobil bisa tersusun dari beberapa komponen lain dan pada saat yang sama, komponen tersebut menjadi bagian dari komponen yang lain.

Selain menggunakan contoh di atas, relasi tunggal *many-to-many* dapat dicontohkan dengan produk-produk yang dijual di toko bangunan. Peralatan mendasar seperti palu dan kunci inggris dapat digabung dan dijual secara paket dengan produk-produk lain. Paket peralatan (*tool sets*) pertukangan yang lebih lengkap dapat terdiri dari paket-paket peralatan yang lebih kecil ditambah dengan peralatan tambahan seperti palu atau kunci inggris. Produk-produk tersebut, baik yang berupa peralatan tambahan (satuan) ataupun paket, baik yang ukurannya besar ataupun kecil, semuanya dikategorikan sebagai produk. Oleh karena itu, sebuah produk boleh jadi merupakan bagian dari beberapa produk lain, atau mungkin juga tidak sama sekali. Sebaliknya, sebuah produk dapat disusun dari beberapa produk lain, atau tidak sama sekali.

Khusus untuk kasus relasi tunggal *many-to-many*, dibutuhkan dua buah tabel guna konversi dari model data ke basis data fisik. Pada dasarnya, aturan umum dalam konversi model data menjadi basis data fisik adalah bahwa jumlah tabel yang terbentuk akan sama seperti jenis relasi yang ada (unary, binary, ternary) ditambah sebuah tabel khusus untuk yang menggunakan *many-to-many*. Sebagai contoh, relasi binary *many-to-many* akan membentuk dua buah tabel ditambah satu sehingga menjadi tiga tabel. Demikian pula untuk yang unary, akan terbentuk sebuah tabel ditambah satu menjadi dua tabel. Untuk kasus selain *many-to-many* maka jumlah tabel yang dihasilkan tidak perlu ditambah satu.



Gambar 72: Ilustrasi relasi tunggal *many-to-many*

Sebagaimana dilihat pada Gambar 72, tabel PRODUK tidak memiliki *foreign keys*. Tabel KOMPONEN menjelaskan item/komponen apa saja yang digunakan untuk membentuk komponen lain. Atribut *kuantitas* pada tabel KOMPONEN berisi tentang jumlah suatu item yang digunakan untuk membuat komponen lain. Hasil konversinya adalah sebagai berikut:

Tabel PRODUK

kd_produk	nama_produk	harga_satuan

Tabel KOMPONEN

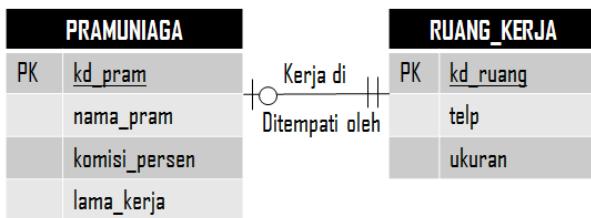
kd_produk	kd_subassembly	kuantitas

Gambar 73: Hasil konversi ERD dengan relasi tunggal *many-to-many* menjadi dua buah tabel relasional

## Relasi Binary

Relasi *Binary* biasa disebut juga dengan sebutan relasi biner atau relasi berderajat dua. Pada relasi ini, jumlah entitas yang terlibat secara langsung berjumlah dua buah saja. Sama halnya dengan relasi tunggal, relasi binary ini

jug memiliki tiga model, yaitu relasi binary *one-to-one*, *one-to-many*, dan *many-to-many*.



Gambar 74: Relasi antara PRAMUNIAGA dan RUANG\_KERJA

### Relasi Binary – *One-to-One*

Pada pembahasa relasi biner *one-to-one* ini, kita akan menggunakan contoh kasus terkait pramuniaga sebagaimana tampak pada Gambar 74. Ada tiga alternatif tabel yang dapat dibentuk dari relasi tersebut (perhatikan Gambar 75).

**Pilihan pertama** yaitu menggabungkan kedua entitas ke dalam satu buah tabel saja. Pada ilustrasi pertama (Gambar 75a), kedua entitas digabung menjadi sebuah tabel relasional bernama PRAMUNIAGA-RUANG\_KERJA. Di satu sisi, hal ini sangat memungkinkan karena relasi *one-to-one* berarti untuk seorang pramuniaga hanya dapat diasosiasikan dengan sebuah ruang kerja (kantor) saja. Demikian pula sebaliknya, sebuah ruang kerja hanya bisa ditempati oleh seorang pramuniaga saja. Oleh karena itu, kombinasi antara pramuniaga beserta ruang kerjanya (kantor) dapat disimpan cukup dengan satu baris (*record*). Meskipun demikian, ada sisi-sisi lain yang perlu dipertimbangkan. Setidaknya ada dua alasan mengapa pilihan untuk menyimpan dua data ke dalam sebuah baris tunggal bukan merupakan pilihan yang paling tepat. Alasan pertama, yaitu pramuniaga dan kantor (ruang kerja) tidak digambarkan dalam satu kotak yang sama pada ERD. Hal ini berarti keduanya dianggap berbeda secara bisnis, sehingga bagaimanapun juga keduanya perlu untuk tetap dipisah. Alasan kedua, yaitu modalitas bernilai nol pada pramuniaga. Dengan membaca diagram pada Gambar 74 dari kiri ke kanan, bisa saja suatu ruang kerja tidak ditempati oleh seorangpun pramuniaga. Apabila keadaan seperti ini disimpan pada satu record di tabel, maka sangat memungkinkan akan muncul banyak *record*/baris yang memiliki isian-nilai seperti: kode kantor, telepon dan ukuran namun tidak memiliki isian pada atribut-atribut terkait dengan pramuniaga (karena tidak ada pramuniaga yang *di-assign*). Record-record seperti ini tentu sangat tidak baik karena menyebabkan pemborosan memori/harddisk. Lebih buruk dari itu, apabila kode id pramuniaga dijadikan sebagai *primary key* dari tabel yang

terbentuk, skenario ini memungkinkan munculnya record yang tidak memiliki *primary key* (karena id pramuniaga bernilai kosong). Situasi seperti ini tentu tidak boleh terjadi.

Tabel PRAMUNIAGA						
	kd_pram	nama_pram	komisi_persen	lama_kerja	kd_ruang	telp
a.						

Tabel PRAMUNIAGA					Tabel RUANG_KERJA		
	kd_pram	nama_pram	komisi_persen	lama_kerja	kd_ruang	telp	ukuran
b.							

Tabel PRAMUNIAGA					Tabel RUANG_KERJA		
	kd_pram	nama_pram	komisi_persen	lama_kerja	kd_ruang	telp	ukuran
c.							

Gambar 75: 3 alternatif konversi relasi biner *one-to-one*

**Pilihan kedua** (perhatikan Gambar 75b) merupakan pilihan yang lebih baik dibandingkan pilihan pertama. Di sini akan terbentuk dua buah tabel yang terpisah yang disesuaikan dengan entitas-entitas yang terlibat, dalam hal ini adalah tabel PRAMUNIAGA dan tabel KANTOR. Untuk menjaga relasi antar entitas, misal pramuniaga X pada ruang kantor Y, maka nomor ruang kerja (*kd\_ruang*) dijadikan sebagai *foreign key* pada tabel *Pramuniaga*. Hal ini akan memastikan adanya relasi antara pramuniaga tertentu dengan ruang kantor tertentu. Lagi-lagi, coba perhatikan modalitas pada ERD di Gambar 74. Dengan kita membaca diagram tersebut dari kiri ke kanan, setiap pramuniaga di-*assign* ke tepat satu ruang kerja. Pada Gambar 75b terlihat tabel PRAMUNIAGA dengan sebuah field tambahan yaitu *kd\_ruang*. Dengan cara ini maka dapat dipastikan bahwa masing-masing field pada tabel PRAMUNIAGA akan memiliki nilai, termasuk field *kd\_ruang*. Bagaimana dengan ruang kerja yang kosong, alias yang tidak di-*assign* ke satupun pramuniaga? Jika menggunakan pilihan pertama (dibahas sebelumnya – menghasilkan satu tabel saja) maka problem ini akan menjadi problem serius. Namun pada pilihan kedua ini, problem tersebut sudah dapat di atasi. Apabila ada ruang kerja yang kosong,

maka data tersebut hanya akan tersimpan di tabel *RUANG\_KERJA* saja, dan tidak akan muncul sebagai *foreign key* pada tabel *PRAMUNIAGA*.

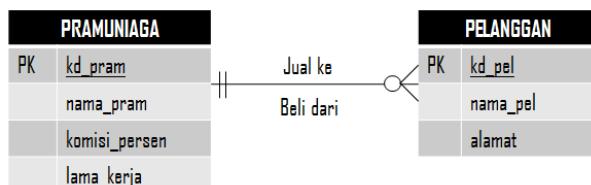
Pilihan selanjutnya ialah **pilihan ketiga** dimana kita menempatkan file kode pramuniaga (*kd\_pram*) sebagai *foreign key* pada tabel *RUANG\_KERJA*. Apabila Gambar 74 kita baca dari kanan ke kiri, dapat dijelaskan bahwa setiap ruang kerja bisa ditempati oleh maksimal seorang pramuniaga dan memungkinkan untuk tidak ditempati oleh seorangpun pramuniaga. Dengan demikian, pada tabel *RUANG\_KERJA* bisa jadi ada *record* atau baris yang memiliki isian atribut ruang kerja namun dengan field *kd\_pram* (*foreign key*) bernilai *null* alias kosong. Jika situasinya demikian, maka pilihan kedua jelas lebih baik dari pilihan ketiga, karena tidak akan ada field yang bernilai kosong.

Jadi kuncinya adalah seberapa efektif tabel-tabel tersebut merepresentasikan relasi data. Apabila kemunculan *atribut* atau *field* yang memungkinkan bernilai kosong dapat dihindari, maka sebaiknya dihindari saja.

### Relasi Binary – *One-to-Many*

Gambar 76 merupakan contoh relasi *one-to-many* yang akan kita selesaikan dengan mengkonversinya menjadi tabel relasional. Relasi ini menggambarkan relasi antara pramuniaga dan pelanggannya, dimana seorang pramuniaga bisa menjual barang ke banyak pelanggan atau bahkan tidak sama sekali. Sedangkan pada arah sebaliknya dijelaskan bahwa seorang pelanggan hanya bisa membeli barang dari satu pramuniaga saja. Aturan yang berlaku untuk relasi biner dengan *one-to-many relationshihp* yaitu, *primary key* pada tabel yang bernilai *one* dijadikan *foreign key* pada tabel yang bernilai *many*.

Untuk kasus PRAMUNIAGA-PELANGGAN yang sedang kita bahas ini, *field kd\_pram* (yaitu *primary key* pada tabel PRAMUNIAGA) dijadikan sebagai *foreign key* pada tabel PELANGGAN. Dengan demikian, tabel PELANGGAN akan memiliki sebuah tambahan field baru, bernama: *kd\_pram* (perhatikan Gambar 77).



**Gambar 76:** Contoh relasi biner *one-to-many* (1:M) yang akan diselesaikan

**Tabel PRAMUNIAGA**

<u>kd_pram</u>	<u>nama_pram</u>	<u>komisi_persen</u>	<u>lama_kerja</u>

**Tabel PELANGGAN**

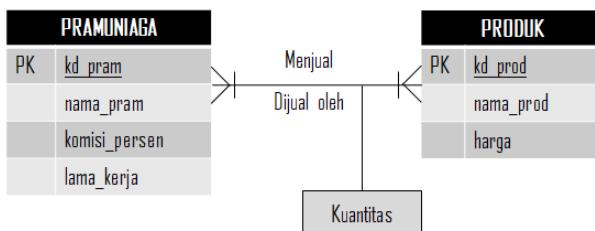
<u>kd_pel</u>	<u>nama_pel</u>	<u>alamat</u>	<u>kd_pram</u>

**Gambar 77: Hasil konversi relasi biner *one-to-many* ke dalam dua buah tabel relasional**

Field *kd\_pram* yang ada pada tabel PELANGGAN berfungsi untuk menghubungkan kedua tabel. Mengingat ERD pada Gambar 76 menjelaskan bahwa setiap pelanggan harus punya tepat satu (seorang) pramuniaga, maka tidak akan ada *atribut/field* yang bernilai kosong pada baris-baris di dalam tabel PELANGGAN.

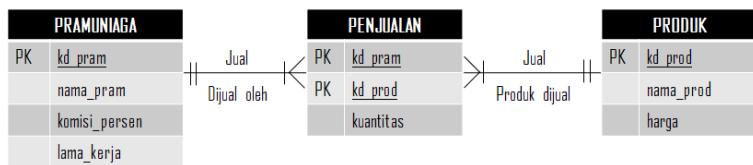
### Relasi Binary – *Many-to-Many*

Gambar 78 menjelaskan relasi *many-to-many* yang melibatkan entitas bernama PRAMUNIAGA dan PRODUK. Makna dari relasi *many-to-many* ialah bahwa seorang pramuniaga boleh menjual lebih dari satu (banyak) produk, demikian pula sebaliknya, satu jenis produk boleh dijual oleh banyak pramuniaga. Sesuai dengan Gambar 78, masing-masing entitas dibatasi dengan modalitas bernilai 1, dimana pramuniaga menjual minimal satu jenis produk, dan satu jenis produk dijual oleh minimal satu orang pramuniaga.



**Gambar 78: Relasi biner *many-to-many* antara PRAMUNIAGA dan PRODUK menghasilkan sebuah entitas baru yang memuat atribut *kuantitas***

Relasi biner *many-to-many* apabila dikonversi ke dalam table relasional, maka akan membentuk dua buah tabel plus sebuah tabel baru dengan atribut yang juga baru. Jika kita perhatikan pada Gambar 78, di sana terlihat adanya atribut *kuantitas* yang memiliki keterkaitan langsung dengan entitas PRAMUNIAGA dan PRODUK. Pada seorang pramuniaga, tentu perlu dicatat produk apa saja yang ia jual, berikut dengan jumlah atau kuantitasnya. Hal ini juga akan memudahkan apabila nantinya manajer ingin melihat juga berapa jumlah pramuniaga yang menjual suatu produk tertentu dan berapa banyak produk yang mereka jual. Oleh karena itu, kuantitas menjadi hal penting dalam relasi antara PRAMUNIAGA dan PRODUK. Dengan demikian, perlu dibuat sebuah entitas baru guna menampung atribut *kuantitas* ini.



**Gambar 79: Entitas baru bernama PENJUALAN terbentuk**

Atribut *kuantitas* menjadi cikal bakal terbentuknya suatu entitas baru (di sini diberi nama PENJUALAN) yang berisi atribut *kuantitas* itu sendiri ditambah dengan *primary key* dari entitas PRAMUNIAGA dan PRODUK. Sehingga, entitas PENJUALAN memiliki tiga buah atribut, yaitu *kd\_pram*, *kd\_prod*, dan *kuantitas* (perhatikan Gambar 79). Setelah diagram relasi selesai diperbaharui, langkah berikutnya adalah mengkonversinya ke dalam tabel. Struktur tabel yang terbentuk akan mengikuti diagram relasi yang telah di-update, seperti pada Gambar 79. Tabel hasil konversi terlihat pada Gambar 80, dimana ada tiga buah tabel yang dihasilkan.

**Tabel PRAMUNIAGA**

kd_pram	nama_pram	komisi_persen	lama_kerja

**Tabel PRODUK**

kd_prod	nama_prod	harga

**Tabel PENJUALAN**

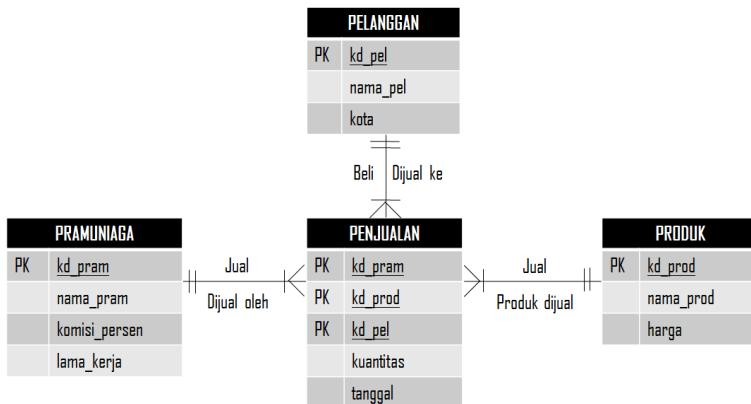
kd_pram	kd_prod	kuantitas

**Gambar 80:** Tabel hasil konversi dari ERD sebelumnya

## Relasi Ternary

Sebagaimana dijelaskan pada Bab 4 (subbab Elemen-elemen pada ERD → Relasi dan Kardinalitas), relasi *ternary* merupakan relasi yang memiliki derajat tiga, yaitu melibatkan tiga entitas. Jenis relasi ini digunakan ketika *binary relationship* dianggap tidak cukup akurat dalam menjelaskan makna antar 3 entitas. Dalam kondisi relasi dengan 3 entitas, maka akan memunculkan sebuah entitas baru sebagai entitas penhubung, yang berisikan semua *primary key* dari tiga entitas asli, ditambah dengan *field-field* yang sekiranya dibutuhkan dan menjadi pembeda antar satu kejadian dengan kejadian yang lain.

Gambar 81 merupakan contoh relasi *ternary* yang akan kita konversikan ke dalam tabel relasional. Pada gambar tersebut, terbentuk sebuah entitas baru bernama **PENJUALAN**. Isi dari entitas baru ini yaitu semua *primary key* dari 3 entitas asal (*kd\_pram*, *kd\_pel*, dan *kd\_prod*) ditambah dengan 2 *field* baru, yaitu *kuantitas* dan *tanggal*. Kuantitas diperlukan untuk mencatat berapa jumlah produk X yang dijual oleh pramuniaga Y kepada pelanggan Z. Selanjutnya atribut *tanggal* dibutuhkan untuk membedakan waktu jual/beli produk X yang dijual oleh pramuniaga Y kepada pelanggan Z.



Gambar 81: Relasi *ternary* yang akan dikonversi

Dalam konversi relasi yang bersifat *ternary*, semua entitas (PRAMUNIAGA, PELANGGAN, PRODUK) termasuk satu entitas baru yang terbentuk (PENJUALAN) akan diubah secara langsung menjadi tabel-tabel yang terpisah. Oleh karena itu, pada kasus ini akan ada 4 buah tabel yang tercipta, yaitu tabel: PRAMUNIAGA, PELANGGAN, PRODUK, dan PENJUALAN. Jumlah field, nama field dan *primary key* pada masing-masing tabel disesuaikan dengan entitas pada relasi yang tampak pada Gambar 81.

Tabel PRAMUNIAGA

kd_pram	nama_pram	komisi_persen	lama_kerja

Tabel PRODUK

kd_prod	nama_prod	harga

Tabel PELANGGAN

kd_pel	nama_pel	kota

Tabel PENJUALAN

kd_pram	kd_pel	kd_prod	tanggal	kuantitas

Gambar 82: Tabel hasil konversi dari ERD sebelumnya

Khusus untuk entitas baru yang terbentuk, yaitu PENJUALAN, *primary key*-nya adalah kumpulan dari 3 buah *primary key* yang diambil dari 3 tabel yang terhubung kepadanya, yaitu *kd\_pram*, *kd\_pel*, *kd\_prod* dikombinasikan dengan atribut *tanggal* untuk menciptakan keunikan sebagaimana sifat *primary key* yang sesungguhnya. Hasil konversi dapat dilihat pada Gambar 82.

# Bab 6

## BAHASA PADA BASIS DATA

## MODEL RELASIONAL

---

### Bab ini membahas:

- Operasi Aljabar Relasional
  - Operasi Kalkulus Relasional
  - Bahasa Query
- 

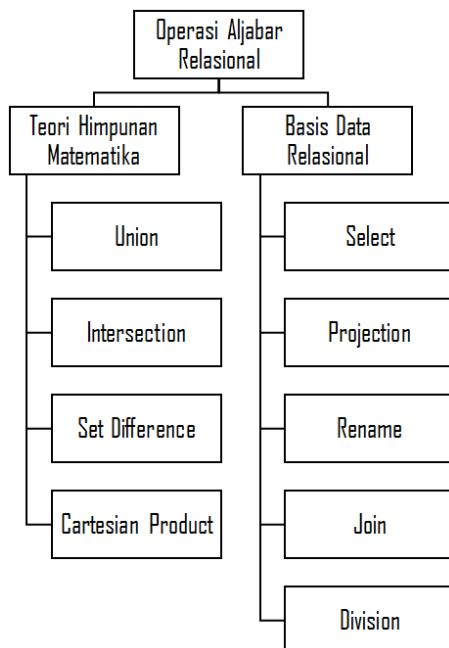
Sejak Edgar F. Codd memperkenalkan model relasional pada tahun 1970, ada dua jenis bahasa yang telah diusulkan dan diimplementasikan untuk digunakan pada basis data relasional. Jenis pertama yaitu bahasa non-prosedural yang berbasiskan kalkulus relasional, dengan bahasa *query* Quel. Sedangkan jenis kedua, yaitu bahasa prosedural yang menggunakan konsep aljabar relasional dengan bahasa *query* yang popular yaitu SQL (*Structured Query Language*).

Pada bahasa prosedural, *query* akan langsung mengarahkan DBMS tentang bagaimana cara mendapatkan data yang dimaksud. Sebaliknya, dalam bahasa non-prosedural, *query* hanya memberi kode atau *clue* tentang apa yang dibutuhkan dan menyerahkan ke sistem tentang proses yang ditempuh untuk mendapatkan hasil tersebut. Meskipun terdengar lebih mudah, namun ternyata bahasa prosedural jauh lebih popular dibandingkan dengan bahasa non-prosedural. Faktanya, SQL (sebagai bahasa *query* prosedural) secara luas diterima dan digunakan oleh *end user* sebagai media untuk berinteraksi dengan sistem relasional sekarang ini.

### Operasi Aljabar Relasional

Berdasarkan peruntukannya, operasi aljabar relasional secara umum dapat dikelompokkan ke dalam dua grup, yaitu untuk teori himpunan

matematika (*mathematics set theory*) dan khusus untuk basis data relasional (*specific for relational database*). Bahasan terkait teori himpunan meliputi: *union*, *intersection*, *set difference*, dan *cartesian product*. Sedangkan terkait basis data relasional mencakup: *select*, *project*, *rename*, *join*, dan *division* (perhatikan Gambar 83).

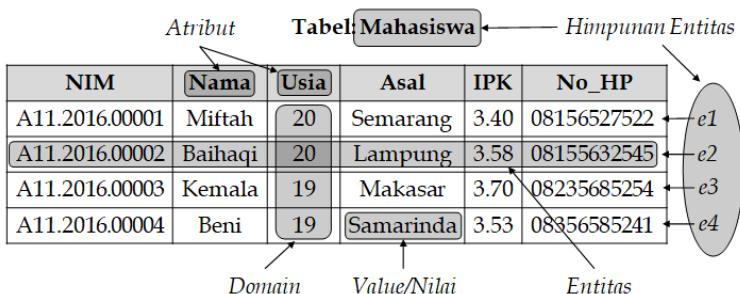


**Gambar 83:** Pembagian materi pada operasi aljabar relasional

Beberapa operasi yang dilakukan pada relasi tunggal disebut dengan operasi tunggal atau *unary operation*. Sedangkan operasi yang dilakukan dengan melibatkan dua relasi, dikenal dengan istilah operasi biner atau *binary operation*. Beberapa contoh dari operasi tunggal (*unary operation*) adalah: *Select*, *Project*, dan *Rename*. Sedangkan contoh dari operasi biner adalah: *Cartesian Product*, *Union*, dan *Difference*.

Sebelum kita membahas lebih jauh terkait operasi aljabar relasional, ada baiknya kita mengulas kembali konsep yang telah dibahas pada Bab 3 sebelumnya, yaitu terkait tabel beserta komponen-komponennya dalam E-R Model. Perhatikan gambar 84 berikut ini.

- **Enterprise:** merujuk pada suatu organisasi, seperti sekolah, universitas, bank, dll.
- **Entity:** merujuk pada objek pada dunia nyata, seperti mahasiswa, siswa, karyawan bank, dll. Nama lain yang sering digunakan untuk menggantikan entity/entitas adalah *row*, *tuple*, *record*, dan juga baris.
- **Attributes:** merupakan karakteristik dari suatu entitas, contoh: entitas mahasiswa memiliki atribut seperti NIM, nama, alamat, tgl lahir, dll.
- **Value:** merupakan informasi atau data yang disimpan pada setiap atribut masing-masing data.
- **Entity Sets:** entitas-entitas yang memiliki atribut yang sama menghasilkan himpunan entitas.
- **Domain (Value Set):** merupakan himpunan dari semua nilai atau informasi tentang suatu atribut.



**Gambar 83: Komponen-komponen dalam E-R Model**

Dari ilustrasi di atas, dapat diketahui bahwa kolom pada tabel merepresentasikan atribut dari suatu domain. Domain sendiri ialah kumpulan atau himpunan dari semua nilai-nilai yang mungkin ada pada kolom tersebut. Di sisi lain, basis data relasional ialah kumpulan tabel-tabel dengan beberapa *integrity constraints* (dibahas pada Bab 4). Perhatikan contoh berikut:

kd_pegawai	nama	gaji

**Gambar 84: Contoh *integrity constraints* pada suatu tabel**

Ada beberapa operasi aljabar yang umum digunakan dalam basis data, diantaranya yaitu:

- Operasi *Select* ( $\sigma$ ): digunakan untuk memilih suatu nilai atau *tuples* atau *values* pada tabel yang sesuai dengan kondisi yang diberikan. Simbol yang digunakan untuk operasi ini adalah sigma kecil ( $\sigma$ ).
- Operasi *Projection* ( $\pi$ ):
- Operasi *Union* ( $\cup$ ):
- Operasi *Difference* ( $-$ ):
- Operasi *Intersection* ( $\cap$ ):
- Operasi *Cartesian Product* ( $\times$ ):
- Operasi *Natural Join* ( $\bowtie$ ):

## Operasi *Select* ( $\sigma$ )

Operasi *select* digunakan apabila kita hendak memilih nilai tertentu pada suatu tabel. Untuk merepresentasikan operasi ini, maka digunakan simbol *sigma* kecil (*lowercase sigma*), yaitu  $\sigma$ . Notasi aljabar relasionalnya adalah sebagai berikut:

$$\sigma_p(R)$$

### Keterangan:

- $\sigma$  : sigma kecil sebagai simbol operasi *select*
- $p$  : predikat (ditulis lebih kecil atau subscript)
- $(R)$  : tabel atau relasi

Operasi *select* dilakukan pada basis data tunggal (*single database*). Operasi ini berguna untuk memilih data atau sekumpulan data yang memenuhi kriteria/kondisi tertentu. Untuk menentukan predikat yang digunakan, kita dapat menggunakan operator relasi/pembanding (lihat tabel 9) dan juga operator logika (lihat tabel 10).

**Tabel 9: Operator pembanding (*comparison operators*)**

Operator	Arti	Contoh
=	Sama dengan ( <i>equal to</i> )	nama = “Adi”
<>	Tidak sama dengan ( <i>not equal to</i> )	kota <> “Semarang”
>	Lebih besar dari ( <i>greater than</i> )	usia > 25

$\geq$	Lebih besar dari atau sama dengan ( <i>greater than equal to</i> )	$\text{gaji} \geq 1000000$
$<$	Kurang dari ( <i>less than</i> )	keuntungan $< 50000$
$\leq$	Kurang dari atau sama dengan ( <i>less than equal to</i> )	nilai $\leq 50$

Tabel 10: Operator logika (*logical operators*)

Operator	Arti	Contoh
AND	Logika AND (dan)	nama = "Adi" AND kota = "Semarang"
OR	Logika OR (atau)	usia = 25 OR usia $\geq 40$

Agar operasi *select* dapat lebih mudah dipahami, maka kita akan mencobanya secara langsung dengan menggunakan tabel MAHASISWA sebagaimana tampak pada Gambar 85.

Tabel MAHASISWA

NIM	nama	nilai_daspro	nilai_alpro	nilai_strukdat
A2001	Adi	70	80	75
A2002	Beni	70	68	88
A2003	Candra	82	85	78
A2004	Danu	65	75	76
A2005	Egi	90	86	88

Gambar 85: Tabel MAHASISWA

Berikut ini merupakan contoh-contoh penggunaan aljabar untuk operasi *select*:

1. **Tujuan:** memilih (*select*) data dari tabel MAHASISWA dimana nilai daspro  $\geq 82$ .

### Solusi

Notasi :  $\sigma_{\text{nilai\_daspro} \geq 82}$  (MAHASISWA)

Hasil :

NIM	nama	nilai_daspro	nilai_alpro	nilai_strukdat
A2003	Candra	82	85	78
A2005	Egi	90	86	88

Gambar 86: Hasil query pertama pada tabel MAHASISWA

Dari Gambar 86, tampak bahwa query yang dilakukan memunculkan dua baris data saja, yaitu mahasiswa bernama Candra dan Egi, yang mana keduanya memiliki nilai daspro 82 dan 90 (di atas atau sama dengan 82).

2. **Tujuan:** memilih (*select*) data dari tabel MAHASISWA dimana nama mahasiswanya adalah Candra.

### Solusi

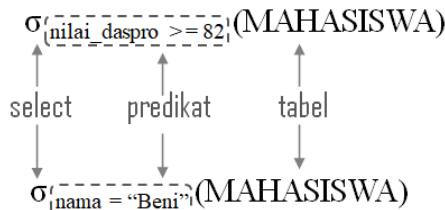
Notasi :  $\sigma_{\text{nama} = \text{"Beni"}}(\text{MAHASISWA})$

Hasil :

NIM	nama	nilai_daspro	nilai_alpro	nilai_strukdat
A2002	Beni	70	68	88

Gambar 87: Hasil query ke-dua pada tabel MAHASISWA

Pada Gambar 87 terlihat hanya 1 baris data saja yang ditampilkan, karena hanya ada 1 mahasiswa yang bernama “Beni” sesuai predikat yang diberikan. Gambar 88 menunjukkan bagian-bagian dari notasi yang telah diberikan.



Gambar 88: Bagian-bagian pada notasi aljabar relasional

### Operasi *Projection* ( $\pi$ )

Operasi project atau *projection* termasuk operasi unary, yaitu operasi yang hanya melibatkan satu tabel saja. Operasi yang disimbolkan dengan aksara Yunani pi ( $\pi$ ) ini digunakan untuk menampilkan field atau kolom tertentu pada suatu tabel. Berikut ini adalah notasi aljabar relasional dari operasi *projection*:

$\pi_p(R)$
------------

**Keterangan:**

- $\pi$  : pi sebagai simbol operasi *projection*
- $p$  : predikat (ditulis lebih kecil atau subscript)
- $(R)$  : tabel atau relasi

Pada dasarnya, operasi *projection* ini tidak melibatkan operator apapun sebagaimana pada operasi *select*. Namun, operasi ini juga bisa digabungkan dengan operasi *select* apabila kita menginginkan hasil *query* yang lebih spesifik. Masih menggunakan tabel MAHASISWA yang ada pada Gambar 85, berikut ini adalah contoh penggunaan operasi *projection* secara mandiri dan ketika digabung dengan operasi *select*.

1. **Tujuan:** menampilkan kolom NIM, nama, dan nilai\_alpro saja.

**Solusi**

Notasi :  $\pi_{NIM, \text{nama}, \text{nilai\_alpro}}(\text{MAHASISWA})$

Hasil :

NIM	nama	nilai_alpro
A2001	Adi	80
A2002	Beni	68
A2003	Candra	85
A2004	Danu	75
A2005	Egi	86

**Gambar 89:** Hasil *query* pertama menggunakan operasi *projection*

2. **Tujuan:** menampilkan kolom nama dan nilai\_alpro dimana nilai untuk mata kuliah alpro lebih besar dari 80 ( $\text{nilai\_alpro} > 80$ ).

**Solusi**

Notasi :  $\pi_{\text{nama}, \text{nilai\_alpro}}(\sigma_{\text{nilai\_alpro} > 80}(\text{MAHASISWA}))$

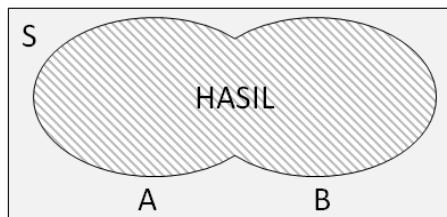
Hasil :

nama	nilai_alpro
Candra	85
Egi	86

**Gambar 90:** Hasil *query* ke-dua menggunakan operasi *projection*

## Operasi Union ( $\cup$ )

Operasi *union* direpresentasikan dengan symbol  $\cup$  (sama dengan symbol union pada teori himpunan). *Union* atau gabungan dari dua relasi dibentuk dengan mengkombinasikan nilai atau *tuple* dari relasi pertama ke relasi kedua untuk membentuk relasi ketiga.



Gambar 91: Ilustrasi operasi *union* pada diagram Venn

Bagian yang diarsir pada Gambar 91 menunjukkan bawah nilai-nilai yang ada pada diagram A digabungkan dengan nilai-nilai pada diagram B. Artinya, hasil dari operasi *union*  $A \cup B$  pada ilustrasi di atas adalah semua anggota himpunan A maupun B (wilayah yang diarsir). Notasi aljabar operasi *union* adalah sebagai berikut:

$$\pi_{p1}(R) \cup \pi_{p2}(R)$$

### Keterangan:

- $\pi$  : pi sebagai simbol operasi *projection*
- $p1$  : predikat pertama (ditulis lebih kecil atau subscript)
- $p2$  : predikat ke-dua (ditulis lebih kecil atau subscript)
- $(R)$  : tabel atau relasi
- $\cup$  : simbol operasi *union*

Sebagaimana tampak pada notasi di atas, operasi *union* biasa disandingkan dengan operasi lain, seperti *projection*. Operasi *projection* berperan dalam memilih kolom-kolom yang sama dari dua tabel atau lebih. Sedangkan operasi *union* bertugas menggabungkan dua atau lebih kolom yang sama tersebut.

Merujuk pada notasi di atas,  $p1$  dan  $p2$  harus bernilai sama. Jika tidak, maka cara dan fungsinya akan berbeda (akan dibahas pada bagian operasi *cartesian product*). Sedangkan jika predikat 1 dan 2 bernilai sama, maka kedua kolom pada dua tabel akan disatukan (asumsi hanya melibatkan 2

tabel saja). Apabila saat digabungkan memunculkan nilai-nilai yang kembar, maka hanya satu nilai saja yang diambil. Perhatikan contoh berikut:

Tabel MHS

NIM	nama	tgl_lahir
A2001	Adi	300499
A2002	Beni	250398
A2003	Candra	010699
A2004	Danu	1210999
A2005	Egi	231297

Tabel IPK

NIM	nilai_ipk
A2001	3.8
A2002	3.4
A2003	3.1
A2006	3.7
A2007	3.5

Gambar 92: Ilustrasi tabel MHS dan tabel IPK

**Tujuan:** menggabungkan kolom NIM pada tabel MHS dan IPK, menggunakan operasi UNION.

### Solusi

Notasi :  $\pi_{NIM}(\text{MHS}) \cup \pi_{NIM}(\text{IPK})$

Hasil :

NIM
A2001
A2002
A2003
A2004
A2005
A2006
A2007

Gambar 93: Hasil operasi UNION

Jika hendak menggabungkan semua data tanpa mengeliminasi yang redundan, digunakan operasi UNION ALL. Contohnya hasilnya adalah sebagai berikut:

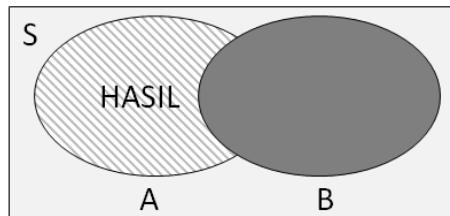
nama
A2001
A2001
A2002
A2002
A2003
A2003
A2004
A2005
A2006
A2007

Gambar 94: Hasil operasi **UNION ALL**

Perbedaan antara *union* dan *union all* bisa Anda amati dengan membandingkan gambar 92 dan 93. Pada gambar 92 (*union*), NIM A2001, A2002 dan A2003 hanya muncul masing-masing satu kali. Operasi *union* mengeliminasi data-data kembar dan memunculkannya hanya satu kali saja. Sedangkan pada gambar 93 (*union all*), NIM A2001, A2002 dan A2003 dimunculkan sesuai jumlah yang sebenarnya, yaitu dua kali.

### Operasi Set Difference (-)

Operasi *difference* direpresentasikan dengan simbol *minus* (-). Hasil operasi *difference* antar dua relasi membentuk satu relasi baru, yaitu relasi atau tabel yang mengandung nilai/tuple yang ada di relasi pertama namun tidak ada pada relasi ke-dua. Apabila A dan B adalah dua relasi, maka hasil dari A – B adalah sekumpulan nilai pada A yang tidak adai pada B. Syaratnya, A dan B memiliki kesamaan dalam hal tipe maupun nama atribut/kolom pada tabel/relasi. Wilayah yang diarsir pada Gambar 95 menunjukkan hasil dari operasi A – B.



**Gambar 95:** Ilustrasi operasi *set difference* pada diagram Venn

Sama halnya dengan operasi *union*, operasi ini juga berlaku jika disandingkan dengan operasi lain, seperti operasi *projection* yang telah dibahas sebelumnya. Berikut ini merupakan contoh operasi *difference*, dengan menggunakan tabel MHS dan IPK:

1. **Tujuan:** mencari NIM-NIM mahasiswa yang hanya ada di tabel pertama (MHS) saja dan tidak ada pada tabel ke dua (IPK).

**Solusi**

Notasi :  $\pi_{\text{NIM}}(\text{MHS}) - \pi_{\text{NIM}}(\text{IPK})$

Hasil :

NIM
A2004
A2005

**Gambar 96:** Hasil operasi *difference* (MHS – IPK)

2. **Tujuan:** mencari NIM-NIM mahasiswa yang ada di tabel IPK tapi tidak ada pada tabel MHS.

**Solusi**

Notasi :  $\pi_{\text{NIM}}(\text{IPK}) - \pi_{\text{NIM}}(\text{MHS})$

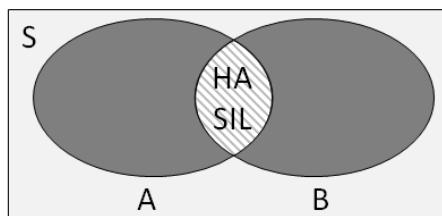
Hasil :

NIM
A2006
A2007

**Gambar 97:** Hasil operasi *difference* (MAHASISWA-2 – MAHASISWA-1)

## Operasi *Intersection* ( $\cap$ )

Operasi *intersection* direpresentasikan dengan simbol  $\cap$ . Jika A dan B adalah dua buah relasi, maka  $A \cap B$  merupakan kumpulan nilai-nilai yang ada pada A dan juga B. Gambar 98 merupakan ilustrasi operasi *intersection* pada diagram Venn.



Gambar 98: Ilustrasi operasi *intersection* pada diagram Venn

$$A \cap B = A - (A - B)$$

Dalam teori himpunan, *intersection* diartikan sebagai irisan, tepatnya nilai yang ada pada ke-dua himpunan terkait. Sedangkan dalam konsep relasi, *intersection* dari dua relasi akan membentuk relasi ke-tiga yang mengandung nilai-nilai dari relasi ke-1 dan ke-2. Dengan masih menggunakan tabel MHS dan IPK, hasil operasi *intersection* akan menghasilkan relasi baru berisi nama-nama yang ada di tabel MHS dan IPK. Contoh penerapannya adalah sebagai berikut:

**Tujuan:** mencari NIM-NIM mahasiswa yang ada pada tabel MHS dan juga pada tabel IPK.

### Solusi

Notasi :  $\pi_{NIM}(MHS) \cap \pi_{NIM}(IPK)$

Hasil :

NIM
A2001
A2002
A2003

Gambar 99: Hasil operasi *intersection* ( $A \cap B$ )

Hasil dari operasi  $(A \cap B)$  di atas sama persis dengan penggunaan operasi *set difference*  $A - (A - B)$ , yang mana notasi aljabarnya adalah sebagai berikut:

$$\pi_{NIM}(MHS) - (\pi_{NIM}(MHS) - \pi_{NIM}(IPK))$$

Singkatnya, operasi *intersection* akan memunculkan nilai-nilai yang ada pada relasi 1 dan 2.

### Operasi *Cartesian Product* (X)

Apabila operasi *union* digunakan untuk menyatukan dua atau lebih kolom yang sama, maka operasi *cartesian product* bisa melakukan yang lebih kompleks lagi. Ia bisa menggabungkan dua tabel atau lebih yang kolom-kolomnya tidak sama persis. Singkatnya, *cartesian product* digunakan untuk menggabungkan informasi dari dua relasi berbeda.

Operasi *cartesian product* direpresentasikan dengan simbol X. Operasi ini akan menggabungkan semua data yang ada pada tabel A dan tabel B. Jumlah barisnya pun merupakan hasil perkalian dari jumlah baris pada tabel A dan tabel B. Oleh karena itu, operasi *cartesian product* bersifat komutatif, yaitu dibolak-balik hasilnya sama saja ( $A \times B = B \times A$ ).

Tabel MHS_NAMA		Tabel MHS_IPK	
NIM	nama	NIM	IPK
A2001	Adi	A2001	3.88
A2002	Beni	A2003	3.45
A2003	Candra	A2004	3.75
A2004	Danu		
A2005	Egi		

Gambar 100: Tabel MHS\_NAMA dan MHS\_IPK

Untuk lebih jelasnya, mari perhatikan contoh berikut ini!

Misal, kita memiliki dua buah tabel di dalam basis data atau database, yaitu MHS\_NAMA dan MHS\_NILAI (lihat Gambar 100). Tujuan kita saat ini adalah ingin mengetahui nilai IPK mahasiswa atas nama “Candra”. Barangkali secara kasat mata dengan menggunakan mata manusia, kita dapat secara mudah mengetahui bahwa Candra memiliki IPK 3,45, meskipun tabelnya terpisah. Namun di sini, yang akan kita bahas adalah proses yang harus ditempuh oleh

suatu basis data, sehingga bisa memunculkan jawaban IPK Candra adalah 3,45. Langkah-langkahnya adalah sebagai berikut:

1. **Tujuan** : melakukan operasi *cartesian product* untuk tabel MHS\_NAMA dan MHS\_IPK.

### Solusi

Notasi :  $\sigma$  (MHS\_NAMA) X (MHS\_IPK)

Hasil :

NIM	nama	NIM	IPK
A2001	Adi	A2001	3.88
A2002	Beni	A2001	3.88
A2003	Candra	A2001	3.88
A2004	Danu	A2001	3.88
A2005	Egi	A2001	3.88
A2001	Adi	A2003	3.45
A2002	Beni	A2003	3.45
A2003	Candra	A2003	3.45
A2004	Danu	A2003	3.45
A2005	Egi	A2003	3.45
A2001	Adi	A2004	3.75
A2002	Beni	A2004	3.75
A2003	Candra	A2004	3.75
A2004	Danu	A2004	3.75
A2005	Egi	A2004	3.75

Gambar 101: Tampak hasil operasi *cartesian product* (MHS\_NAMA X MHS\_IPK)

2. **Tujuan** : menampilkan baris data dengan nama “Candra” saja.

### Solusi

Notasi :  $\sigma_{\text{nama} = \text{"Candra"}}$  (MHS\_NAMA X MHS\_IPK)

Hasil :

NIM	nama	NIM	IPK
A2003	Candra	A2001	3.88
A2003	Candra	A2003	3.45
A2003	Candra	A2004	3.75

**Gambar 102:** Proses nomor 2 menampilkan hasil yang belum akurat

Dengan penambahan kondisi  $nana = \text{"Candra"}$ , ternyata masih belum bisa menghasilkan output yang akurat. Pada Gambar 102 masih tampak adanya perbedaan kolom NIM di sebelah kiri nama Candra dan sebelah kanannya. Sampai tahap ini, kita masih belum bisa mengetahui secara pasti berapa IPK Candra. Sehingga, diperlukan langkah untuk membandingkan antara kolom NIM di sebelah kiri (milik MHS\_NAMA) dan kolom NIM di sebelah kanan (milik MHS\_IPK). Apabila NIM di kiri sama dengan NIM di kanan, maka dapat diketahui berapa IPK Candra. Terkait membandingkan dua nilai ini, akan dibahas pada tahap selanjutnya.

3. **Tujuan :** mencari data dimana kolom NIM sebelah kiri sama dengan kolom NIM di sebelah kanan.

### Solusi

Notasi :  $\sigma_{MHS\_NAMA.NIM = MHS\_IPK.NIM} (\sigma_{\text{nama} = \text{"Candra"} } (MHS\_NAMA \times MHS\_IPK))$

Hasil :

NIM	nama	NIM	IPK
A2003	Candra	A2003	3.45

**Gambar 103:** Tampak bahwa hasil operasi pada langkah 3 sudah akurat

Pada tahap ini, hasil yang diperoleh (lihat Gambar 103) sudah akurat. NIM di sebelah kiri dan kanan bernilai sama. Dengan demikian, dapat dipastikan bahwa IPK mahasiswa yang bernama Candra adalah 3,45. Sayangnya, tabel tersebut masih memuncul 2 buah kolom NIM. Mestinya kolom NIM yang muncul hanya satu saja, yaitu pada sisi ujung kirim. Oleh karena itu, dibutuhkan sebuah operasi *projection* untuk mengatur kemunculan kolom agar sesuai dengan yang dibutuhkan. Perhatikan langkah nomor 4 berikut ini!

4. **Tujuan :** menampilkan tiga kolom data saja untuk mahasiswa bernama “Candra”, yaitu kolom NIM, nama, dan IPK.

**Solusi**

Notasi :  $\pi_{\text{NIM, nama, IPK}} (\sigma_{\text{MHS\_NAMA.NIM}} = \text{MHS\_IPK.NIM} \ (\sigma_{\text{nama}} = \text{"Candra"}) (\text{MHS\_NAMA} \times \text{MHS\_IPK}))$

Hasil :

NIM	nama	IPK
A2003	Candra	3.45

**Gambar 104: Hasil akhir dari operasi cartesian product**

Gambar 104 menunjukkan hasil akhir dari rangkaian operasi *cartesian product* yang disertai kombinasi dengan beberapa operasi-operasi lain (*select* dan *product*) akhirnya dapat memunculkan tujuan yang diharapkan, yaitu IPK dari mahasiswa yang bernama Candra: 3,45.

### Operasi Natural Join ( $\bowtie$ )

Operasi *naturan join* dinotasikan dengan menggunakan simbol  $\bowtie$ . Operasi ini memungkinkan terjadi hanya jika relasi A dan B memiliki setidaknya satu atribut dengan nama yang sama. Tahap-tahap dalam melakukan *natural join* adalah sebagai berikut:

1. Menemukan *cartesian product*-nya terlebih dahulu.
2. Pada atribut-atribut yang sama, temukan nilai-nilai yang sama, kemudian masukkan ke dalam tabel relasi baru. Tabel baru tersebut dinamakan sebagai *equijoin*.
3. Hilangkan atribut-atribut yang sama, dan hasil ini lah yang disebut dengan *natural join*.

Untuk lebih memahaminya, perhatikan studi kasus berikut ini.

Tabel MHS_NAMA		Tabel MHS_IPK	
NIM	nama	NIM	IPK
A2001	Adi	A2001	3.88
A2003	Candra	A2003	3.45
A2004	Danu	A2004	3.75

**Gambar 105:** Tabel MHS\_NAMA dan MHS\_IPK yang akan digabungkan menggunakan operasi *natural join*

**Tujuan:** menggabungkan dua tabel pada Gambar 105 (tabel MHS\_NAMA dan MHS\_IPK) sekaligus menyesuaikan atribut-atributnya.

#### Solusi

Notasi :  $\pi_{NIM, nama, IPK} (MHS\_NAMA \bowtie MHS\_IPK)$

Hasil :

NIM	nama	IPK
A2001	Adi	3.88
A2003	Candra	3.45
A2004	Danu	3.75

**Gambar 106:** Hasil akhir operasi *natural join* antara tabel MHS\_NAMA dan MHS\_IPK

Agar bisa memperoleh hasil seperti tampak pada Gambar 105, maka ada beberapa langkah yang harus dilalui, yaitu:

1. Tujuan: menemukan hasil *cartesian product* antar dua tabel (MHS\_NAMA dan MHS\_IPK).

Notasi:  $\sigma (MHS\_NAMA) \times (MHS\_IPK))$

Hasil:

NIM	nama	NIM	IPK
A2001	Adi	A2001	3.88
A2003	Candra	A2001	3.88
A2004	Danu	A2001	3.88
A2001	Adi	A2003	3.45
A2003	Candra	A2003	3.45
A2004	Danu	A2003	3.45
A2001	Adi	A2004	3.75
A2003	Candra	A2004	3.75
A2004	Danu	A2004	3.75

**Gambar 107:** Hasil operasi *cartesian product* antara tabel MHS\_NAMA dan tabel MHS\_IPK

2. Tujuan: membuat tabel equijoin, dengan cara menemukan nilai yang sama pada atribut yang sama pula. Pada Gambar 107, tampak adanya dua atribut dengan nama yang sama, yaitu atribut NIM. Maka disini yang perlu dilakukan adalah menemukan baris data dimana nilai pada NIM di sebelah kiri sama dengan nilai NIM pada baris di sebelah kanan. Perhatikan baris ke-1, 5, dan 9 memiliki nilai NIM di kiri yang sama dengan NIM di kanan. Maka ketiga baris data tersebut kita ambil untuk dijadikan sebagai tabel *equijoin*.

Notasi:  $\sigma_{MHS\_NAMA.NIM = MHS\_IPK.NIM} (MHS\_NAMA) \times (MHS\_IPK)$

Hasil:

NIM	nama	NIM	IPK
A2001	Adi	A2001	3.88
A2003	Candra	A2003	3.45
A2004	Danu	A2004	3.75

**Gambar 108:** Tampak tabel *equijoin* yang telah terbentuk

3. Tujuan: menghilangkan salah satu atribut yang sama, dalam hal ini adalah menghilangkan salah satu atribut NIM, karena pada tabel equijoin terdapat dua atribut NIM dengan nilai yang sama persis.

Notasi:  $\pi_{NIM, nama, IPK} (\sigma_{MHS\_NAMA.NIM = MHS\_IPK.NIM} (MHS\_NAMA) \times (MHS\_IPK))$

Hasil:

NIM	nama	IPK
A2001	Adi	3.88
A2003	Candra	3.45
A2004	Danu	3.75

Gambar 109: Tampak tabel *equijoin* dengan satu atribut NIM saja

Gambar 109 merupakan hasil akhir dari operasi *natural join*. Pada tabel tersebut, tidak lagi ditemukan atribut serta data dengan nama yang sama pula. Dengan demikian, penggabungan dua tabel, yaitu antara tabel MHS\_NAMA dan MHS\_IPK telah berhasil dilakukan, khususnya menggunakan operasi *natural join*.

Jika diperhatikan, terdapat perbedaan antara notasi pada point nomor 3 dengan notasi awal pada bagian solusi. Notasi awal adalah:

$$\pi_{NIM, nama, IPK} (MHS\_NAMA \bowtie MHS\_IPK)$$

Sedangkan notasi pada point nomor 3, adalah:

$$\pi_{NIM, nama, IPK} (\sigma_{MHS\_NAMA.NIM = MHS\_IPK.NIM} (MHS\_NAMA) \times (MHS\_IPK))$$

Sekilas terlihat berbeda namun sebetulnya sama. Di sini, operasi *natural join* ( $\bowtie$ ) dapat menjadi pengganti dari gabungan operasi *select* ( $\sigma$ ) dan *cartesian product* ( $\times$ ). Dalam contoh kasus di atas, maka notasi aljabar (MHS\_NAMA  $\bowtie$  MHS\_IPK) memiliki nilai yang sama dengan ( $\sigma_{MHS\_NAMA.NIM = MHS\_IPK.NIM} (MHS\_NAMA) \times (MHS\_IPK)$ ). Oleh karena itu, jelas bahwa operasi *natural join* hanya menghasilkan nilai atau tupel yang memiliki nilai yang sama pada 2 atribut dengan nama yang sama juga, namun pada 2 tabel relasi yang berbeda.

## Operasi Kalkulus Relasional

Kalkulus relasional merupakan bahasa *query* non-prosedural yang menggunakan pendekatan kalkulus. Di sini, kita hanya perlul mendeskripsikan informasi yang dikehendaki tanpa perlu memberitahu prosedur spesifik untuk memperoleh informasi tersebut. Singkatnya, kalkulus relasional fokus pada data apa yang mau diambil, bukan tentang bagaimana cara mengambil data itu.

Kalkulus relasional tidak seperti kalkulus yang ada pada matematika yang diantaranya mengupas tentang diferensial serta integral. Pada basis data ini, digunakan istilah kalkulus, khususnya kalkulus relasional, karena memiliki keterkaitan dengan kalkulus predikat (*predicate calculus*) yang merupakan cabang dari logika simbolik. Kalkulus relasional memiliki dua bentuk, yaitu:

- Kalkulus Relasional Tuple
- Kalkulus Relasional Domain (diusulkan oleh Lacroix dan Pirotte pada tahun 1977).

### Kalkulus Relasional Tuple

Kalkulus Relasional Tuple (*Tuple Relational Calculus*) pertama kali diusulkan oleh Edgar F. Codd pada tahun 1972. Pada kalkulus relasional jenis ini, kita harus menemukan suatu tuple dimana predikatnya bernilai benar.

Perhatikan contoh berikut ini:

Untuk menentukan range tuple dari variable S sebagai relasi bernama Staf, dapat ditulis:

Staf (S)

Untuk menyatakan query “temukan himpunan dari semua tuple S dimana F(S) bernilai benar”, ditulis:

$\{S \mid F(S)\}$

Di sini, F dikatakan sebagai Formula (tepatnya *well-formed formula* atau *wff* pada logika matematika). Sebagai contoh, untuk menyatakan suatu query “tampilkan atribut kd\_staff, nama, jabatan, j\_kel, tgl\_masuk, dan gaji dari semua staf yang gajinya di atas Rp. 5.000.000, dapat kita tulis:

$\{S \mid \text{Staf}(S) \wedge S.\text{gaji} > 5000000\}$

Contoh lain:

{G | Guru (G) dan G.gaji>3000000}

{G | Guru (G) dan G.gaji>3000000}

## Bahasa Query

Bahasa Query (*Query Language*) merujuk pada suatu bahasa pemrograman komputer yang secara khusus digunakan untuk meminta, mengambil, atau menampilkan data dari suatu basis data. Data-data yang dihasilkan itu akan diolah lebih lanjut hingga menjadi suatu informasi yang berguna. Perintah yang dikirimkan untuk keperluan tersebut dikenal dengan perintah *query* atau bisa disebut juga dengan *query* saja.

Sistem basis data komersial membutuhkan bahasa *query* yang sederhana dan mudah dipahami. Berangkat dari kebutuhan ini, maka muncullah SQL (*Structure Query Language*), yang sekarang ini telah umum digunakan dalam dunia basis data. SQL didasarkan pada gabungan konsep aljabar relasional dan kalkulus relasional. Selain berguna untuk melakukan *query* pada basis data, SQL juga memiliki beberapa fitur lain, seperti:

1. Mendefinisikan struktur dari suatu data, seperti: menentukan tipe data, panjang karakternya, dll.
2. Memodifikasi data di database.
3. Mengatur batasan terkait keamanan data (*security constraints*).

SQL kini digunakan sebagai standard yang digunakan pada basis data relasional, berkat kemampuannya dalam mengakses beragam RDBMS (*Relational Data-Base Management System*), seperti: Oracle, Sybase, MySQL, Informix, Postgre, dll. Penjelasan lebih dalam terkait SQL akan dibahas pada Bab 8. Pada bagian ini hanya akan dibahas tentang konversi dari aljabar relasional menjadi bentuk bahasa *query*.

### Query Select

Sebagaimana dijelaskan pada subbab Operasi Aljabar Relasional, operasi select digunakan untuk memilih nilai tertentu dari satu atau beberapa tabel. Operasi ini direpresentasikan dengan simbol sigma kecil ( $\sigma$ ). Sekedar mengingatkan, rumus dasarnya adalah sebagai berikut:

$$\sigma_p (R)$$

Untuk penggunaan notasi sekaligus bahasa SQL-nya, perhatikan contoh-contoh berikut ini:

1. Menampilkan semua data yang ada pada tabel MAHASISWA,

**Notasi:**

$$\sigma (\text{MAHASISWA})$$

**Perintah SQL:**

```
select * from MAHASISWA
```

2. Menampilkan data mahasiswa yang bernama ‘Beni’ dari tabel MAHASISWA.

**Notasi:**

$$\sigma_{\text{nama} = \text{"Beni"}} (\text{MAHASISWA})$$

**Perintah SQL:**

```
select * from MAHASISWA where nama = "Beni"
```

**Keterangan:**

- **select:** merupakan bahasa *query* dari operasi sigma kecil ( $\sigma$ ).
- **\* (asterisk):** simbol bintang (asterisk) digunakan untuk mewakili semua kolom pada tabel. Untuk melakukan operasi *select* pada kolom tertentu saja, maka simbol bintang ini diganti dengan nama-nama kolom yang ingin diproses.
- **where:** klausa yang wajib ditulis apabila ada predikat atau kondisi yang harus dipenuhi pada *query*.
- **from MAHASISWA:** nama tabel yang akan digunakan. Pada bahasa *query*, nama tabel tidak perlu diapit dengan tanda kurung.

## Query Projection

*Query* ini digunakan apabila kita hendak menentukan kolom tertentu pada suatu tabel. Sebagai contoh, apabila kita ingin menampilkan 2 kolom saja dari tabel MAHASISWA, seperti kolom *nama* dan *nilai\_alpro*, maka Anda dapat menggunakan *query* ini. Operasi *projection* dinotasikan dengan aksara *pi*, dan berikut ini adalah formula lengkap dari operasi *projection*:

$\pi_p (\mathbf{R})$
----------------------

Untuk memahami penggunaan *query* jenis ini, perhatikan contoh-contoh berikut:

1. Menampilkan kolom NIM, nama, dan nilai\_alpro saja dari tabel MAHASISWA.

**Notasi:**

$$\pi_{\text{NIM}, \text{nama}, \text{nilai\_alpro}}(\text{MAHASISWA})$$

**Perintah SQL:**

```
select NIM, nama, nilai_alpro from MAHASISWA
```

2. Menampilkan kolom nama dan nilai\_apro untuk mahasiswa yang nilai mata kuliah alpronya lebih dari 80.

**Notasi:**

$$\pi_{\text{nama}, \text{nilai\_alpro}}(\sigma_{\text{nilai\_alpro} > 80}(\text{MAHASISWA}))$$

**Perintah SQL:**

```
select nama, nilai_alpro from MAHASISWA where nilai_alpro > 80
```

Untuk melihat ilustrasi hasil dari dua perintah *query* di atas, Anda dapat merujuk pada Gambar 89 dan 90.

**Keterangan:**

- **select NIM, nama, nilai\_alpro:** menampilkan 3 kolom yang masing-masing bernama NIM, nama, dan nilai\_alpro.
- **select nama, nilai\_alpro:** menampilkan 2 kolom yang masing-masing bernama nama dan nilai\_alpro.
- **nilai\_alpro > 80:** kondisi yang digunakan. Kondisi ini bisa disesuaikan dengan situasi yang dibutuhkan pada saat itu.
- **from MAHASISWA:** nama tabel yang akan digunakan. Pada bahasa *query*, nama tabel tidak perlu diapit dengan tanda kurung.
- **where:** klausa yang wajib ditulis apabila ada predikat atau kondisi yang harus dipenuhi pada *query*.

## Query Union

*Query union* digunakan untuk menggabungkan dua kolom dengan nama yang sama dari tabel yang berbeda. Query ini akan menghasilkan gabungan data dari dua kolom tabel yang disatukan. Apabila dihasilkan lebih dari 1 tuple yang

bernilai sama, maka yang digunakan hanya salah satunya saja. Penjelasan lebih detail terkait operasi *union* telah dibahas di subbab Operasi Aljabar Relasional pada bagian Operasi Union. Berikut ini merupakan notasi yang digunakan pada operasi *union*:

$$\pi_{p1}(R) \cup \pi_{p2}(R)$$

Perhatikan contoh-contoh berikut ini untuk dapat memahami query *union* dengan lebih mudah.

1. Menampilkan hasil gabungan antara dua kolom NIM pada tabel MHS dan IPK, menggunakan operasi UNION (tabel MHS dan IPK dapat dilihat pada Gambar 92).

**Notasi:**

$$\pi_{NIM}(MHS) \cup \pi_{NIM}(IPK)$$

**Perintah SQL:**

select NIM from MHS UNION select NIM from IPK

2. Sedangkan perintah SQL untuk menampilkan hasil gabungan antara dua kolom NIM pada tabel MHS dan IPK, tanpa harus menghapus nilai-nilai yang sama adalah:

select NIM from MHS UNION ALL select NIM from IPK

Ilustrasi hasil dari penggunaan dua *query* di atas dapat dilihat pada gambar 93 dan 94.

**Keterangan:**

- **select NIM from MHS:** memilih kolom NIM dari tabel MHS.
- **select NIM from IPK:** memilih kolom NIM dari tabel IPK.
- **UNION:** perintah SQL untuk menggabungkan dua tabel dengan mengeliminasi data-data redundan. Jika ada data yang sama, maka diambil salah satu saja.
- **UNION ALL:** perintah SQL untuk menggabungkan dua tabel berbeda secara apa adanya. Di sini, data redundan tidak akan dieleminasi.

## **Query Set Difference**

Pada dua tabel yang mana memiliki nilai yang bersinggungan, *Operasi Set Difference* digunakan untuk mencari nilai-nilai pada tabel A yang tidak ada pada tabel B. Operasi ini disimbolkan dengan tanda minus (-).

Untuk memahami penggunaan query terkait dengan operasi set difference, perhatikan contoh berikut ini:

1. Menampilkan NIM-NIM mahasiswa dari tabel MHS yang tidak ada pada tabel IPK.

**Notasi:**

$$\pi_{\text{NIM}}(\text{MHS}) - \pi_{\text{NIM}}(\text{IPK})$$

**Perintah SQL:**

```
select NIM from MHS where not exists (select NIM from IPK where
                                         MHS.NIM = IPK.NIM)
```

2. Menampilkan NIM-NIM mahasiswa pada tabel IPK yang tidak ada pada tabel MHS.

**Notasi:**

$$\pi_{\text{NIM}}(\text{IPK}) - \pi_{\text{NIM}}(\text{MHS})$$

**Perintah SQL:**

```
select NIM from IPK where not exists (select NIM from IPK where IPK.NIM
                                         = MHS.NIM)
```

Ilustrasi hasil dari penggunaan dua *query set difference* di atas dapat dilihat pada gambar 96 dan 97.

**Keterangan:**

- **select NIM from MHS:** memilih kolom NIM dari tabel MHS.
- **where not exists:** kondisi yang dimaksudkan, yaitu kondisi dimana data-data yang tidak tersedia pada tabel yang disebut setelahnya.
- **select NIM from IPK:** memilih kolom NIM dari tabel IPK.
- **where MHS.NIM = IPK.NIM:** kondisi yang mensyaratkan NIM yang sama antara tabel MHS dan IPK. Karena di bagian depan ada kondisi ‘where not exists’, maka yang akan muncul hanyalah yang berbeda saja.

## Query Intersection

Intersection digunakan untuk mencari nilai yang sama pada dua tabel berbeda. Masih menggunakan tabel MHS dan IPK, berikut ini adalah contoh penggunaan operasi sekaligus *query intersection*.

**Tujuan:** Menampilkan NIM-NIM mahasiswa yang muncul baik di tabel MHS maupun IPK.

**Notasi:**

$$\pi_{\text{NIM}}(\text{MHS}) \cap \pi_{\text{NIM}}(\text{IPK})$$

**Perintah SQL:**

```
select NIM from MHS where exist (select NIM from IPK where MHS.NIM =  
IPK.NIM)
```

**Keterangan:**

- **select NIM from MHS:** memilih kolom NIM dari tabel MHS.
- **where:** kondisi yang dimaksudkan, yaitu kondisi dimana data-data yang tersedia pada tabel yang disebut setelahnya.
- **select NIM from IPK:** memilih kolom NIM dari tabel IPK.
- **where MHS.NIM = IPK.NIM:** kondisi yang mensyaratkan NIM yang sama antara tabel MHS dan IPK. Karena di bagian depan ada kondisi ‘where exists’, maka yang akan muncul hanyalah data-data yang ada pada kedua tabel tersebut.

## BAB 7

# DESAIN BASIS DATA

---

### Bab ini membahas:

- Proses Perancangan Basis Data
- Pengembangan Sistem
- Contoh Aplikasi ER
- Bahasa Query Komersial

## Proses Perancangan Basis Data

Siklus hidup basis data (*database life cycle*) biasa disebut dengan *micro life cycle*. Sementara itu, siklus hidup sistem informasi (*information system life cycle*) disebut dengan *macro life cycle*. (Fundamentals of Database System (2011), Elmasri & Navathe, hal: 307)

Database DeMystified (2004), Andrew J. Oppel, hal: 129

Database Modeling and Design (2011), Tobey Teorey et al, MK, hal: 3

Database Design: Know it All (2009), Teorey et al, MK, hal: 2

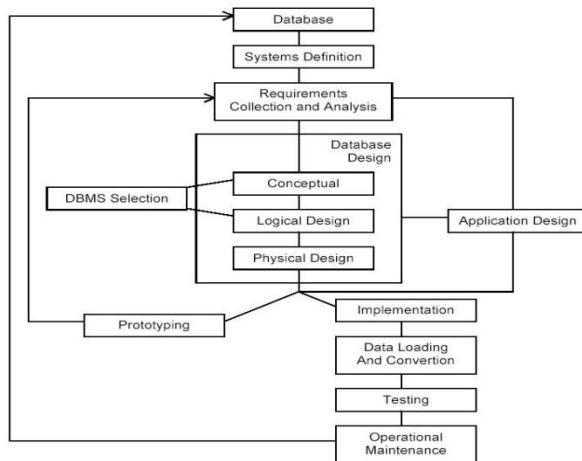
---

\*\*\*\*\*

---

Proses perancangan desain secara garis besar dibagi menjadi dua bagian utama, yaitu konseptual design dan physical design. Proses perancangan sistem database juga merupakan bagian dari siklus hidup database sebagai *micro lifecycle*. Suatu sistem database merupakan bagian dari komponen dasar sistem informasi organisasi yang lebih besar. Oleh karena itu siklus hidup sistem database berhubungan dengan siklus hidup

sistem informasi. Langkah-langkah siklus hidup aplikasi adalah berikut ini :



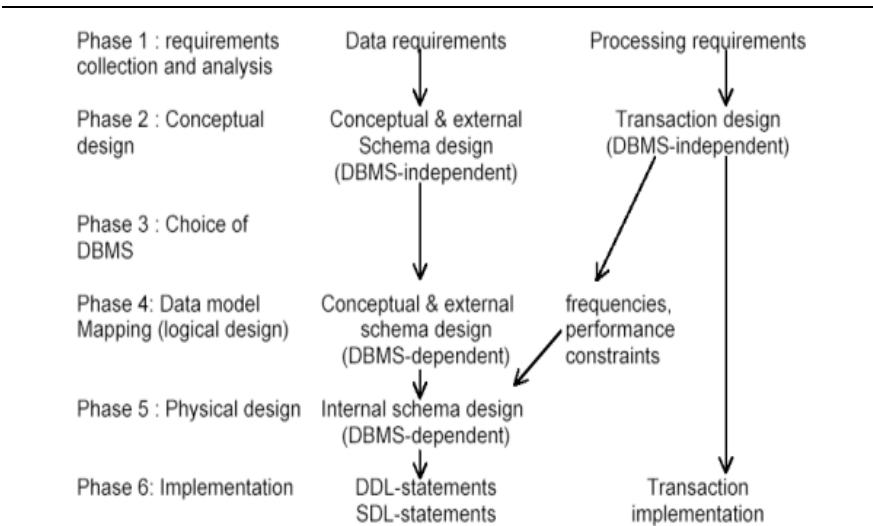
**Gambar 84**Gambar : Siklus Perancangan Database

Sedangkan tujuan dari perancangan database adalah sebagai berikut :

- untuk memenuhi informasi yang berisikan kebutuhan-kebutuhan user secara khusus dan aplikasi-aplikasinya.
- memudahkan pengertian struktur informasi.
- mendukung kebutuhan-kebutuhan pemrosesan dan beberapa obyek penampilan (response time, processing time, dan storage space).

Secara umum ada 6 fase dalam proses perancangan database, fase tersebut adalah :

1. Pengumpulan data dan analisis.
2. Perancangan database secara konseptual.
3. Pemilihan DBMS
4. Perancangan record database secara logika.
5. Perancangan database secara fisik.
6. Implementasi sistem database



Gambar : Fase Proses Perancangan Database

Untuk mempermudah proses perancangan database maka dapat digunakan pemodelan data sebagai sarana abstraksi data, dimana pemodelan data merupakan sebuah konsep untuk membuat deskripsi struktur basis data yang dapat digunakan untuk memuat spesifikasi dalam operasi dasar basis data. Banyak cara yang digunakan untuk mempresentasikan dalam memodelkan data. Secara umum dapat dikelompokan menjadi dua pemodelan data, yaitu :

1. *Object based logical model*. Pemodelan ini merupakan kelompok conceptual design, dimana dalam pemodelan ini struktur dari basis data diilustrasikan sebagai obyek. Model ini meliputi: 1) Model keterhubungan entitas (Entity Relationship Model atau ERD). 2) Model berorientasi object (Object-oriented Model). 3) Model Data Semantik(Semantic Data Model). 4) Model data Fungsional (Function Data Model).
2. *Record-based logical model*. Pemodelan ini merupakan kelompok physical design, dalam model ini struktur basis data diilustrasikan berdasarkan record. Model ini meliputi: 1) Model relational

---

(*Relational Model*). 2) Model Hierarkis (*Hierarchical Model*) 3) Model Jaringan (*Network Model*).

Pada proses perancangan ini kami berikan ilustrasi kasus pembentukan basis data dari proses transaksi jual beli pada sebuah toko. Transaksi pada sebuah toko ini melibatkan beberapa entity (sebagai table master) dan relasional ship (sebagai table transaksi). Penamaan tabel master dan table transaksi disini dapat dibedakan dengan kata benda untuk entity atau unit dan kata kerja untuk relationship atau hubungan.

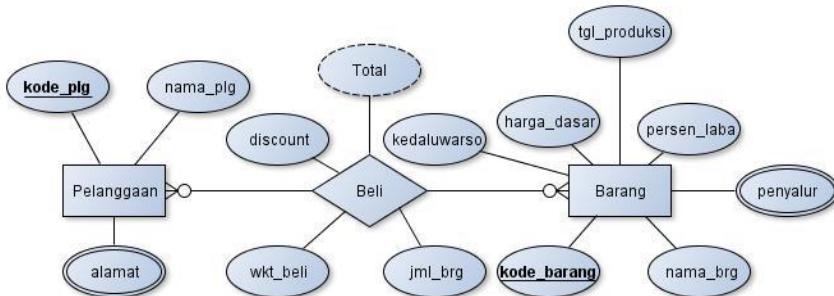
Misal diskripsi transaksi pada sebuah toko tersebut melibatkan obyek barang dan obyek pelanggan. Obyek pelanggan mempunya atribut kode pelanggan (selanjutnya disingkat kode\_plg), nama pelanggan (nama\_plg) dan alamat (alamat\_plg), kunci primernya adalah kode\_plg. Kunci primernya (dituliskan di dalam elips yang bergaris bawah) adalah kode\_plg. Jika dibuat schema sederhana adalah ***pelanggan(kode\_plg, nama\_plg, alamat\_plg)***. Sedangkan obyek barang mempunya atribut kode\_barang (kode\_brg), nama barang (nama\_brg), tipe barang (tipe\_brg), harga beli dari penyalur (harga\_dasar), persen laba (persen\_laba) dan penyalur. Kunci primernya adalah kode\_brg. Jika dibuat schema sederhana adalah ***barang(kode\_brg, nama\_brg, tipe\_brg, harga\_dasar, persen\_laba, penyalur)***.

Sedangkan transaksinya meliputi pembelian akan menjadi hubungan beli dengan atribut waktu beli suatu barang (wkt\_beli), jumlah barang tertentu yang dibeli (jml\_brg) bersifat field derivative dan discount dari harga suatu barang (discount). Di sini akan terbentuk kunci primer composite (gabungan dari kunci primer master). Kunci primer master yang dihubungkan pada transaksi beli disebut sebagai kunci tamu, yaitu dari kunci primer kode\_plg dan kunci primer kode\_brg. Jika dibuat schema ***beli(kode\_plg, kode\_brg, wkt\_beli, jml\_brg, discount)*** Namun kita harus ingat bahwa kunci primer gabungan (composite) harus selalu unik. Pada penerapannya misalnya terjadi seorang pelanggan dengan kode\_pelanggan P-M101 membeli suatu barang dengan kode\_barang B-T101 pada tanggal 2-3-2018. Kemudian pada tanggal 3-3-2018 si pelanggan tadi membeli barang yang sama, maka akan terjadi redudansi (penulisan data yang berulang) sehingga informasi ini akan ditolak oleh

sistem basis data, atau dianggap data sudah ada sehingga pembelian yang kedua yaitu pada tanggal 3-3-2003 tidak akan tercatat. Oleh karena itu diperlukan kunci primer tambahan yaitu wkt\_beli, sehingga unit hubungan akhirnya memiliki kunci primer gabungan tambahan (kode\_plg, kode\_brg, wkt\_beli).

### Konseptual Design

Jika dibuat pemodelan dalam Diagram ER dari sistem basis data Toko tersebut secara keseluruhan terlihat pada gambar E-R dibawah ini.



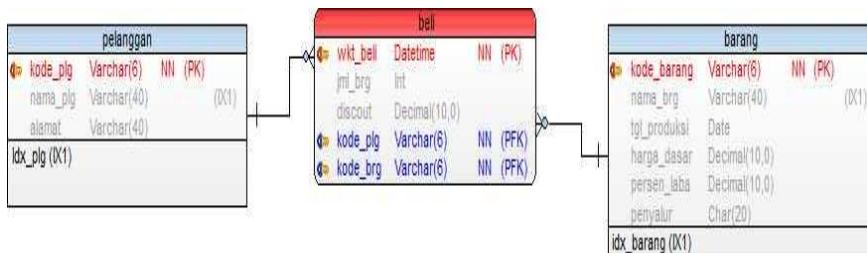
**Gambar : ER-Model Sistem Pembelian Barang**

---

### Skema Diagram

Sebelumnya kita perlu mengenal istilah relasi yang merupakan “embrio” dari table master jika sumbernya adalah entity sedangkan jika sumbernya adalah dari relationship maka akan jadi “embrio” table transaksi. Relasi dalam basis data berbentuk kotak yang mewakili suatu

himpunan unit, yang terdiri dari beberapa baris yang merupakan atribut-atribut dari himpunan unit tersebut. Jadi kalau suatu relasi mempunyai N baris berarti mempunyai N atribut. Atribut yang berwarna merah pada gambar skema diagram menggambarkan atribut kuncinya. Skema Diagram merupakan suatu sistem yang terdiri dari kumpulan dari beberapa relasi yang saling berhubungan. Di bawah ini contoh Diagram Skema dari ketiga buah relasi yaitu pelanggan, pembelian dan barang, seperti terlihat pada gambar dibawah ini.



Gambar : Skema Diagram **Sistem Pembelian Barang**

### Proses Normalisasi Pertama (1NF)

Proses normalisasi bagian dari konseptual desain, dilakukan pada perancangan konsep ER-D dengan transformasi pada Model Relasional. Data diuraikan dalam bentuk relasi model, selanjutnya dianalisis berdasarkan persyaratan tertentu ke beberapa tingkat normalisasi. Langkah atau proses ini diterapkan pada relasi-relasi yang terbentuk pada sistem basis data dengan Model Relasional. Sedangkan Model relasional adalah hasil dari mapping ER-Model. Tentunya mapping yang menghasilkan harus sampai pada tahap 3NF. Hal tersebut untuk mencapai hasil relasi yang konsisten dan sudah tidak akan terjadi anomali pada saat proses transaksi

(insert, update, delete). Contoh kasus : Misalkan dilakukan analisa hubungan relasi pada gambar ER-Model diatas :

1. Atribut alamat pada entitas pelanggan merupakan composite.  
**(Hasil analisa hubungan :** Terdapat fakta bahwa pelanggan bisa memiliki lebih dari satu alamat, artinya minimal harus punya satu alamat bisa juga banyak. Sehingga terjadi hubungan satu ke banyak dan mandatory sebagai partisipasi total).
2. Atribut penyalur barang pada entitas barang juga composite.  
**(Hasil analisa hubungan :** Terdapat fakta satu jenis barang yang sama dapat dipasok oleh lebih dari satu supplier terhadap barang tersebut. Supplier juga dapat memasuk lebih dari satu jenis barang yang berbeda. Sehingga terjadi hubungan banyak ke banyak dan mandatory sebagai partisipasi total).

Dari contoh analisa diatas maka table tersebut akan dihasilkan table yang belum memenuhi bentuk normal pertama (syarat utama 1stNF adalah semua attribute dari semua relasi sudah dalam bentuk atomic atau menghilangkan duplikasi kolom dari tabel yang sama/repetisi), seperti pada gambar dibawah ini :

Tabel : Tabel Tidak Normal (table pelanggan memiliki atribut composite alamat )

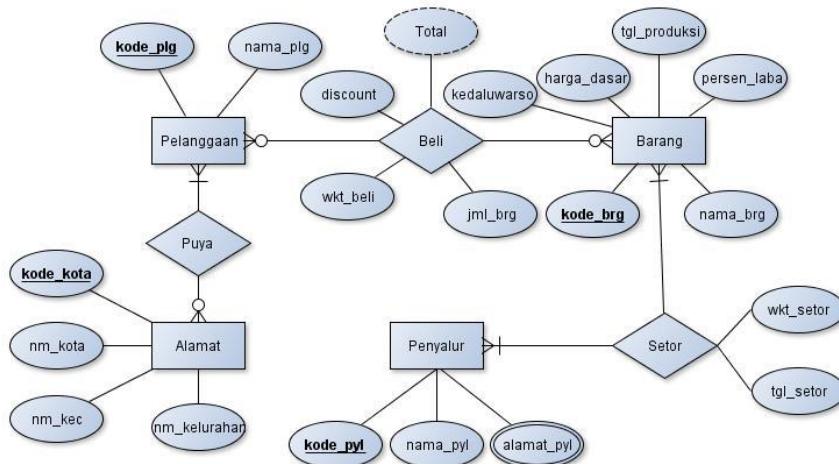
kode_plg	nama_plg	Alamat1	alamat2	alamat ...
P-M-301	Brindys	Banyumanik I/V	Perum BSB I No.3	Karangpaing
P-M-302	Cycal	Karangrejo B-2		
...				

Tabel : Tabel Tidak Normal (table pelanggan memiliki atribut composite alamat )

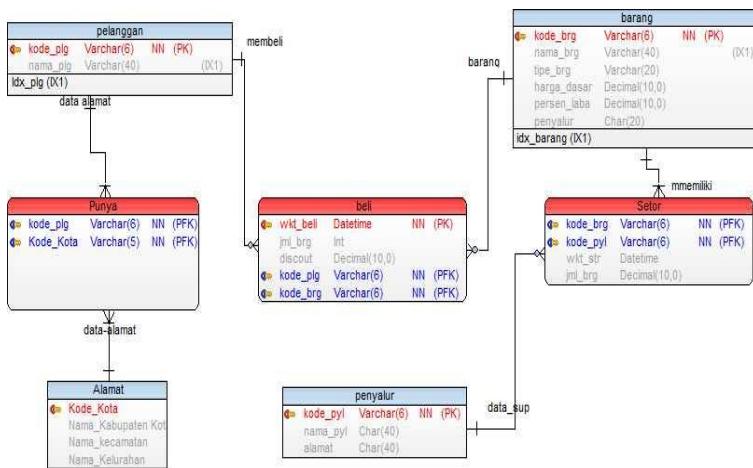
k o d e _ b r g	n a m a _ b r g	T g l _ p r o d u k s i	K e d a l u w a r s o	h a r g a _ d a s a r	p u n g _ l i h	P e n y a l u r 1	P e n y a l u r 2	P e n y a l u r 3
B - M - 0 0 1	Sabun Lux	27/09/2017	A k h i r 2 0 1 9	2 . 0 0 0	5	CV. ABC	Cv. Indomaxi	Cv. UniLife
...	...	...						

Jika dianalisa penyebab ketidak normalan (1NF) adalah atribut alamat (pada entitas pelanggan), maka cara melakukan perbaikannya adalah dengan merubah atribut alamat (entitas pelanggan) dan atribut penyalur (entitas barang) menjadi entitas alamat dan entitas penyalur. Sehingga dibutuhkan relasi “punya” sebagai transaksi antara entitas pelanggan dan entitas alamat (agar tidak kehilangan informasi alamat). Begitu juga antara entitas barang dan entitas penyalur dibutuhkan relasi “penyalur” sebagai transaksi diantar 2 entitas master tersebut (agar tidak kehilangan informasi penyalur). Proses ini merupakan konsep decomposisi, tetapi proses dekomposisi tidak boleh kehilangan informasi (lossless decomposition). Sehingga tidak ada atribut multi value yang menjadi biang dari atribut repetisi (berulang) pada model relasi. Hasil dari dekomposisi atribut alamat menjadi entitas alamat seperti pada ER-Diagram dibawah ini.

Gambar : ER-Digram Setelah Decomposisi Alamat



Gambar : Er-Diagram setelah di decomposisi atribut penyalur barang dan atribut alamat pelanggan (dalam 1stNF)



Gambar : Skema Diagram setelah di decomposisi atribut penyalur barang dan atribut alamat pelanggan (dalam 1stNF) (dalam 1stNF)

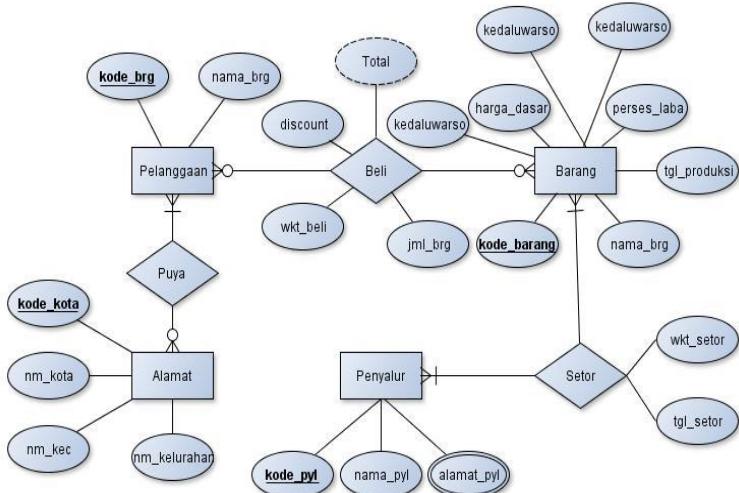
Dari gambar ER-Diagram dan Skema Diagram tersebut sudah tampak memenuhi syarat normal pertama (1stNF), yaitu semua atribut semua atribut dalam bentuk atomic.

### Proses Normalisasi Bentuk Kedua (2NF)

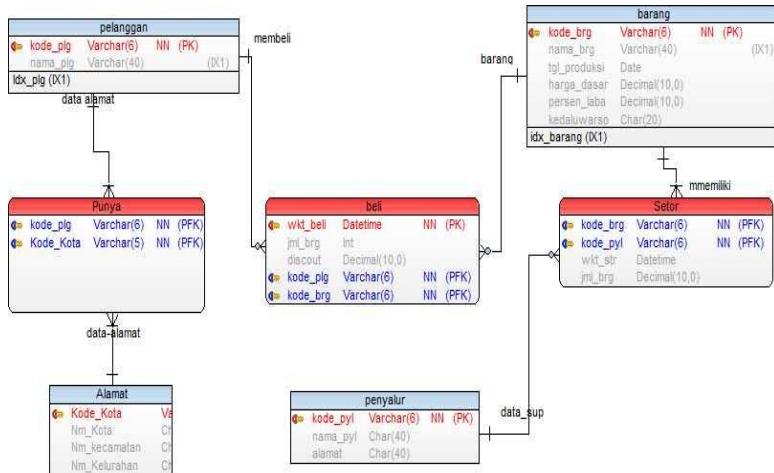
Bentuk normal kedua (2NF) tercapai jika sistem basis data sudah dalam bentuk normal pertama (1NF) dan tidak ada ketergantungan parsial. Ketergantungan prasial merupakan konsekuensi dari relasi yang memiliki kunci primer komposit. Kondisi normal bentuk kedua menuntut semua atribut bukan kunci harus tergantung penuh pada kunci primer gabungan. Dengan kata lain untuk memenuhi 2NF maka seluruh atribut bukan kunci pada setiap relasi hanya tergantung pada kunci primer gabungan saja,

bukan tergantung pada sebagian kunci primer saja (jika ada atribut bukan kunci hanya tergantung penuh pada sebagian kunci primer komposit maka disebut sebagai ketergantungan parsial).

Jika di analisa lebih lanjut maka kunci primer komposit biasanya ada pada relasi-relasi transaksi. Misal hasil analisa (dari transaksi diatas) pada gambar ER-Model dan Skema model diatas, *discount* pada relasi “beli”. Atribut *discount* eksistensinya tidak tergantung penuh pada seluruh kunci primer gabungan (kode\_plg, kode\_brg, wkt\_beli), namun eksistensinya hanya tergantung pada sebagian kunci primer yaitu (kode\_brg). Hasil analisa tersebut menunjukkan atribut *discount* hanya tergantung pada sebagian kunci primer (kode\_brg), fakta diskon tidak tergantung pada siapa pelanggannya (*kode\_plg*) atau kapan waktu pembelian barang tersebut (*wkt\_beli*). Hal ini menunjukkan relasi beli terdapat atribut tertentu (*discount*) yang memiliki ketergantungan parsial. Fakta ini menunjukkan relasi tersebut belum dalam bentuk normal ke dua (2NF) karena ada atribut *discount* yang eksistensinya hanya tergantung pada sebagian kunci primer composit. Permasalahan disini karena faktanya diskon tidak pandang bulu siapa pelanggannya atau kapan waktu pembeliannya, maka *discount* bisa didekomposisi menjadi entitas baru misalnya entitas diskon atau atribut tersebut dipindah pada entitas barang. Diperlukan analisa terhadap pilihan perlakuan terhadap atribut *discount* tersebut. Tentunya alternatif yang dipilih adalah yang paling efisien dan tidak kehilangan informasi nilai dikon terhadap barang. Jika eksistensi *discount* sebagai atribut bukan kunci hanya tergantung pada kode\_brg (bagian dari kunci primer) saja maka atribut *discount* yang semula ada pada relasi beli maka atrinbut tersebut dapat di pindah pada relasi barang.



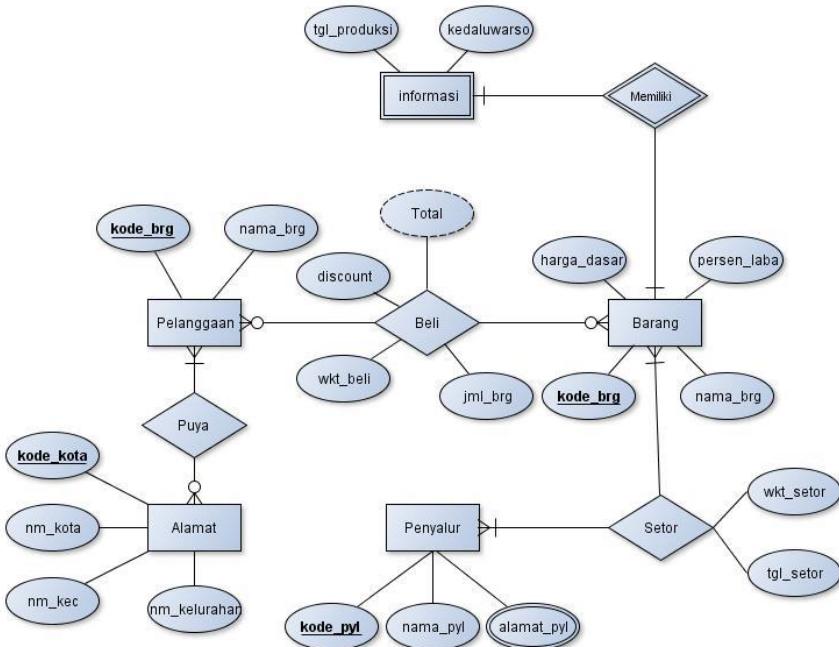
Gambar: ER-Diagram Dalam Bentuk 2NF



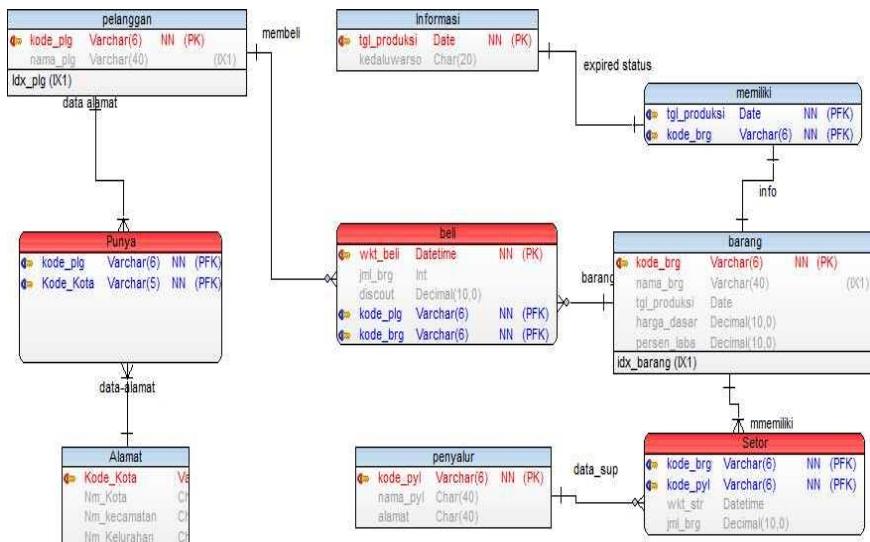
Gambar: Skema Diagram Memenuhi Bentuk 2NF

## Proses Normalisasi Bentuk Kedua (3NF)

Syarat bentuk 3NF adalah sudah dalam bentuk 2NF dan semua atribut bukan kunci tergantung secara langsung pada kunci primernya tanpa melalui atribut non kunci utama, hal ini berarti tidak ada ketergantungan transitif. Hasil analisis pada skema diatas. Terjadi ketergantungan keterangan kedaluwarsa (expired) dari atribut yang ada di entitas barang. Dimana kedaluwarsa tidak dapat ditentukan oleh kode\_brg sebagai kunci utama tetapi tergantung pada atribut bukan kunci yang ada pada entitas barang yaitu atribut tgl\_produksi barang. Dengan demikian dilakukan dekomposisi pada kedua atribut tersebut menjadi entitas informasi barang dan dibuatkan relasi memiliki (informasi expired).

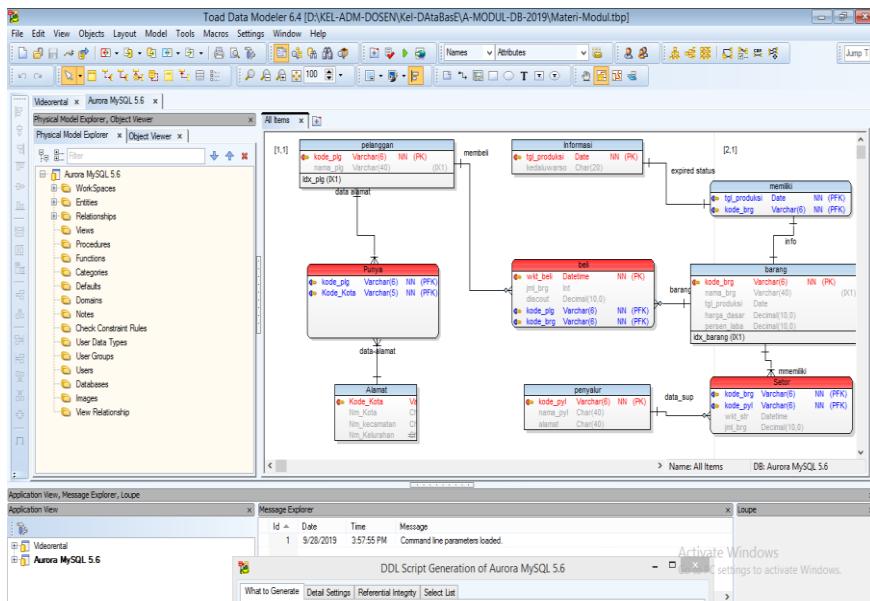


Gambar ER-Diagram Bentuk 3NF

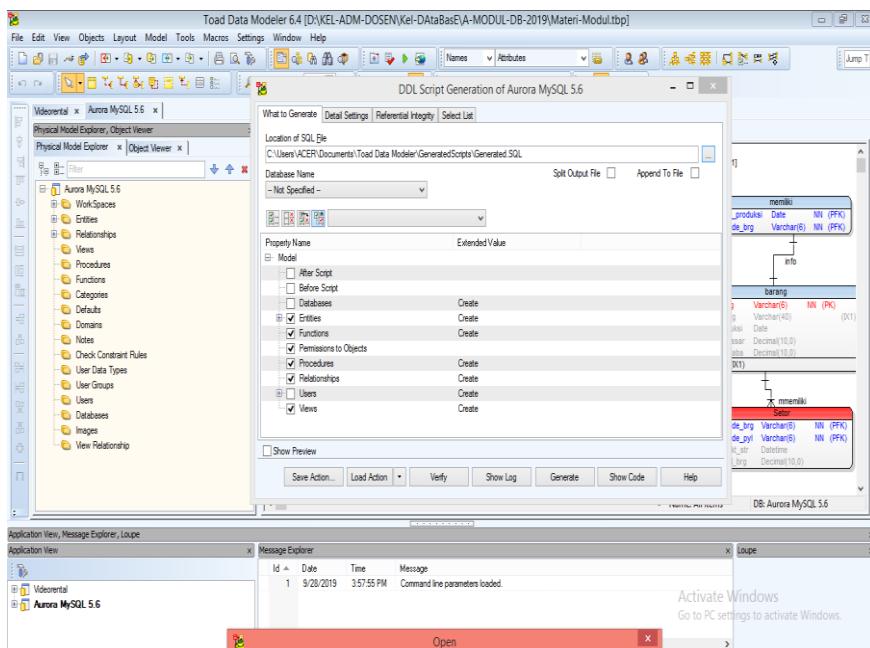


Gambar : Skema Diagaram setalah dekomposisi alamat dan penyalur  
(dalam 3NF)

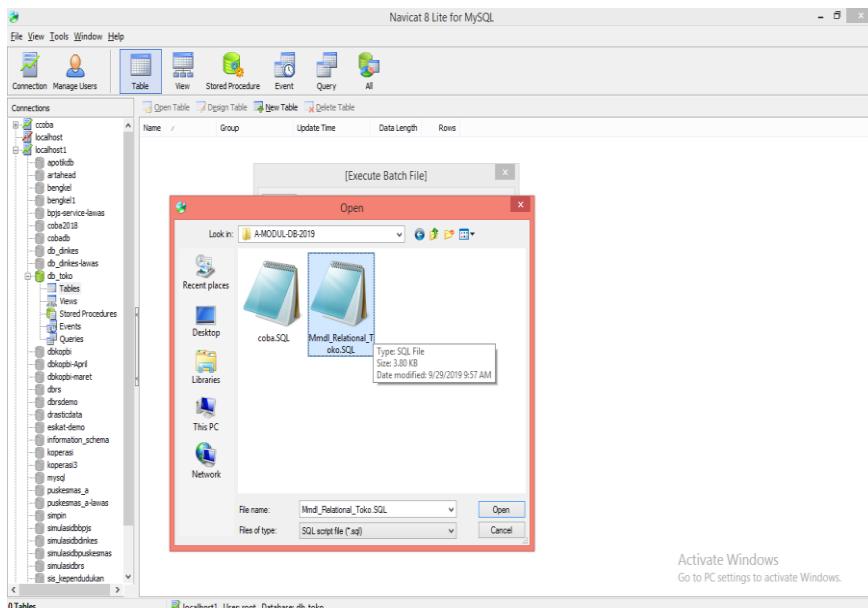
## Transformasi Conceptual Desain Menjadi Physical Design



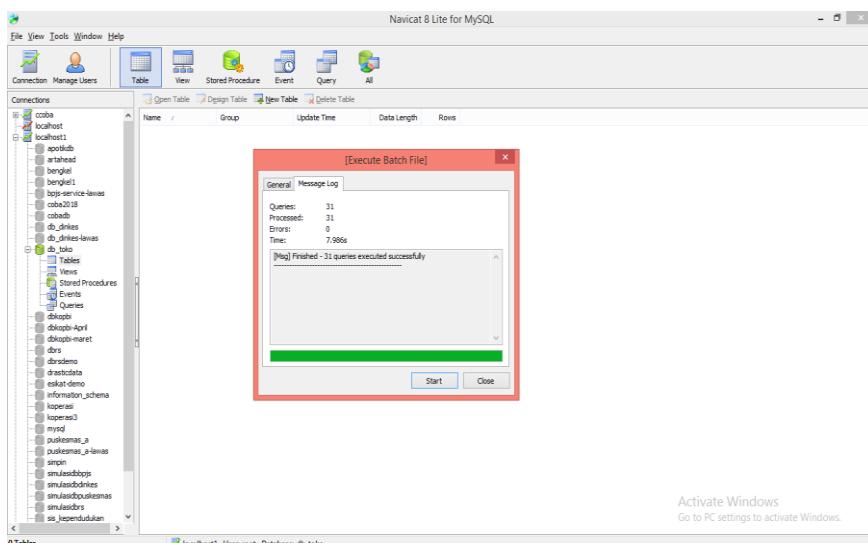
Gambar : Melakukan Generate Skema Model (3NF) Ke Script SQL



Gambar : Mengatur Properti Untuk File Script SQL



Gambar : Mengekstrak File Srip SQL Pada DBMS tertentu



Gambar : Proses Extraksi File Script SQL Menjadi Tabel

```
/*
Created: 9/22/2019
Modified: 9/28/2019
Model: Aurora MySQL 5.6
Database: Aurora MySQL 5.6
*/
-- Create tables section -----
-- Table pelanggan
CREATE TABLE `pelanggan`
(
  `kode_plg` Varchar(6) NOT NULL,
  `nama_plg` Varchar(40)
);
CREATE INDEX `idx_plg` ON `pelanggan`(`nama_plg`)
;
ALTER TABLE `pelanggan` ADD PRIMARY KEY (`kode_plg`)
;

-- Table barang
CREATE TABLE `barang`
(
  `kode_brg` Varchar(6) NOT NULL,
  `nama_brg` Varchar(40),
  `tgl_produksi` Date,
  `harga_dasar` Decimal(10,0),
  `persen_laba` Decimal(10,0)
```

```
)  
;  
CREATE UNIQUE INDEX `idx_barang` ON `barang`(`nama_brg`)  
;  
ALTER TABLE `barang` ADD PRIMARY KEY (`kode_brg`)  
;  
  
-- Table beli  
CREATE TABLE `beli`  
(  
`wkt_beli` Datetime NOT NULL,  
`jml_brg` Int DEFAULT 7,  
`discout` Decimal(10,0),  
`kode_plg` Varchar(6) NOT NULL,  
`kode_brg` Varchar(6) NOT NULL  
)  
;  
ALTER TABLE `beli` ADD PRIMARY KEY (`kode_plg`,`kode_brg`,`wkt_beli`)  
;  
-- Table Punya  
CREATE TABLE `Punya`  
(  
`kode_plg` Varchar(6) NOT NULL,  
`Kode_Kota` Varchar(5) NOT NULL  
)  
;  
ALTER TABLE `Punya` ADD PRIMARY KEY (`kode_plg`,`Kode_Kota`)  
;  
-- Table Alamat  
CREATE TABLE `Alamat`  
(  
`Kode_Kota` Varchar(5) NOT NULL,  
`Nm_Kota` Char(20),  
`Nm_kecamatan` Char(20),  
`Nm_Kelurahan` Char(20)
```

```
)  
;  
ALTER TABLE `Alamat` ADD PRIMARY KEY (`Kode_Kota`)  
;  
-- Table Setor  
CREATE TABLE `Setor`  
(  
`kode_brg` Varchar(6) NOT NULL,  
`kode_pyf` Varchar(6) NOT NULL,  
`wkt_str` Datetime,  
`jml_brg` Decimal(10,0)  
)  
;  
ALTER TABLE `Setor` ADD PRIMARY KEY (`kode_brg`,`kode_pyf`)  
;  
-- Table penyalur  
CREATE TABLE `penyalur`  
(  
`kode_pyf` Varchar(6) NOT NULL,  
`nama_pyf` Char(40),  
`alamat` Char(40)  
)  
;  
ALTER TABLE `penyalur` ADD PRIMARY KEY (`kode_pyf`)  
;  
-- Table Informasi  
CREATE TABLE `Informasi`  
(  
`tgl_produksi` Date NOT NULL,  
`kedaluwarsa` Char(20)  
)  
;  
ALTER TABLE `Informasi` ADD PRIMARY KEY (`tgl_produksi`)
```

```

;

-- Table memiliki

CREATE TABLE `memiliki`
(
`tgl_produksi` Date NOT NULL,
`kode_brg` Varchar(6) NOT NULL
)
;

ALTER TABLE `memiliki` ADD PRIMARY KEY (`tgl_produksi`,`kode_brg`)
;

-- Create foreign keys (relationships) section -----
ALTER TABLE `beli` ADD CONSTRAINT `Relationship1` FOREIGN KEY (`kode_plg`) REFERENCES `pelanggan`(`kode_plg`) ON DELETE RESTRICT ON UPDATE RESTRICT
;

ALTER TABLE `beli` ADD CONSTRAINT `Relationship2` FOREIGN KEY (`kode_brg`) REFERENCES `barang`(`kode_brg`) ON DELETE RESTRICT ON UPDATE RESTRICT
;

ALTER TABLE `beli` ADD CONSTRAINT `membeli` FOREIGN KEY (`kode_plg`) REFERENCES `pelanggan`(`kode_plg`) ON DELETE NO ACTION ON UPDATE CASCADE
;

ALTER TABLE `beli` ADD CONSTRAINT `barang` FOREIGN KEY (`kode_brg`) REFERENCES `barang`(`kode_brg`) ON DELETE NO ACTION ON UPDATE CASCADE
;

ALTER TABLE `Punya` ADD CONSTRAINT `data alamat` FOREIGN KEY (`kode_plg`) REFERENCES `pelanggan`(`kode_plg`) ON DELETE RESTRICT ON UPDATE RESTRICT
;

ALTER TABLE `Punya` ADD CONSTRAINT `data-alamat` FOREIGN KEY (`Kode_Kota`) REFERENCES `Alamat`(`Kode_Kota`) ON DELETE RESTRICT ON UPDATE RESTRICT
;

ALTER TABLE `Setor` ADD CONSTRAINT `mmemiliki` FOREIGN KEY (`kode_brg`) REFERENCES `barang`(`kode_brg`) ON DELETE RESTRICT ON UPDATE RESTRICT
;

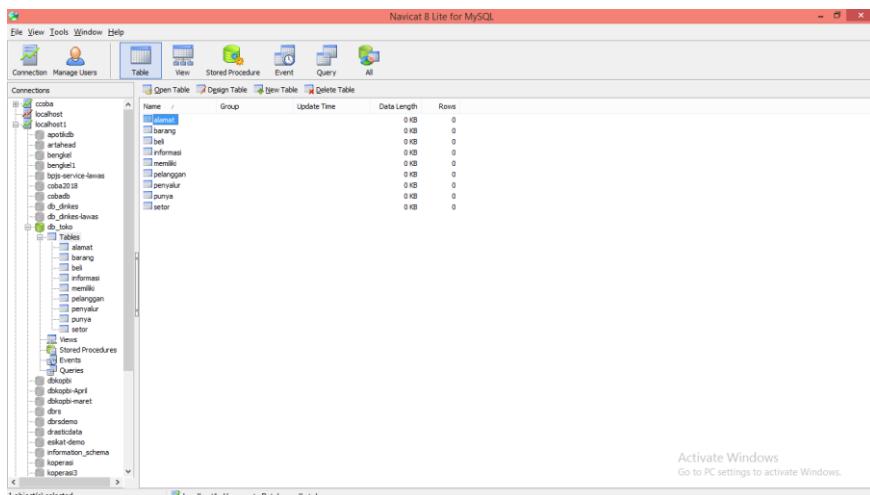
ALTER TABLE `Setor` ADD CONSTRAINT `data_sup` FOREIGN KEY (`kode_pyf`) REFERENCES `penyalur`(`kode_pyf`) ON DELETE RESTRICT ON UPDATE RESTRICT
;

ALTER TABLE `memiliki` ADD CONSTRAINT `expired status` FOREIGN KEY (`tgl_produksi`) REFERENCES `Informati`(`tgl_produksi`) ON DELETE RESTRICT ON UPDATE RESTRICT
;

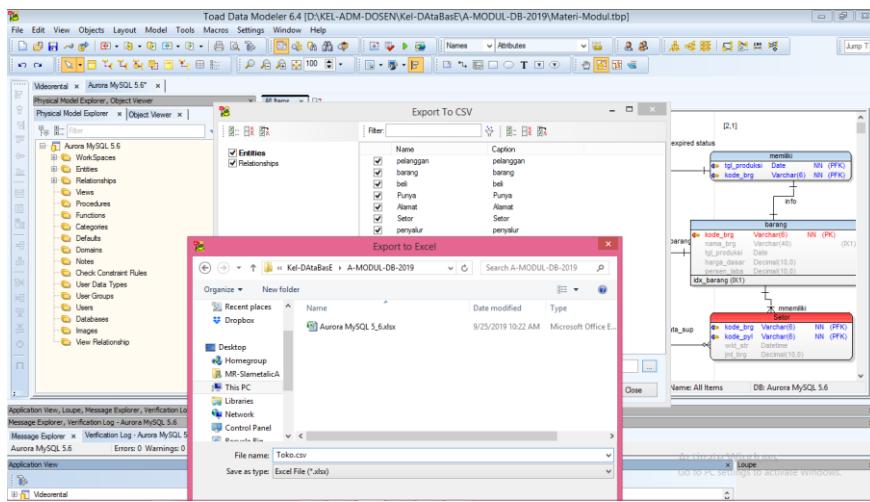
```

```
ALTER TABLE `memiliki` ADD CONSTRAINT `info` FOREIGN KEY (`kode_brg`) REFERENCES `barang`  
(`kode_brg`) ON DELETE RESTRICT ON UPDATE RESTRICT  
;
```

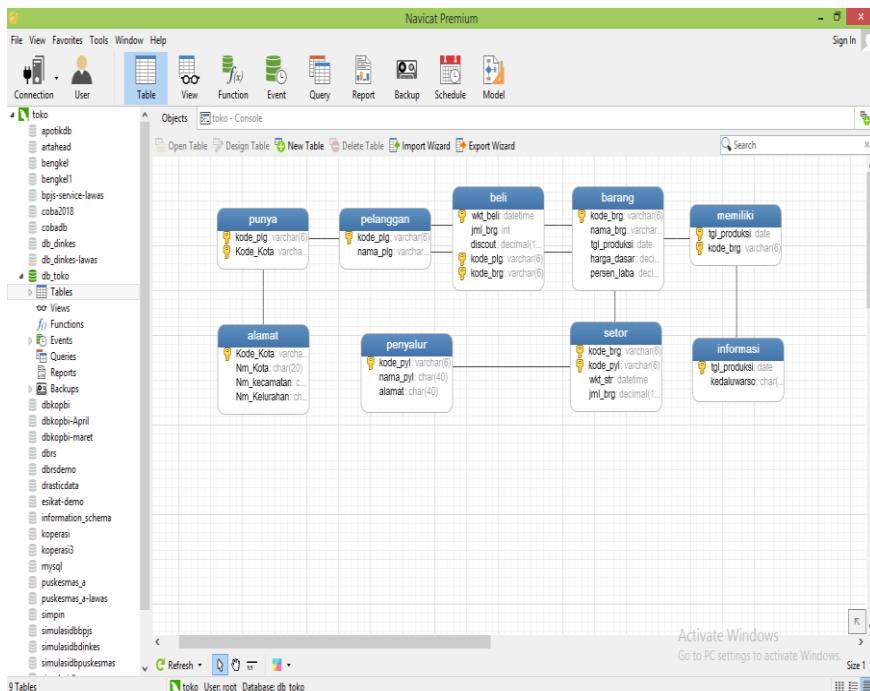
Gambar : Isi File Script SQL Berupa DDL



Gambar : Hasil Ekstrak File Script SQL



Gambar : Skema Model Juga Dapat Di Ekstrak Dalam CSV



Gambar : Skema Tabel Yang Berelasi

## Bahasa Query Komersial

Dalam penggunaan bahasa Query komersial (DDL, DML, DCL), jika kita menggunakan tools dalam perancangan database maka focus utamanya ada pada DML, karena DDL dan DCL dapat dihasilkan dari menggenerate dari model ER-Diagram sebagai conceptual desain ke schema model dan relation model yang bersifat physical desain. Dalam melakukan query jika konseptual desain dilakukan dengan benar maka dapat melakukan query multi table sebagai manifestasi retrival bahwa informasi yang didapatkan berasal dari kumpulan table yang berelasi dengan minimal dalam bentuk normal ke 3 (3NF). Bentuk ini sudah dapat menghindari anomaly sehingga database akan dijaga konsistensinya. Contoh dalam pencarian informasi dibawah ini, mencari informasi yang memiliki criteria pencarian : tampilkan nama barang lengkap dengan nama supplier barang, tanggal produksi barang, nama pelanggan, alamat

lengkap pelanggan. Informasi yang akan dihasilkan memiliki sumber data dari 9 tabel atau relasi sehingga akan melibatkan 9 relasi dalam proses pencarinya.

```

SELECT B.nama_brg AS NAMA_BARANG,P.nama_pyl AS NAMA_SUPLIER,I.tgl_produksi AS "TANGGAL di PRODUKSI", G.nama_plg AS NAMA_PELANGGAN, CONCAT(T.nm_kelurahan," ", "Kecamatan",",",T.nm_Kecamatan," ", "Nama Kota" " ", T.nm_kota) AS Alamat_Lengkap_Pelanggan
FROM barang as B, penyalur as P, informasi as I, pelanggan as G, setor as S, memiliki as M, beli as L, punya as U, alamat as T
WHERE B.kode_brg = S.kode_brg and S.kode_pyl = P.kode_pyl and B.kode_brg = M.kode_brg and M.tgl_produksi= I.tgl_produksi and B.kode_brg=L.kode_brg and L.kode_plg=G.kode_plg and G.kode_plg=U.kode_plg and U.kode_kota = T.kode_kota ;

```

Gambar : Perintah Query Multi Tabel

	NAMA_BARANG	NAMA_SUPLIER	TANGGAL di PRODUKSI	NAMA_PELANGGAN	Alamat_Lengkap_Pelanggan
1	Sampo XYZ	CV. Aneka Utama	2016-10-01	Hakik Kuera P	Kebondalem, KecamatanPurowadi Tengah, Nama Kota, Purwodadi
2	Sampo XYZ	CV. Aneka Utama	2016-10-01	Hakik Kuera P	Kebondalem, KecamatanPurowadi Tengah, Nama Kota, Purwodadi
3	Sampo XYZ	PT.Lini Degan	2016-10-01	Hakik Kuera P	Kebondalem, KecamatanPurowadi Tengah, Nama Kota, Purwodadi
4	Sabun ABC	CV. Trans Utama	2017-10-01	Hakik Kuera P	Kebondalem, KecamatanPurowadi Tengah, Nama Kota, Purwodadi
5	Sampo XYZ	CV. Aneka Utama	2016-10-01	Hakik Kuera P	Nglordan Etan, KecamatanGilingan Nama Kota Solo
6	Sampo XYZ	CV. Aneka Utama	2016-10-01	Pavina Jenar P	Nglordan Etan, KecamatanGilingan Nama Kota Solo
7	Sampo XYZ	CV. Trans Utama	2016-10-01	Hakik Kuera P	Nglordan Etan, KecamatanGilingan Nama Kota Solo
8	Sampo XYZ	CV. Trans Utama	2016-10-01	Pavina Jenar P	Nglordan Etan, KecamatanGilingan Nama Kota Solo
9	Sampo XYZ	PT.Lini Degan	2016-10-01	Hakik Kuera P	Nglordan Etan, KecamatanGilingan Nama Kota Solo
10	Sampo XYZ	PT.Lini Degan	2017-10-01	Pavina Jenar P	Nglordan Etan, KecamatanGilingan Nama Kota Solo
11	Sabun ABC	CV. Trans Utama	2017-10-01	Hakik Kuera P	Nglordan Etan, KecamatanGilingan Nama Kota Solo
12	Sabun ABC	CV. Trans Utama	2018-10-01	Pavina Jenar P	Nglordan Etan, KecamatanGilingan Nama Kota Solo
13	Odei ABC	CV. Trans Utama	2018-10-01	Pavina Jenar P	Nglordan Etan, KecamatanGilingan Nama Kota Solo
14	Odei ABC	PT. Lini Degan	2018-10-01	Pavina Jenar P	Nglordan Etan, KecamatanGilingan Nama Kota Solo
15	Sampo XYZ	CV. Aneka Utama	2016-10-01	Hakik Kuera P	Demangan, KecamatanGondo Kusuman, Nama Kota, Jogjakarta
16	Sampo XYZ	CV. Trans Utama	2016-10-01	Hakik Kuera P	Demangan, KecamatanGondo Kusuman, Nama Kota, Jogjakarta
17	Sampo XYZ	PT. Lini Degan	2016-10-01	Hakik Kuera P	Demangan, KecamatanGondo Kusuman, Nama Kota, Jogjakarta
18	Sabun ABC	CV. Trans Utama	2017-10-01	Hakik Kuera P	Demangan, KecamatanGondo Kusuman, Nama Kota, Jogjakarta

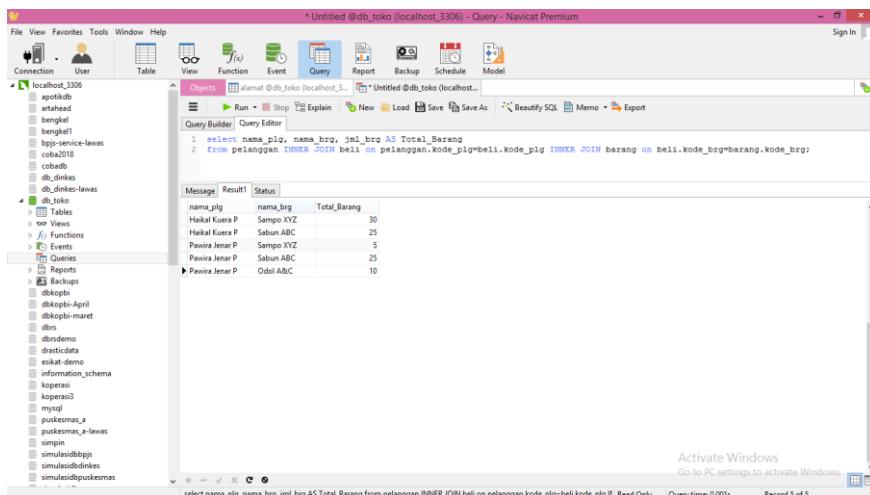
Gambar : Hasil dari Query Multi Tabel

Dalam melakukan relasi data pencarian, juga dapat dilakukan dengan menggunakan perintak INNER JOIN. Misal dicari Nama Pelanggan memiliki barang apa saja dan jumlahnya berapa saja.

Select nama\_plg, nama\_brg, jml\_brg AS Total\_Barang

From pelanggan INNER JOIN beli on pelanggan.kode\_plg=beli.kode\_plg  
INNER JOIN barang on beli.kode\_brg=barang.kode\_brg;

Gambar : Mencari informasi dari beberapa tabel dengan Inner Join



The screenshot shows the Navicat Premium interface with a database connection named 'localhost\_3306'. In the center, there's a 'Query Editor' window displaying the following SQL code:

```
1 select nama_plg, nama_brg, jml_brg AS Total_Barang
2 from pelanggan INNER JOIN beli on pelanggan.kode_plg=beli.kode_plg
   INNER JOIN barang on beli.kode_brg=barang.kode_brg;
```

Below the code, the results are displayed in a table:

nama_plg	nama_brg	Total_Barang
Halal Kue P	Sampo XYZ	30
Halal Kue P	Sabun ABC	25
Pawira Jerni P	Sampo XYZ	5
Pawira Jerni P	Sabun ABC	25
Pawira Jerni P	Odeon ABC	10

Gambar : Perintah query antara relasi melibatkan Inner Join

Dalam SQL Query juga dapat menggunakan beberapa fungsi matematika. Misal ingin mengetahui total barang yang dibeli oleh masing-masing pelanggan menurut jenis barang.

Jika perintah SQL diatas akan di permanenkan maka dapat dibuat view sehingga jika ingin mengeksekusi sejumlah perintah tinggal memanggil nama view nya saja. Biasanya view ini digunakan untuk membantu membuat laporan, misalnya sejumlah perintah query tersebut diberi nama ReportData\_Toko (Create View ReportData\_toko). Perintah lengkapnya seperti gambar dibawah ini.

```
CREATE VIEW ReportData_Toko AS
```

```
SELECT B.nama_brg AS NAMA_BARANG,P.nama_pyl AS  
NAMA_SUPLIER,I.tgl_produksi AS "TANGGAL di PRODUKSI",  
G.nama_plg AS NAMA_PELANGGAN, CONCAT(T.nm_kelurahan," "  
"Kecamatan",T.nm_Kecamatan," " "Nama Kota" " ",T.nm_kota) AS  
Alamat_Lengkap_Pelanggan
```

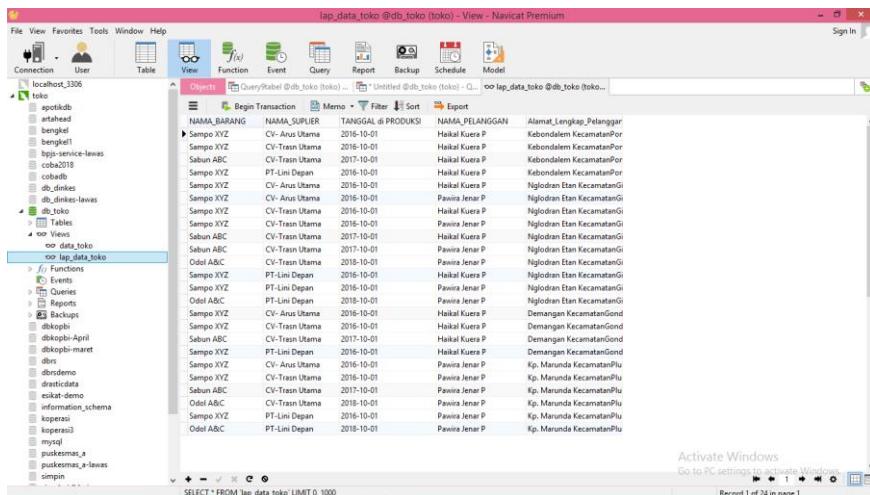
```
FROM barang as B, penyalur as P, informasi as I, pelanggan as G, setor as S,  
memiliki as M, beli as L, punya as U, alamat as T
```

```
WHERE B.kode_brg = S.kode_brg and S.kode_pyl = P.kode_pyl and  
B.kode_brg = M.kode_brg and M.tgl_produksi= I.tgl_produksi and  
B.kode_brg=L.kode_brg and L.kode_plg=G.kode_plg and  
G.kode_plg=U.kode_plg and U.kode_kota = T.kode_kota ;
```

Gambar : Perintah Membuat View

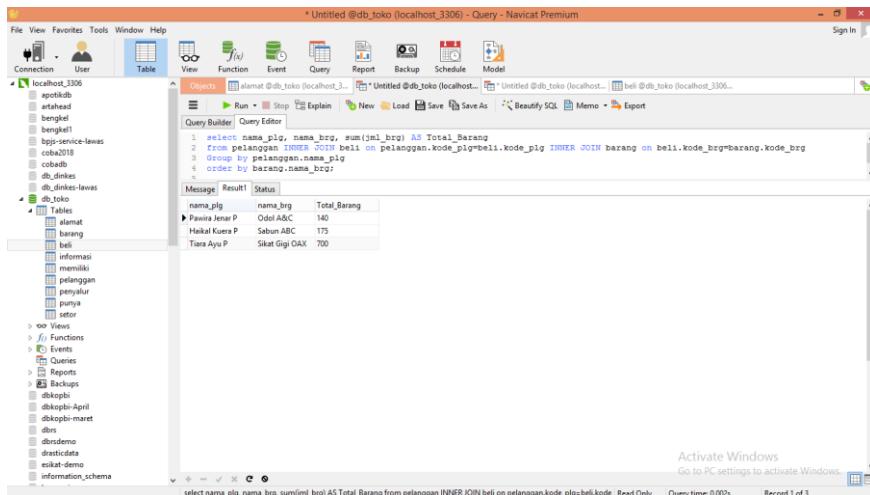
Sehingga cara mengeksekusinya tinggal menjalankan nama viewnya (misalnya : select \* from ReportData\_Toko).

Jika kita menggunakan editor pengelola database maka akan muncul nama view yang tinggal di double klik, maka hasilnya akan tampil sebagaimana dibawah ini.



Gambar : Menjalankan View

Nama Pelanggan, Jumlah barang yang dibeli oleh pelanggan dan ....



# BAB 8

## NORMALISASI DATA DAN KETERGANTUNGAN FUNGSIONAL

---

### Bab ini membahas:

- Pengertian dan Tujuan Normalisasi
  - Pentingnya Normalisasi
  - Tahapan Normalisasi
  - Closure dan Ketergantungan Fungsional
  - Anomali dan Dependensi
  - Diagram Dependensi Fungsional
  - Dekomposisi Tak Hilang
  - Contoh Kasus Bentuk Normal dan Tidak Normal
- 

### Pengertian dan Tujuan Normalisasi

Normalisasi sederhananya merupakan proses penyaringan struktur database dimana pengguna menghapus kumpulan data yang muncul berulang kali (*repeated*) dan menjadikannya satu atau lebih tabel yang terpisah. Normalisasi juga bisa didefinisikan sebagai proses identifikasi dan penghapusan (*elimination*) anomali pada tabel di database. Definisi yang lebih lengkap tentang normalisasi adalah proses mengurai tabel-tabel di database untuk menghindari redundansi data dan hal-hal yang tidak diharapkan, seperti anomali pada saat menambah, mengedit serta menghapus data. Secara khusus, anomali akan dibahas pada subbab X.XX.

---

Proses normalisasi dimulai dari satu atau lebih tabel universal (*star table*) atau tabel yang masih mentah dan apa adanya. Selanjutnya kita terapkan aturan-aturan untuk mengeliminasi anomali-anomali yang ada, sehingga akan muncul

sekumpulan tabel baru yang bebas dari anomali. Kumpulan aturan-aturan yang diterapkan tersebut, dinamakan *normal forms*.

Tujuan dilakukannya normalisasi pada tabel di database adalah sebagai berikut:

1. Untuk mengurangi redundansi data secara signifikan.
2. Untuk mengurangi kompleksitas.
3. Untuk mempermudah pemodifikasiannya data.

Ketiga poin di atas (redundansi data, kompleksitas dan untuk kemudahan dalam modifikasi data) berhubungan secara langsung dengan anomali pada data.

## Pentingnya Normalisasi

Tanpa normalisasi, atau jika suatu tabel tidak secara tepat dinormalkan serta masih memiliki redundansi data, maka suatu database tidak hanya akan memakan tempat yang besar pada harddisk, tapi juga menghadirkan kesulitan dalam menangani atau memperbarui data/tabel tanpa menimbulkan permasalahan baru, yaitu hilangnya data yang lain. Anomali pada saat proses penambahan data (*insertion*), perubahan data (*update*), dan penghapusan data (*deletion*) merupakan hal yang seringkali terjadi ketika database tidak dinormalisasi.

Selanjutnya, tanpa dilakukan normalisasi pada database, maka suatu rancangan database memungkinkan menjadi rancangan yang buruk. Rancangan database disebut buruk, yaitu jika:

- Data yang sama tersimpan di beberapa tempat sekaligus, seperti pada file, tabel, atau record.
- Database tidak dapat menghasilkan informasi tertentu, akibat adanya nilai-nilai yang ambigu (akibat redundansi).
- Terjadinya kehilangan informasi, yang diantaranya disebabkan oleh adanya anomali saat proses update maupun delete.
- Adanya redundansi atau pengulangan data. Hal ini dapat memakan ruang penyimpanan yang tidak perlu, dengan kata lain boros memori. Data yang *redundant* ini pada akhirnya juga memunculkan kesulitan ketika kita hendak melakukan *update* data.
- Munculnya *Null Value* (akan di bahas di bab terpisah).

## Jenis-Jenis Normalisasi

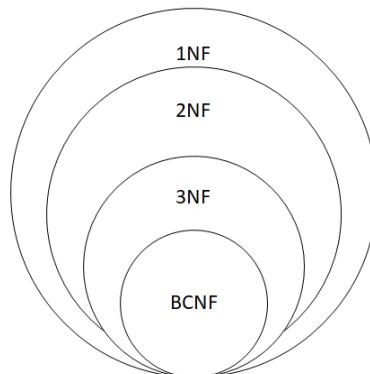
Dalam beberapa literatur disebutkan ada setidaknya 9 bentuk normal pada database, yaitu:

1. 1NF (1<sup>st</sup> Normal Form)
2. 2NF (2<sup>nd</sup> Normal Form)
3. 3NF (3<sup>rd</sup> Normal Form)
4. EKNF (Elementary Key Normal Form)
5. BCNF (Boyce-Codd Normal Form)
6. 4NF (4<sup>th</sup> Normal Form)
7. 5NF (5<sup>th</sup> Normal Form)
8. DKNF (Domain/Key Normal Form)
9. 6NF (6<sup>th</sup> Normal Form)

Namun dalam prakteknya, hanya ada 4 bentuk normal (*normal form*) yang popular dan seringkali digunakan, yaitu:

1. 1NF (1<sup>st</sup> Normal Form)
2. 2NF (2<sup>nd</sup> Normal Form)
3. 3NF (3<sup>rd</sup> Normal Form)
4. BCNF (Boyce-Codd Normal Form)

Pada gambar X.XX, terlihat bahwa BCNF memiliki lingkaran paling kecil. Artinya, BCNF memiliki aturan yang paling ketat dibanding yang lainnya. Maka untuk bisa mencapai BCNF, harus terlebih dahulu melewati fase-fase sebelumnya, yaitu 3NF, 2NF dan yang pertama harus dilalui terlebih dahulu yaitu 1NF.



Gambar X.XX: Ilustrasi jenis-jenis bentuk normal (*normal forms*)

## Tahapan Normalisasi

Secara sederhana, urutan dan syarat masing-masing bentuk normal adalah sebagai berikut:

1. **Tabel Universal**, yaitu tabel mentah yang mana masih berada dalam bentuk yang tidak normal sama sekali.
2. **1NF**, syaratnya adalah:
  - a. Bersifat *atomic values*.
  - b. Konsistensi data.
  - c. Memiliki nama kolom yang unik.
3. **2NF**, syaratnya adalah:
  - a. Sudah memenuhi kriteria 1NF.
  - b. Terbebas dari ketergantungan sebagian (*partial dependency*).
4. **3NF**, syaratnya adalah:
  - a. Sudah memenuhi kriteria 2NF.
  - b. Terbebas dari ketergantungan transitif (*transitive dependency*).
5. **BCNF**, syaratnya adalah:
  - a. Sudah memenuhi kriteria 3NF.
  - b. Untuk dependensi  $A \rightarrow B$ , maka A harus sebagai Super Key.

Penjelasan lebih lanjut tentang 1NF, 2NF, 3NF, dan BCNF akan dibahas di sub bab berikut ini.

### Tabel Universal

Tabel universal (*star tabel*) merupakan bentuk tabel yang paling awal alias data mentah, dan berada dalam bentuk yang tidak normal. Karena berada dalam bentuk yang tidak normal, maka sangat memungkinkan terjadinya redundansi, inkonsistensi dan anomali data. Umumnya, tabel mentah hanya menggunakan 1 tabel saja dengan field/kolom/atribut yang cukup banyak. Namun tidak menutup kemungkinan, tabel universal terdiri dari lebih dari 1 tabel yang masih berada dalam bentuk tidak normal.

no_order	tanggal	no_invoice	tanggal	no_produk	nama_produk	jml_order
20120	1 Maret 2019	100231	29/3/19	P1	Laptop	5
20130	19032019	100232	29/3/19	P4	Printer	2
20221	26/3/19	100233	30/3/19	P4	Printer	3
				P5	Speaker	3
				P2, P3	Keyboard, Mouse	@5

**Tabel X.XX: Contoh tabel universal order\_report**

Pada contoh tabel universal di atas, terlihat bahwa tabel tersebut sangat tidak normal. Tabel X.XX bahkan tidak memenuhi bentuk normal pertama (*INF*), dan memungkinkan terjadi anomali saat proses insert, update dan delete data. Selanjutnya, dengan adanya anomali data, maka fungsi DML (*Data Manipulation Language*) dan DQL (*Data Query Language*) tidak dapat berjalan dengan baik. Sebagai contoh, jika kita hendak menghapus nama produk ‘Laptop’ pada tabel X.XX, maka satu baris yang memuat data ‘Laptop’ akan turut terhapus, termasuk data invoice dan order.

## 1NF – Bentuk Normal Pertama

Sebagaimana dibahas sebelumnya, bentuk normal pertama (*first normal form*) memiliki syarat sebagai berikut:

- Setiap field/kolom/atribut harus mengandung nilai tunggal (*atomic values*). Tidak boleh ada field yang memiliki nilai lebih dari satu (*multiple values*).
- Setiap field/kolom/atribut harus konsisten atau memiliki jenis nilai/data yang sama. Sebagai contoh: kolom nama tidak boleh berisi tanggal lahir, atau kolom nomor handphone tidak boleh berisi alfabet.
- Setiap field/kolom/atribut harus memiliki nama yang unik alias harus berbeda dengan kolom yang lain.

Langkah-langkah dalam melakukan normalisasi bentuk pertama, adalah sebagai berikut:

- Memastikan tabel universal memenuhi 3 syarat yang dibahas sebelumnya (*atomic values*, konsistensi data, dan nama kolom yang unik).
- Mengelompokkan atribut-atribut yang dilepaskan/dipecah atau didekomposisi dari tabel universal, dan menjadikannya sebagai tabel sendiri.

3. Memberikan primary key pada tabel baru yang terbentuk.

Sekarang mari kita coba terapkan 3 langkah di atas untuk menormalisasi (1NF) tabel universal sebagaimana tampak pada tabel X.XX. Langkah awal yang perlu kita lakukan ada memastikan item-itemnya memenuhi 3 syarat yang telah dibahas sebelumnya. Untuk itu, perhatikan gambar X.XX dimana telah diberi tanda item-item yang perlu dinormalisasi. Penjelasannya adalah sebagai berikut:

no_order	tanggal	no_invoice	tanggal	no_produk	nama_produk	jml_order
20120	1 Maret 2019	100231	29/03/19	P1	Laptop	5
20130	19032019	200232	29/03/19	P4	Printer	2
20221	26/3/19	100233	30/03/19	P4	Printer	3
				P5	Speaker	3
				P2, P3	Keyboard, Mouse	@5
						4

Gambar X.XX: ilustrasi item-item yang perlu dinormalisasi

1. Item ini berisikan nama kolom yang tidak unik, dimana terdapat 2 kolom dengan nama yang sama, yaitu ‘tanggal’. Untuk menormalisasi item ini, maka kita harus merubah salah satu kolom tersebut atau bahkan keduanya, untuk memberikan identitas yang jelas pada masing-masing kolomnya. Di sini kita akan menggantinya dengan nama ‘tgl\_order’ dan ‘tgl\_invoice’. Hasil perubahannya tampak pada gambar X.XX.
2. Pada item ini, data tanggal atau tgl\_order ditulis secara tidak konsisten, dimana tampak menggunakan 3 format yang berbeda-beda. Jika menemui kasus semacam ini, yang perlu kita lakukan adalah membuatnya normal, yaitu dengan menentukan 1 format tanggal dan merubah isiannya sehingga terlihat konsisten. Misal di sini kita menggunakan format DD/MM/YY. Perhatikan gambar X.XX untuk hasil perubahannya.
3. Kasus nomor 3 melibatkan dua kolom dengan data yang bersifat *multiple value* alias tidak atomic. Untuk membuatnya normal, maka data seperti ini harus dipecah ke dalam baris yang berbeda, sebagaimana tampak pada gambar X.XX.
4. Masih berkaitan dengan nomor 3, item ini memiliki maksud bahwa kedua produk tersebut dipesan sebanyak masing-masing 5 unit. Penambahan simbol @ didepan angka dapat menimbulkan

inkonsistensi data pada kolom `jml_order`. Oleh karena itu, sebaiknya penulisannya dipecah ke dalam baris yang berbeda, menyesuaikan hasil nomor 3, sebagaimana tampak pada gambar X.XX.

5. Kasus nomor 5 menunjukkan adanya baris yang tidak semestinya kosong. Baris tersebut bisa diisi dengan nomor order, tanggal order, nomor invoice dan tanggal invoice sebagaimana mestinya. Jangan biarkan sel-sel tersebut kosong agar proses DML dan DQL berjalan normal.

<code>no_order</code>	<code>tgl_order</code>	<code>no_invoice</code>	<code>tgl_invoice</code>	<code>no_produk</code>	<code>nama_produk</code>	<code>jml_order</code>
20120	01/03/19	100231	29/03/19	P1	Laptop	5
20130	19/03/19	100232	29/03/19	P4	Printer	2
20221	26/03/19	100233	30/03/19	P4	Printer	3
20221	26/03/19	100233	30/03/19	P5	Speaker	3
20221	26/03/19	100233	30/03/19	P2	Keyboard	5
20221	26/03/19	100233	30/03/19	P3	Mouse	5

Gambar X.XX: Tampak hasil penggeraan langkah awal normalisasi 1NF

Saat ini, kita telah menyelesaikan tahap pertama untuk normalisasi 1NF. Yang perlu kita lakukan adalah mendekomposisi tabel hasil dari langkah pertama. Yang perlu kita perhatikan dari tabel di atas adalah bahwa kolom `no_order`, `tgl_order`, `no_invoice`, dan `tgl_invoice` memungkinkan terjadinya perulangan dengan isian yang sama persis. Oleh karena itu, keempat kolom atau atribut ini bisa dipisahkan menjadi tabel baru dengan nama tabel `order_record`. Sedangkan 3 atribut sisanya juga akan dijadikan sebagai tabel baru dengan nama tabel `order_item`. Dengan demikian kita memiliki 2 tabel yang terpisah, yaitu `order_record` dan `order_item` (perhatikan tabel X.XX dan tabel X.XX). Pada tabel `order_record`, nomor order 20221 cukup ditulis 1 kali. Nantinya akan dibuat kolom tambahan pada tabel `order_item` untuk menghubungkan kedua tabel tersebut, sehingga nomor order 20221 bisa terhubung ke 4 produk yang dipesan.

no_order	tgl_order	no_invoice	tgl_invoice
20120	01/03/19	100231	29/03/19
20130	19/03/19	100232	29/03/19
20221	26/03/19	100233	30/03/19

Tabel X.XX: tabel *order\_record*

no_produk	nama_produk	jml_order
P1	Laptop	5
P4	Printer	2
P4	Printer	3
P5	Speaker	3
P2	Keyboard	5
P3	Mouse	5

Tabel X.XX: tabel *order\_item*

Sampai di sini, tabel universal telah kita dekomposisi menjadi 2 tabel berbeda, yaitu tabel *order\_record* dan tabel *order\_item*. Agar kedua tabel tersebut bisa terhubung, dimana nomor order menentukan produk apa yang dipesan, maka kita tambahkan kolom *no\_order* pada tabel *order\_item*, sekaligus mengisinya sesuai dengan nomor order dan barang yang dipesan (sekaligus dengan jumlahnya). Sehingga hasil akhirnya akan tampak seperti pada tabel X.XX dan tabel X.XX. Dengan demikian, normalisasi 1NF sudah berhasil kita lakukan.

no_order	tgl_order	no_invoice	tgl_invoice
20120	01/03/19	100231	29/03/19
20130	19/03/19	100232	29/03/19
20221	26/03/19	100233	30/03/19

Tabel X.XX: tabel *order\_record*

no_order	no_produk	nama_produk	jml_order
20120	P1	Laptop	5
20130	P4	Printer	2
20221	P4	Printer	3
20221	P5	Speaker	3
20221	P2	Keyboard	5
20221	P3	Mouse	5

Tabel X.XX: tabel *order\_item*

## 2NF – Bentuk Normal Kedua

Bentuk normal ke-dua (*2<sup>nd</sup> Normal Form*) ialah bentuk normal yang berbasiskan konsep ketergantungan fungsional secara penuh (*full functional dependency*).

2NF, syaratnya adalah:

1. Sudah memenuhi kriteria 1NF.
2. Terbebas dari ketergantungan sebagian (*partial dependency*).

### **3NF – Bentuk Normal Ketiga**

Syarat terbentuknya bentuk normal ketiga, yaitu:

1. Sudah memenuhi kriteria 2NF.
2. Terbebas dari *transitive dependency*.

### **BCNF – Bentuk Normal Boyce-Codd**

Syarat terbentuknya bentuk normal ketiga, yaitu:

1. Sudah memenuhi kriteria 3NF.
2. Untuk dependensi  $A \rightarrow B$ , maka A harus sebagai *super key*.

## BAB 9

# PENGENALAN SQL

---

### Bab ini membahas:

- Pengantar SQL
  - Pengelompokan Perintah SQL
  - SQL Sebagai Sub Bahasa
  - Antarmuka SQL Terhadap DBMS
  - Elemen SQL
  - Integrasi dan Relasi Tabel
- 

## Pengantar SQL

Nama SQL merupakan kepanjangan dari *Structured Query Language*. Pada awalnya, SQL disebut dengan nama SEQUEL yaitu, *Structured English Query Language* yang didesain dan dikembangkan pada tahun 1970-an di Laboratorium Penelitian IBM oleh E.F.Codd. Pada tahun 1986, SQL distandardkan oleh ANSI yang kemudian diadopsi sebagai standar internasional oleh International Organization for Standardization (ISO) pada tahun 1987. Saat ini, banyak DBMS yang mendukung SQL, dapat dijalankan di berbagai platform perangkat keras dari PC ke mainframe.

SQL merupakan Bahasa query database yang mendukung pengambilan, manipulasi dan administrasi data yang disimpan dalam bentuk tabel. SQL adalah Bahasa yang memungkinkan pengguna berinteraksi dengan relasional database. Secara ideal, tujuan dari SQL adalah:

- Membuat database dan struktur relasi
- Melakukan tugas manajemen basis data seperti memasukkan, memodifikasi dan menghapus data dari relasi
- Melakukan query sederhana dan kompleks

SQL memiliki tiga komponen yaitu DDL (*Data Definition Language*), DML (*Data Manipulation Language*) dan DCL (*Data Control Language*). DDL digunakan untuk mendefinisikan struktur database dan mengontrol akses ke data. DML digunakan untuk mengambil dan memperbarui data. DCL digunakan untuk memberikan hak akses atau otoritas pengguna ke database. DDL dan DML tersebut memiliki aturan yang dapat menyematkan pernyataan SQL ke dalam Bahasa pemrograman seperti Java, PHP.

## Perintah SQL Dasar

Bahasa SQL terdiri dari beberapa bagian yaitu:

- DDL (*Data Definition Language*)
- DML (*Data Manipulation Language*).
- Integritas
- Definisi *View*
- Kontrol Transaksi
- *Embedded SQL*
- Otorisasi

Pada bab ini akan dibahas perintah SQL dasar pada DDL.

### SQL: Data Definition

Pada DDL, SQL fokus pada perintah membuat obyek database seperti *table*, *index*, dan *view* serta perintah untuk menentukan hak akses ke obyek database. Berikut adalah daftar perintah DDL:

**Tabel 9 Perintah SQL *Data Definition***

Command	Deskripsi
CREATE SCHEMA AUTHORIZATION	Membuat skema database
CREATE TABLE	Membuat table baru pada skema database pengguna
NOT NULL	Pastikan kolom tidak memiliki nilai NULL

UNIQUE	Pastikan kolom tidak memiliki nilai duplikat
PRIMARY KEY	Menentukan kunci utama untuk tabel
FOREIGN KEY	Menentukan kunci tamu untuk tabel
DEFAULT	Menentukan nilai default ketika tidak ada nilai yang diberikan
CHECK	Validasi data pada sebuah atribut
CREATE INDEX	Membuat index untuk tabel
CREATE VIEW	Membuat subset dinamis baris/kolom dari satu atau lebih tabel
ALTER TABLE	Modifikasi definisi tabel (tambah, modifikasi, atauhapus atribut atau <i>constraint</i> )
CREATE TABLE AS	Membuat tabel baru berdasarkan query pada skema database pengguna
DROP TABLE	Hapus tabel secara permanen
DROP INDEX	Hapus index secara permanen
DROP VIEW	Hapus view secara permanen

Perintah utama SQL untuk *data definition* adalah perintah CREATE, dimana perintah ini dapat digunakan untuk membuat skema, tabel(relasi), tipe dan domain serta konstruksi lainnya seperti *view*, *assertion*, dan *trigger*.

Proses membuat database secara signifikan berbeda dari produk ke produk. Pada *single-user system*, standar database (*default*) dapat dibuat ketika sistem diinstal dan dikonfigurasi, dapat dibuat oleh pengguna sesuai kebutuhan.

Standar ISO tidak menentukan bagaimana database dibuat dan umumnya tiap dialek memiliki pendekatan yang berbeda.

## 1. CREATE SCHEMA

Berdasarkan standar ISO, relasi dan objek database berada dalam *environment*. Setiap *environment* terdiri dari satu atau lebih *catalog*, dan masing - masing *catalog* terdiri dari satu set skema. Skema adalah kumpulan objek database yang beberapa hal terkait satu sama lain (semua objek dalam database dijelaskan dalam satu skema atau yang lain). Objek dalam skema bisa berupa *table*, *view*, *domain*, *assertion*, *collation*, *translation*, dan set *character*. Skema diidentifikasi oleh **nama skema** dan pengidentifikasi otorisasi untuk mengindikasikan pengguna yang memiliki skema serta deskriptor untuk tiap elemen dalam skema. Skema dibuat melalui perintah **CREATE SCHEMA**. Skema dapat diberi nama dan pengidentifikasi otorisasi, elemen – elemennya dapat didefinisikan nanti. Sebagai contoh, saya akan membuat skema untuk toko online (*e-commerce*) dengan nama skema TOKOKU dan pengidentifikasi otorisasi adalah ‘FASILKOM’.

### **CREATE SCHEMA TOKOKU AUTHORIZATION ‘DINUS’;**

Pada umumnya, tidak semua pengguna diizinkan untuk membuat skema dan elemen skema. Hak untuk membuat skema, *table*, dan konstruksi lainnya harus secara eksplisit diberikan kepada akun pengguna yang relevan oleh *system administrator* atau DBA.

Instalasi database memiliki standar *environment* dan skema, jadi ketika pengguna telah terkoneksi dan masuk ke dalam instalasi database. Pengguna akan diarahkan langsung ke tabel dan konstruksi lainnya dalam skema tersebut, tanpa menentukan nama skema.

## 2. TIPE DATA

Tipe data dasar yang tersedia untuk atribut adalah *numeric*, *character*, *Boolean*, *date*, dan *time*.

- **Numeric**, tipe data angka yang terdiri dari bilangan bulat dari berbagai ukuran seperti INTEGER atau INT, SMALLINT dan tipe data angka *floating-point* seperti FLOAT, REAL, dan DOUBLE. Format bilangan *floating-point* dapat dideklarasikan menggunakan DECIMAL(i,j) atau DEC(i,j) atau NUMERIC (i,j) dimana i adalah presisi yaitu jumlah digit angka (panjang digit termasuk desimal) dan j adalah skala yaitu jumlah digit decimal. Standar untuk skala adalah nol dan presisi ditentukan oleh

- implementasi. Contoh: DECIMAL(7,2) memiliki panjang digit (presisi) 7 dan jumlah decimal (skala) 2. Nilai 99.999,99 memiliki presisi 7 dan skala 2 (jumlah digit setelah desimal).
- **Character**, tipe data karakter dinyatakan dengan CHAR(n) atau CHARACTER(n), dengan n adalah jumlah dari karakter (panjang karakter tetap) atau panjang bervariasi – VARCHAR(n). Nilai string ditempatkan diantara tanda kutip tunggal (apostrof), dan bersifat *case sensitive*, artinya ada perbedaan antara penulisan *uppercase* dengan *lowercase*. Contoh VARCHAR(10) memiliki panjang karakter maksimal 10. Misal karakter ‘Dataku’ memiliki jumlah karakter 6, maka ukuran penyimpanan hanya 6 karakter. Berbeda dengan CHAR(10), walaupun karakter yang diinput adalah 1,3, 4, atau 6 karakter, SQL tetap menyimpan kolom tersebut untuk 10 karakter. Kesimpulannya, tipe data CHAR lebih cocok untuk kolom dengan panjang tetap (jumlah karakter tetap) seperti id pelanggan, nomor ktp pelanggan, id produk. Sedangkan VARCHAR lebih cocok digunakan untuk kolom dengan panjang bervariasi seperti nama pelanggan, nama produk, alamat.
  - **Boolean**, tipe data logika yang hanya berisi TRUE atau FALSE.
  - **Date dan Time**, tipe data yang berisi komponen waktu yaitu YEAR, MONTH, dan DAY, formatnya adalah YYYY-MM-DD. Tipe data TIME berisi komponen HOUR, MINUTE, dan SECOND, formatnya adalah HH:MM:SS. Nilai direpresentasikan dalam tanda kutip tunggal. Contoh Date ‘2019-04-21’ atau Time ’08:24:50’.

### 3. CREATE TABLE

Tabel dibuat dengan perintah CREATE TABLE. Perintah CREATE TABLE digunakan untuk menentukan relasi baru dan menentukan atribut dengan *constraint* awal. Berikut adalah syntak dasar dalam penulisan perintah CREATE TABLE.

```

CREATE TABLE nama tabel
(
    Nama kolom 1 tipe data [constraint] [,
    Nama kolom 2 tipe data [constraint]] [,
    PRIMARY KEY (nama kolom 1)] [,,
    FOREIGN KEY (list foreign key kolom)
        REFERENCES tabel parent (ListKunciKandidatKolom) ]
        ON UPDATE AksiReferensi
        ON DELETE
);

```

Perintah CREATE TABLE digunakan untuk menentukan relasi baru dengan memberikan nama dan menentukan atribut dan *constraint*. Pertama, tentukan nama tabel terlebih dahulu. Di dalam tabel terdapat atribut, tipe data untuk menentukan domain nilai dari atribut, dan *constraint* seperti NOT NULL.

PRIMARY KEY menentukan kolom sebagai kunci utama untuk tabel. Pada pembuatan tabel standar, hanya ada 1(satu) PRIMARY KEY dan umumnya CONSTRAINT pada kolom tersebut di set NOT NULL. Tujuan dari PRIMARY KEY adalah mencegah adanya duplikasi data dalam tabel tersebut. Sebagai contoh kolom id barang, id produk, atau id pelanggan tidak boleh sama dalam tabel. SQL menolak operasi INSERT atau UPDATE jika terdapat duplikasi data pada kolom PRIMARY KEY. SQL memberikan keunikan pada PRIMARY KEY.

FOREIGN KEY menentukan kunci tamu pada tabel (anak) dan hubungan yang dimilikinya dengan tabel (induk) lainnya. klausa FOREIGN KEY mengimplementasikan referensi integritas *constraint* sebagai berikut:

- *List foreign key kolom* merupakan kolom dari tabel yang membentuk kunci tamu.
- REFERENCES memberikan referensi ke tabel (induk), dimana disesuaikan dengan atribut yang telah ditentukan (*list foreign key kolom*). Jika *ListKunciKandidatKolom* dihilangkan, maka FOREIGN KEY diasumsikan telah cocok dengan PRIMARY KEY pada tabel induk. Dalam hal ini, tabel induk harus memiliki PRIMARY KEY.

- ON UPDATE menentukan aksi yang diambil ketika *ListKunciKandidatKolom* mengalami pembaruan pada tabel induk, maka atribut FOREIGN KEY pada tabel (anak) secara otomatis mengalami pembaruan.
- AksiReferensi dapat berupa CASCADE, SET NULL, SET DEFAULT, atau NO ACTION. Jika klausa ON UPDATE dihilangkan, maka (default) NO ACTION akan diasumsikan.

Sebagai contoh, kita akan membuat tabel produk dan tabel kategori.

**Tabel 10 Struktur Tabel Produk dan Kategori**

Nama Table	Nama Atribut	Tipe Data	PK atau FK	Referensi Tabel
Produk	id_produk	INTEGER	PK	
	id_kategori	INTEGER	FK	Kategori
	nama_produk	VARCHAR		
	deskripsi	TEXT		
	harga	INTEGER		
	stok	INTEGER		
	tgl_masuk	DATE		
	gambar	VARCHAR		
Kategori				
	id_kategori	INTEGER	PK	
	nama_kategori	VARCHAR		

Pada tabel 2, kita asumsikan id\_kategori menjadi kunci tamu pada tabel induk yaitu tabel **Produk**.

CREATE TABEL PRODUK(

```

    id_produk          INT NOT NULL,
    id_kategori        INT NOT NULL,
    nama_produk        VARCHAR(30) NOT NULL,
    deskripsi          TEXT,
    harga              INT NOT NULL,
```

```
stok          INT NOT NULL,  
tgl_masuk      DATE,  
gambar VARCHAR(50) NOT NULL,  
PRIMARY KEY(id_produk),  
FOREIGN KEY (id_kategori)  
    REFERENCES KATEGORI(id_kategori)  
    ON UPDATE CASCADE  
);
```

```
CREATE TABEL KATEGORI(  
    id_kategori      INT NOT NULL,  
    nama_kategori    VARCHAR(30) NOT NULL,  
    PRIMARY KEY(id_kategori)  
);
```

Pada penulisan SQL diatas tidak semua atribut pada tabel PRODUK memiliki CONSTRAINT NOT NULL. NOT NULL hanya dijadikan sebagai kondisi bahwa atribut nantinya tidak boleh kosong. Umumnya, pada toko online, penjual tidak boleh mengosongi atribut id produk, id kategori, nama produk, harga, stok, dan gambar. Standarnya, atribut – atribut tersebut adalah atribut yang wajib ditampilkan pada tampilan produk.

CONSTRAINT pada FOREIGN KEY menggunakan ON UPDATE CASCADES, tujuannya untuk otomatisasi pembaruan pada tabel PRODUK. jika ada perubahan pada tabel KATEGORI, pengguna tidak perlu merubah id\_kategori pada tabel PRODUK karena otomatis nilai akan berubah sesuai id\_kategori pada tabel KATEGORI.

# BAB 10

## PEMROSESAN QUERY

---

### Bab ini membahas:

- Struktur Data
  - Fungsi Agregasi
  - Nilai Null
  - Manipulasi Data
- 

## Manipulasi Data

Perintah DML fokus pada bagaimana SQL dapat memasukkan, memodifikasi, menghapus dan menampilkan data dalam tabel database. Berikut adalah daftar perintah DML:

**Tabel 11 Perintah SQL Data Manipulation**

Command	Deskripsi
INSERT	Memasukkan baris dalam tabel
SELECT	Memilih atribut dari baris dalam satu atau lebih tabel atau view
WHERE	Batas pemilihan baris berdasarkan ekspresi kondisional
ORDER BY	Permintaan dari baris yang dipilih berdasarkan satu atau lebih atribut. Biasanya digunakan untuk pengurutan nilai

---

GROUP BY	Pengelompokan baris yang dipilih berdasarkan satu atau lebih atribut
HAVING	Batas pemilihan baris yang telah dikelompokan berdasarkan kondisi
UPDATE	Modifikasi nilai atribut dalam satu atau lebih baris tabel
DELETE	Menghapus satu atau lebih baris dari tabel
COMMIT	Menyimpan perubahan data secara permanen
ROLLBACK	Mengembalikan data ke nilai aslinya
=, <, >, <=, >=, ◊	Digunakan untuk ekspresi kondisi
AND/OR/NOT	Digunakan untuk ekspresi kondisi
BETWEEN	Cek apakah nilai atribut berada pada rentang nilai
IS NULL	Cek apakah nilai atribut adalah null
LIKE	Cek apakah nilai atribut cocok dengan string yang diberikan
IN	Cek apakah nilai atribut cocok dengan nilai dalam daftar nilai
EXISTS	Cek apakah subquery mengembalikan baris (row)
DISTINCT	Membatasi nilai dengan nilai unik

	Digunakan dengan SELECT untuk mengembalikan ringkasan matematika pada kolom
COUNT	Mengembalikan nilai pada baris dengan nilai non-null untuk kolom yang diberikan
MIN	Mengembalikan nilai atribut minimum yang ditemukan pada kolom
MAX	Mengembalikan nilai atribut maksimum yang ditemukan pada kolom
SUM	Mengembalikan penjumlahan semua nilai pada kolom
AVG	Mengembalikan rata – rata nilai pada kolom

Pada subbab *Data Manipulation* akan dibahas perintah SQL dasar seperti *insert*, *select*, *update*, *delete*, operator logika, operator perbandingan, operator special, fungsi agregat dan nilai null.

## INSERT

Perintah INSERT digunakan untuk memasukkan *tuple* tunggal (row) ke dalam relasi (tabel). Data yang dimasukkan ke dalam tabel harus sesuai dengan format nama tabel dan atributnya. Selain itu, nilai yang dimasukkan harus disesuaikan dengan urutan dari atribut dalam tabel. Dalam memasukkan nilai perlu diperhatikan cara penulisannya:

- Konten (row) dimasukkan dalam tanda kurung.
- Karakter (string) dan nilai bertipe *date* ditulis dalam tanda petik tunggal (apostrof).
- Data numerik tidak menggunakan apostrof.
- Atribut dimasukkan menggunakan koma.

- nilai diperlukan untuk setiap kolom dalam tabel.
- 

Berikut sintak penulisan INSERT:

**INSERT INTO nama\_tabel VALUES (nilai1, nilai2,..., nilai-N)**

Misal kita akan memasukkan nilai pada tabel KATEGORI (tabel 2)

INSERT INTO KATEGORI VALUES (1,'Komputer');

Pada perintah diatas, tabel KATEGORI akan terisi:

id_kategori	nama_kategori
1	Komputer

Bagi pengguna yang tidak hafal dengan urutan dari atributnya, perintah INSERT juga dapat ditulis dengan menyertakan atribut pada tabel yang dituju. Selain itu, penulisan ini berguna jika hanya beberapa atribut yang diisi pada tabel tersebut. Namun, perlu diperhatikan juga CONSTRAINT yang digunakan pada atribut tersebut. Hanya atribut yang tidak memiliki CONSTRAINT NOT NULL, diijinkan untuk dikosongi (tidak diisi). Selain itu, pengguna juga tidak dapat mengosongi atribut yang telah di set PRIMARY KEY. Sebagai contoh pada tabel PRODUK (tabel 2), atribut **deskripsi** dan atribut **tgl\_masuk** tidak di set NOT NULL. Jika kita ingin melewatkkan dua atribut tersebut, maka penulisannya:

INSERT INTO PRODUK(id\_produk, id\_kategori, nama\_produk, harga, stok, gambar)

VALUES (01, 1, 'ASUS', 16500000, 10, 'asus.jpg');

id_produk	id_kategori	nama_produk	deskripsi	harga	stok	tgl_masuk	gambar
01	1	ASUS		1650000	10		asus.jpg

## **SELECT**

Perintah SELECT digunakan untuk menampilkan satu atau lebih data pada tabel database. Data yang ditampilkan (diambil) disesuaikan dengan kondisi yang

telah digunakan pada perintah SELECT. Jadi, tidak semua nilai atribut ditampilkan pada SQL. Secara umum, perintah SELECT dituliskan dalam bentuk:

**SELECT \* FROM *nama\_tabel*;**

Perintah diatas digunakan untuk menampilkan semua atribut pada tabel tersebut. Misal, kita akan menampilkan semua nilai atribut pada tabel PRODUK, maka penulisannya:

**SELECT \* FROM PRODUK;**

Jika diamati, pada perintah diatas ada tanda \* setelah perintah SELECT. Tanda \* digunakan untuk mengekspresikan semua kolom yang ingin ditampilkan tabel PRODUK. Jika penggunaan perintah SELECT tanpa menggunakan tanda \*, maka atribut harus ditulis semua. Sebagai contoh:

**SELECT id\_produk, id\_kategori, nama\_produk, deskripsi, harga, stok, tgl\_masuk, gambar FROM PRODUK;**

Hasil dari perintah diatas:

**Tabel 12 Hasil Tampilan tabel PRODUK**

id_produk	id_kategori	nama_produk	deskripsi	harga	stok	tgl_masuk	gambar
01	1	ASUS	Asus zenfone	1650000	10	27-3-2019	asus.jpg
02	2	Rubik	Game	75000	8	27-3-2019	Rubik.jpg
03	1	OPPO	Oppo F7	3500000	15	25-4-2019	Oppo.jpg
04	3	Kulkas 2 pintu	Alat rumah tangga	3750000	10	27-4-2019	Kulkas.jpg

Jika pengguna hanya ingin menampilkan beberapa atribut saja, maka tidak semua atribut dituliskan dalam perintah SELECT. Misalnya, pengguna hanya ingin menampilkan id\_produk, nama\_produk, dan harga. Maka perintah SQL dapat ditulis:

```
SELECT id_produk, nama_produk, harga FROM PRODUK;
```

Maka, SQL hanya menampilkan 3 atribut sesuai dengan pilihan atribut yang dituliskan pada perintah SELECT.

**Tabel 13 Tampilan tabel PRODUK dengan 3 atribut pilihan**

id_produk	nama_produk	harga
01	ASUS	1650000
02	Rubik	75000
03	OPPO	3500000
04	Kulkas 2 pintu	3750000

### 1. Klausu WHERE

Pada perintah SELECT, pengguna dapat memberikan kondisi tertentu untuk menampilkan data berdasarkan batasan – batasan yang diberikan. Misalnya, pada tabel PRODUK, pengguna ingin menampilkan tabel produk dengan harga diatas 1.000.000, menampilkan tabel produk dengan id kategori 1, atau menampilkan tabel produk dengan stok lebih dari 10. Klausu WHERE memberikan batasan – batasan pada tabel yang ingin ditampilkan. Format penulisannya, klausu WHERE diikuti kondisi atau batasan yang menentukan baris yang diambil. Contoh: Menampilkan produk yang harganya diatas 1.000.000

SELECT \* FROM PRODUK WHERE harga > 1000000;

Perintah SQL diatas hanya menampilkan nilai atribut sesuai dengan kondisi harga diatas 1.000.000.

Maka hasil dari perintah diatas:

**Tabel 14 Tampilan tabel PRODUK dengan klausu WHERE**

id_produk	id_kategori	nama_produk	deskripsi	harga	stok	tgl_masuk	gambar
01	1	ASUS	Asus zenfone	1650000	10	27-3-2019	asus.jpg
03	1	OPPO	Oppo F7	3500000	15	25-4-2019	Oppo.jpg

04	3	Kulkas 2 pintu	Alat rumah tangga	3750000	10	27-4- 2019	Kulkas. jpg
----	---	-------------------	-------------------------	---------	----	---------------	----------------

Tabel 6 hanya menampilkan data dengan harga diatas 1.000.000. terbukti bahwa, produk RUBIK dengan harga 75.000 tidak ditampilkan pada tabel produk.

Klausa WHERE juga memungkinkan untuk memberikan batasan lebih dari satu. Terdapat 5 standar penulisan kondisi pada perintah SELECT, yaitu:

- 1) **Perbandingan**, membandingkan nilai dari satu ekspresi dengan nilai ekspresi lainnya. Operator yang digunakan adalah operator perbandingan (lihat tabel 3)
- 2) **Rentang (Range)**, apakah nilai ekspresi terdapat dalam rentang nilai yang ditentukan. Klausa yang digunakan adalah BETWEEN/NOT BETWEEN
- 3) **Pengaturan keanggotaan**, apakah nilai ekspresi sama dengan serangkaian nilai yang telah ditentukan. Klausa yang digunakan adalah IN/NOT IN/
- 4) **Kecocokan pola (pattern match)**, apakah string cocok dengan pola yang telah ditentukan. Klausa yang digunakan adalah LIKE/NOT LIKE
- 5) **Null**, apakah kolom memiliki nilai null. Klausa yang digunakan adalah IS NULL/IS NOT NULL

Dalam penggunaan kondisi diatas, SQL juga mengijinkan penggunaan operator logika seperti AND, OR, NOT sebagai penambahan kondisi lebih dari satu. Berikut beberapa contoh penggunaan perintah SELECT yang melibatkan operator spesial, operator perbandingan, dan operator logika:

- Menampilkan nilai atribut dengan harga diatas 1.000.000 dan stok diatas 10 atau sama dengan 10.

SELECT \* FROM PRODUK

WHERE harga > 1000000 AND STOK >=10;

Maka tabel PRODUK yang ditampilkan akan berbeda pada tabel 6, karena terdapat tambahan batasan yaitu, stok diatas 10 atau sama dengan 10.

**Tabel 15 Tampilan tabel PRODUK dengan batasan AND**

id_	id_	nama_	deskripsi	harga	stok	tgl_	gambar
-----	-----	-------	-----------	-------	------	------	--------

produk	kategori	produk				masuk	
01	1	ASUS	Asus zenfone	1650000	10	27-3-2019	asus.jpg
04	3	Kulkas 2 pintu	Alat rumah tangga	3750000	10	27-4-2019	Kulkas.jpg

- Menampilkan nilai atribut dengan rentang nilai harga antara 3.000.000 dan 4.000.000

**SELECT \* FROM PRODUK**

**WHERE harga BETWEEN 3000000 AND 4000000;**

Klausa BETWEEN menampilkan nilai atribut berdasarkan rentang nilai yang ditentukan. SQL akan menampilkan atribut dengan nilai 3.000.0000 , 4.000.000 atau dalam rentang 3.000.000 – 4.000.000.

**Tabel 16 Tampilan tabel PRODUK dengan klausa BETWEEN**

id_produk	id_kategori	nama_produk	deskripsi	harga	stok	tgl_masuk	gambar
03	1	OPPO	Oppo F7	3500000	15	25-4-2019	Oppo.jpg
04	3	Kulkas 2 pintu	Alat rumah tangga	3750000	10	27-4-2019	Kulkas.jpg

Versi lain dari klausa BETWEEN adalah NOT BETWEEN. Klausa NOT BETWEEN digunakan untuk menampilkan nilai diluar range yang ditentukan. Dalam hal ini, NOT BETWEEN merupakan negasi dari BETWEEN.

Klausa BETWEEN juga dapat representasikan dengan menggunakan operator perbandingan, seperti:

**SELECT \* FROM PRODUK**

---

WHERE harga >= 3000000 AND harga <= 4000000;

Perintah SQL diatas juga menampilkan hasil yang sama seperti pada tabel 8. Bagaimanapun juga, klausa BETWEEN adalah cara termudah dalam mengekspresikan kondisi dalam rentang nilai.

- Menampilkan atribut dalam pengaturan keanggotaan (*set membership*). Misal menampilkan produk ‘ASUS’

**SELECT \* FROM PRODUK**

WHERE nama\_produk IN ('ASUS', 'OPPO');

**Tabel 17 Tampilan Tabel PRODUK dengan klausa IN**

id_produk	id_kategori	nama_produk	deskripsi	harga	stok	tgl_masuk	gambar
01	1	ASUS	Asus zenfone	1650000	10	27-3-2019	asus.jpg
03	1	OPPO	Oppo F7	3500000	15	25-4-2019	Oppo.jpg

Klausa IN digunakan untuk mengecek apakah nilai atribut cocok dengan nilai yang telah ditentukan, dalam kasus ini apakah cocok dengan nilai ‘ASUS’ atau ‘OPPO’.

Versi lainnya, untuk mengecek keanggotaan dari nilai atribut pada tabel dapat menggunakan operator logika **OR**, seperti:

**SELECT \* FROM PRODUK**

WHERE nama\_produk = ‘ASUS’ or nama\_produk=’OPPO’;

Bagaimanapun, klausa IN lebih efisien untuk mengekspresikan penggunaan kondisi uji keanggotaan.

## 2. Klausa ORDER BY

Pada umumnya, tampilan baris (row) pada tabel SQL tidak diatur dalam posisi tertentu. Secara standar, tampilan baris diurutkan berdasarkan *primary key*. Dalam kasus toko online, konsumen ingin menampilkan produk sesuai dengan

urutan yang diinginkan, misalnya konsumen ingin menampilkan produk dengan urutan harga yang terendah sampai harga tertinggi atau menampilkan produk dengan urutan stok terbanyak ke stok terendah. Pengurutan nilai atribut pada tabel menggunakan klausua **ORDER BY** di dalam perintah SELECT. Format penulisannya, klausua ORDER BY diikuti dengan kolom atribut yang ingin digunakan sebagai patokan pengurutan. SQL juga mengijinkan atribut diurutkan secara *ascending* (ASC) atau *descending* (DESC). Berikut contoh penggunaan klausua ORDER BY dalam perintah SELECT.

```
SELECT * FROM PRODUK
```

```
ORDER BY harga DESC;
```

Perintah diatas menampilkan tabel PRODUK yang diurutkan berdasarkan nilai atribut harga secara *descending* (urutan nilai terbesar ke nilai terkecil).

**Tabel 18 Tampilan tabel PRODUK dengan klausua ORDER BY**

id_produk	id_kategori	nama_produk	deskripsi	harga	stok	tgl_masuk	gambar
04	3	Kulkas 2 pintu	Alat rumah tangga	3750000	10	27-4-2019	Kulkas.jpg
03	1	OPPO	Oppo F7	3500000	15	25-4-2019	Oppo.jpg
01	1	ASUS	Asus zenfone	1650000	10	27-3-2019	asus.jpg
02	2	Rubik	Game	75000	8	27-3-2019	Rubik.jpg

Pada klausua ORDER BY juga memungkinkan adanya pernyataan lebih dari satu. Jika nilai atribut yang digunakan dalam pengurutan adalah unik (tidak ada nilai yang kembar), maka tidak diperlukan pernyataan tambahan untuk mengontrol pengurutan. Jika klausua ORDER BY dihadapkan pada kasus data yang kembar (seperti atribut id\_katergori), maka diperlukan pernyataan

tambahan agar pengurutan dapat dilakukan secara *ascending* atau *descending*. Perhatikan 2 contoh berikut:

```
SELECT * FROM PRODUK
ORDER BY id_kategori;
```

Pada tabel PRODUK, terdapat 2 nilai yang sama pada atribut id\_kategori yaitu 1. Karena nilai atributnya sama, maka pengurutan akan disusun sesuai dengan *primary key*.

**Tabel 19 Klausura ORDER BY dengan 1 pernyataan**

id_produk	id_kategori	nama_produk	deskripsi	harga	stok	tgl_masuk	gambar
01	1	ASUS	Asus zenfone	1650000	10	27-3-2019	asus.jpg
03	1	OPPO	Oppo F7	3500000	15	25-4-2019	Oppo.jpg
02	2	Rubik	Game	75000	8	27-3-2019	Rubik.jpg
04	3	Kulkas 2 pintu	Alat rumah tangga	3750000	10	27-4-2019	Kulkas.jpg

Pada contoh yang kedua, ORDER BY dapat diikuti oleh 2 pernyataan.

```
SELECT * FROM PRODUK
ORDER BY id_kategori, harga DESC;
```

**Tabel 20 Klausura ORDER BY dengan 2 pernyataan**

id_produk	id_kategori	nama_produk	deskripsi	harga	stok	tgl_masuk	gambar

03	1	OPPO	Oppo F7	3500000	15	25-4-2019	Oppo.jpg
01	1	ASUS	Asus zenfone	1650000	10	27-3-2019	asus.jpg
02	2	Rubik	Game	75000	8	27-3-2019	Rubik.jpg
04	3	Kulkas 2 pintu	Alat rumah tangga	3750000	10	27-4-2019	Kulkas.jpg

Pada tabel 12, selain diurutkan berdasarkan id\_kategori dimana nilai atributnya tidak unik, tabel dapat diurutkan secara *descending* pada atribut harga.

### 3. Klausula GROUP BY

Klausula GROUP BY digunakan untuk menentukan frekuensi distribusi baris (row) pada tabel. Dalam banyak kasus, klausula GROUP BY biasanya digunakan untuk mencari nilai rata – rata dari suatu kolom, nilai terendah atau tertinggi, menjumlahkan antara baris satu dengan baris lainnya dalam satu kolom dalam tiap jenisnya, atau digunakan untuk mengetahui banyaknya nilai untuk tiap jenisnya. Dalam toko online, klausula GROUP BY dapat dimanfaatkan untuk menampilkan harga terendah atau tertinggi produk, mengetahui berapa banyaknya jumlah suatu produk, atau menghitung rata – rata keuntungan. Sebagai contoh, kita akan menerapkan klausula GROUP BY pada tabel PRODUK untuk mengetahui banyaknya jumlah produk berdasarkan id\_kategori. Misal, tabel PRODUK telah terisi 10 data.

**Tabel 21 Tabel PRODUK dengan 10 data**

id_produk	id_kategori	nama_produk	deskripsi	harga	stok	tgl_masuk	gambar
01	1	ASUS Max pro m1	Asus max	1650000	10	27-3-2019	asus.jpg

02	2	Rubik	Game	75000	8	27-3-2019	Rubik.jpg
03	1	OPPO F7	Oppo F series	3500000	15	25-4-2019	Oppo.jpg
04	3	Kulkas 2 pintu	Alat rumah tangga	3750000	10	27-4-2019	Kulkas.Jpg
05	2	PS 4	Game console	7500000	20	03-5-2019	Ps4.jpg
06	2	PS 3	Game console	5500000	15	03-05-2019	Ps3.jpg
07	3	Mesin cuci LG	Alat rumah tangga	2500000	12	04-5-2019	MClg.jpg
08	1	Samsung A6+	Samsung a series	4500000	5	04-5-2019	samsung A6+.jpg
09	1	Asus max pro m2	Asus max series	3500000	25	05-5-2019	Asusm2.jpg
10	2	XBOX	Game console	5000000	10	05-5-2019	Xbox.jpg

Pada tabel PRODUK diatas, kita akan menampilkan jumlah produk berdasarkan id\_kategori. Berikut sintaknya:

```
SELECT id_kategori, COUNT (*)
FROM PRODUK
GROUP BY id_kategori;
```

Maka SQL akan menampikan:

**Tabel 22 Tabel PRODUK dengan Klausa GROUP BY**

id_kategori	Count(*)
1	4
2	4
3	2

Tabel 14 menampilkan banyaknya jumlah produk sesuai dengan id\_kategori. Pada tabel 13 dapat kita lihat, bahwa ada 4 produk dengan id\_kategori 1 dan 2, sedangkan ada 2 produk dengan id\_kategori 3.

Standar ISO, klausa GROUP BY digunakan dengan menerapkan fungsi agregasi (dibahas pada subbab selanjutnya). Fungsi agregasi terdiri dari COUNT, SUM, AVG, MIN, dan MAX. Fungsi – fungsi tersebut digunakan untuk mengumpulkan kolom dan baris sesuai dengan nilai atribut.

## UPDATE

Perintah UPDATE digunakan untuk modifikasi nilai atribut dari tupel yang dipilih. Pada perintah UPDATE menggunakan klausa WHERE untuk menentukan tupel mana yang akan dimodifikasi. Selain menggunakan klausa tambahan yaitu SET digunakan untuk modifikasi nilai baru yang akan dimasukkan. Berikut format penulisan UPDATE pada SQL:

*UPDATE nama\_tabel*

*SET nama\_kolom=nilai\_data*

*WHERE kondisi;*

Nama\_tabel adalah tabel yang dituju untuk dimodifikasi. Klausa SET digunakan sebagai penentu tupel yang akan di modifikasi. SET berisi nama\_kolom yang akan dimodifikasi. Pada klausa SET, jika atribut berupa numerik, maka modifikasi dapat berupa formula, misalnya pemberian diskon pada harga. Klausa WHERE merupakan optional, jika perintah UPDATE digunakan untuk memodifikasi semua nilai atribut pada tabel, maka klausa WHERE dihilangkan. Namun, jika perintah UPDATE digunakan hanya untuk memodifikasi tupel tertentu, maka klausa ditambahkan setelah klausa SET.

Contoh:

**UPDATE PRODUK**

---

```
SET harga=harga*0.2
WHERE id_kategori=1;
```

## **DELETE**

Perintah DELETE digunakan untuk menghapus tupel dari relasi tabel. Sama seperti perintah UPDATE, perintah DELETE melibatkan klausa WHERE untuk menentukan tupel yang akan dihapus. Berikut format penulisan perintah UPDATE.

```
DELETE FROM nama_tabel
```

```
WHERE kondisi;
```

Sifat klausa WHERE pada perintah DELETE adalah opsional. Jika ingin menghilangkan semua row pada tabel, maka klausa WHERE dihilangkan. Namun jika hanya row tertentu yang dihapus, maka klausa WHERE ditambahkan setelah klausa SET.

Contoh:

```
DELETE FROM PRODUK
```

```
WHERE stok=0;
```

Perintah DELETE hanya menghapus row pada tabel. Jika semua row terhapus, maka tabel masih berada pada database sebagai tabel kosong. Jika pengguna ingin menghapus tabel dari database, dapat menggunakan fungsi DROP TABLE.

## **Fungsi Agregasi**

Fungsi agregasi merupakan fungsi untuk meringkas informasi dari beberapa tuple ke tuple tunggal yang telah diringkas. Fungsi agregasi diterapkan pada klausa GROUP BY untuk mengelompokkan tuple – tuple yang akan diringkas. Standar SQL, ada 5 fungsi agregasi yang ditawarkan:

- **Average (AVG):** mengembalikan rata – rata nilai pada kolom yang ditentukan.
- **Minimum (MIN):** mengembalikan nilai terendah pada kolom yang ditentukan.
- **Maximum (MAX):** mengembalikan nilai tertinggi pada kolom yang ditentukan.
- **Total (SUM):** mengembalikan jumlah nilai pada kolom yang ditentukan.

- Count (**COUNT**): mengembalikan banyaknya baris pada kolom yang dipilih.

Contoh:

1. Menampilkan harga terendah pada tabel produk

```
SELECT MIN (harga)
```

```
FROM PRODUK;
```

Maka query akan mengembalikan nilai terendah pada semua baris dalam tabel PRODUK. Berdasarkan tabel 13, maka akan ditampilkan bahwa harga terendah adalah 75000 yaitu pada nama produk rubik.

Pada penulisan fungsi agregasi, SQL mengijinkan untuk merubah nama kolom guna menampilkan hasil dari penggunaan fungsi agregasi. Sebagai contoh, pada penulisan SQL diatas ditambahkan klausa AS.

```
SELECT MIN (harga) AS Harga_Terendah
```

```
FROM PRODUK;
```

**Tabel 23 Tampilan Harga Terendah pada tabel Produk**

Harga_Terendah
75000

2. Menampilkan harga pada tertinggi id\_produk=1

Pada contoh ini, fungsi agregasi diterapkan dengan menambahkan klausa WHERE, karena hanya menampilkan baris dengan kondisi id\_produk = 1. Berikut penulisan SQL-nya:

```
SELECT MAX (harga) AS Harga_Tertinggi
```

```
FROM PRODUK
```

```
WHERE id_produk=1;
```

**Tabel 24 Tampilan Harga Tertinggi pada id\_produk=1**

Harga_Tertinggi
4500000

## Nilai Null

Row dengan nilai null adalah row tanpa nilai. Nilai null tidak diartikan dengan nilai 0 atau yang berisi spasi. Nilai null digunakan untuk merepresentasikan nilai atribut yang mungkin tidak diketahui atau tidak berlaku pada tupel. Nilai spesial seperti NULL digunakan untuk merepresentasikan banyak arti dari tupel yang kosong. Sebagai contoh, pada tabel produk, ada produk yang tidak memiliki deskripsi (deskripsi=null) atau tanggal masuk tidak diisi (tgl\_masuk=null). Dalam hal ini, nilai null memiliki banyak arti seperti nilai tidak diketahui, nilai tidak tersedia, atau nilai tidak didefinisikan. SQL memiliki aturan yang berkaitan dengan nilai null. Berikut definisi penggunaan dari nilai null.

- *Unknown value*, digunakan untuk mewakili nilai tidak diketahui seperti tanggal lahir seseorang tidak diketahui, maka di dalam database dapat direpresentasikan sebagai nilai null. Contoh lain, nomor telepon seseorang mewakili nilai tidak diketahui karena kemungkinan orang yang bersangkutan tidak hafal atau orang tersebut tidak memiliki nomor telepon.
- *Unavailable/withheld value*, digunakan untuk mewakili nilai tidak tersedia seperti nomor telepon, ada yang tidak ingin nomor telepon disebar untuk umum, maka di dalam database dapat direpresentasikan sebagai nilai null.
- *Undefined value / not applicable attribute*, digunakan sebagai atribut nilai tidak berlaku. Sebagai contoh, atribut kartu kredit akan bernilai null bagi seseorang yang tidak memiliki kartu kredit. Dalam hal ini, atribut kartu kredit tidak berlaku bagi orang tersebut.

Namun pada aturan SQL, SQL tidak membedakan arti NULL yang berbeda tersebut. Karena makna null dapat digunakan oleh atribut – atribut yang dapat di set ke null. Sebagai contoh, atribut nomor telepon dapat berarti salah satu dari tiga definisi nilai null diatas atau semua definisi nilai null tersebut.

# BAB 11

## BEKERJA DENGAN SQL

---

### Bab ini membahas:

- Create
  - Drop
  - Alter
  - Insert
  - Update
  - Delete
- 

Banyak aplikasi MySQL yang dapat digunakan untuk praktek SQL, salah satunya adalah XAMPP. XAMPP merupakan aplikasi *open source* digunakan sebagai server lokal yang terdiri dari *Apache HTTP Server*, *MariaDB database*, *PHP* dan *PERL*.

Untuk dapat menggunakan MySQL, terlebih dahulu harus mengaktifkan Server MySQL. Caranya, masuk ke config MySQL di C : \xampp\mysql\bin.

Pertama, buka MS-DOS Prompt melalui Run kemudian ketik command atau cmd. Maka akan dapat masuk ke dalam direktori MySQL melalui Prompt seperti dibawah ini:

```
cd C:\xampp\mysql\bin
```

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\USER>cd C:\xampp\mysql\bin

C:\xampp\mysql\bin>
```

Sesuaikan penggunaan perintah yang digunakan sesuai dengan direktori dari XAMPP (sesuai letak instalasi XAMPP). Sebagai contoh jika direktori XAMPP terletak di dalam “program files”, maka perintah yang ditulis dalam prompt adalah:

```
cd C:\Program Files\xampp\mysql\bin
```

langkah selanjutnya adalah mengakses username dan password pada MySQL. Tujuannya, MySQL memberikan keamanan akses data dimana memiliki kemampuan dalam mengatur akses user. Jadi, tidak sembarang user dapat mengakses database dalam MySQL. Sebagai awal penggunaan MySQL, biasanya kita tidak memiliki username dan password pada MySQL. MySQL memberikan solusi berupa penggunaan username password secara *default* yaitu *username* adalah root dan password adalah kosong (tidak diisi). Berikut adalah perintah yang digunakan untuk mengkoneksikan ke server MySQL.

```
mysql -u root -p
```

```
Microsoft Windows [Version 10.0.17134.829]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\USER>cd C:\xampp\mysql\bin

C:\xampp\mysql\bin>mysql -u root
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 2
Server version: 10.1.30-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> -
```

Perintah *-u* pada diatas adalah tanda dari username dengan nama *root*. Perintah *-p* adalah tanda dari password. Karena password masih kosong, jadi tidak perlu disertakan dalam perintah akses ke server MySQL.

langkah selanjutnya, kita dapat secara langsung membuat database dengan perintah SQL create.

## Create

### Menciptakan Database

Database merupakan media utama yang harus dibuat untuk membuat basis data yang berisi tabel – tabel dengan field didalam tabel tersebut. perintah yang digunakan untuk menciptakan database pada MySQL adalah:

```
CREATE DATABASE nama_database;
```

Contoh:

```
create database toko_online;
```

```
MariaDB [latihan]> create database toko_online;
Query OK, 1 row affected (0.05 sec)
```

Jika database berhasil dibuat, maka akan ada pesan Query OK. Database yang dibuat dapat dilihat dengan menggunakan perintah berikut:

```
show databases;
```

```
MariaDB [(none)]> show databases;
+-----+
| Database      |
+-----+
| colse         |
| db_ciperpus306 |
| information_schema |
| latihan       |
| mysql          |
| performance_schema |
| phpmyadmin     |
| test           |
| toko_online    |
+-----+
9 rows in set (0.00 sec)
```

Perintah diatas menampilkan database – database yang dibuat pada MySQL. Sebagai contoh, database toko\_online tampil pada daftar database MySQL.

## Menciptakan Tabel

Tabel merupakan obyek utama dalam database yang berfungsi sebagai tujuan penyimpanan data. Semua data dan informasi akan disimpan dalam tabel. Pembuatan tabel dapat dilakukan setelah database dibuat (diaktifkan), karena tabel akan pada database yang diaktifkan. Berikut adalah perintah untuk mengaktifkan tabel: `USE toko_online;`

```
MariaDB [(none)]> use toko_online;
Database changed
MariaDB [toko_online]>
```

“Database changed” merupakan tanda bahwa database `toko_online` telah aktif. Dalam keadaan ini, seluruh tabel yang dibuat akan masuk pada database `toko_online`.

Pada MySQL, tabel dibuat dengan perintah `CREATE TABLE`. Format penulisan perintah `CREATE TABLE` adalah sebagai berikut:

```
create table nama_table {
Nama_kolom1 tipe data constraint,
Nama_kolom2 tipe data constraint,
Nama_kolom3 tipe data constraint,
...
PRIMARY KEY (nama_kolom),
FOREIGN KEY (nama_kolom)
}
```

Ketentuan mengenai `CREATE TABLE` telah diulas lebih detail mengenai tipe data, atribut, constraint, penggunaan primary key dan foreign key pada Bab 9.

Sebagai contoh, pada praktik ini kita akan membuat tabel dengan studi kasus toko online. Tabel pertama yang akan kita buat adalah tabel produk dan tabel kategori. Berikut struktur tabel produk:

Nama Tabel	Nama Atribut	Tipe Data	PK / FK	Referensi Tabel
produk	id_produk	Integer	PK	
	id_kategori	Integer	FK	kategori
	nama_produk	Varchar		
	deskripsi	Text		
	harga	Integer		

	stok	Integer		
	tgl_masuk	Date		
	gambar	varchar		
kategori	id_kategori	Integer	PK	
	nama_kategori	Varchar		

Pada kasus ini, tabel kategori harus dibuat terlebih dahulu untuk menghindari kesalahan karena pada tabel produk terdapat kunci tamu (*foreign key*) yang didapat dari referensi tabel kategori.

```
MariaDB [toko_online]> create table kategori
-> (
-> id_kategori int not null,
-> nama_kategori varchar(30) not null,
-> primary key(id_kategori)
-> );
Query OK, 0 rows affected (0.13 sec)

MariaDB [toko_online]> show tables;
+-----+
| Tables_in_toko_online |
+-----+
| kategori             |
+-----+
1 row in set (0.00 sec)

MariaDB [toko_online]> describe kategori;
+-----+-----+-----+-----+-----+
| Field      | Type       | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+
| id_kategori | int(11)    | NO   | PRI | NULL    |       |
| nama_kategori | varchar(30) | NO   |     | NULL    |       |
+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)
```

Untuk melihat hasil dari CREATE TABLE kategori menggunakan perintah show tables, dimana perintah ini akan menampilkan tabel – tabel yang telah berhasil dibuat pada database toko online.

Jika ingin menampilkan struktur tabel yang telah dibuat, kita dapat menggunakan perintah desc atau describe kemudian disertakan nama tabelnya. Contoh diatas : describe kategori.

Tahap selanjutnya adalah pembuatan tabel produk. Format penulisannya sama seperti pembuatan tabel kategori, hanya saja pada tabel produk terdapat *foreign key* yang harus di lengkapi dengan perintah references.

```
MariaDB [toko_online]> create table produk
-> (
-> id_produk int not null,
-> id_kategori int not null,
-> nama_produk varchar(30) not null,
-> deskripsi text,
-> harga int not null,
-> stok int not null,
-> tgl_masuk date,
-> gambar varchar(50) not null,
-> primary key(id_produk),
-> foreign key(id_kategori) references kategori(id_kategori)
-> );
Query OK, 0 rows affected (0.48 sec)
```

Pada penulisan SQL diatas, perintah references disertakan dengan tujuan sebagai *link* ke tabel kategori.

## **Referensi**

- David M. Kroenke, D. J. (2016). *Databse Concepts*. Beijing: Pearson.
- Gillenson, M. (2011). *Fundamentals of Database Management Systems*. Memphis: Willey.
- Manisha Bharambe, U. S. (2017). *Database System*. Pune: Nirali Prakashan.
- Manisha Bharambe, V. G. (2017). *Advanced Relational Database System*. Pune: Nirali Prakashan.
- Oracle. (2020, 01 30). *Union, Minus, and Intersect: Databases for Developers*. Retrieved from Oracle Live SQL: [https://livesql.oracle.com/apex/livesql/file/tutorial\\_GPSAXIY7KMDX4ALZX654OXY6S.html](https://livesql.oracle.com/apex/livesql/file/tutorial_GPSAXIY7KMDX4ALZX654OXY6S.html)
- Singh, S. K. (2011). *Database System: Concepts, Design and Application*. Delhi: Pearson.
- Umakant Shirshetti, K. K. (2018). *Relational Database Management System*. Pune: Narali Prakashan.