

# **DIKTAT KULIAH**

## **IK 330 - BASIS DATA**

**Dosen:**  
**Budi Laksono Putro, S.Si, MT**



**ILMU KOMPUTER**  
**UNIVERSITAS PENDIDIKAN**  
**BANDUNG**  
**2012**



## DAFTAR ISI

DAFTAR ISI .....	i
<b>1    PENGENALAN BASIS DATA .....</b>	<b>1</b>
1.1. Data dan Informasi .....	2
1.2. Terminologi dan Konsep Basis Data .....	3
1.3. Perkembangan Basis Data .....	6
1.4. Sistem Basis Data (DBMS) .....	7
1.5. Model Basis Data .....	10
1.6. Basis Data Relational .....	11
1.7. Bahasa Basis Data .....	14
<b>2    ENTITY RELATIONSHIP MODEL .....</b>	<b>17</b>
2.1 Entitas dan Himpunan Entitas .....	18
2.2 Atribut .....	20
2.3 Relasi .....	22
2.4 Derajat Himpunan Relasi .....	23
2.5 Kardinalitas Relasi .....	24
2.6 Key .....	26
2.7 Diagram ER .....	27
2.8 Constraint Kardinalitas .....	28
<b>3    KONVERSI ER KE BASIS DATA RELATIONAL .....</b>	<b>32</b>
3.1 Himpunan Entitas Lemah .....	33
3.2 Spesialisasi dan Generalisasi .....	33
3.3 Agregasi .....	35
3.4 Ringkasan notasi simbol di ER .....	36
3.5 Penurunan skema ER ke Tabel .....	37
3.6 Representasi Atribut sebagai Kolom .....	37
3.7 Representasi Himpunan Entitas sebagai Tabel .....	38
3.7.1 Representasi Relasi (* pada kardinalitas N to N) .....	39
3.7.2 Representasi Spesialisasi (IS A) .....	40
3.7.3 Representasi Agregasi .....	42
<b>4    NORMALISASI .....</b>	<b>45</b>
4.1 Definisi Normalisasi .....	46
4.2 Tujuan Normalisasi .....	46
4.2.1 Update Anomaly .....	46
4.2.2 Insertion Anomaly .....	47
4.2.3 Deletion Anomaly .....	47
4.3 The Three Keys .....	48

4.4	Functional Dependencies .....	50
4.4.1	Partial Funcional Dependency .....	51
4.4.2	Transitive <i>Functional dependency</i> .....	52
4.5	Bentuk Normal dan Langkah-Langkah Normalisasi .....	53
4.5.1	Bentuk Normal Pertama (1st Normal Form) .....	53
4.5.2	Bentuk Normal Ke Dua (2nd Normal Form) .....	55
4.5.3	Bentuk Normal Ke Tiga (3rd Normal Form) .....	56
4.5.4	Bentuk Normal Boyce Codd (BC Normal Form) .....	58
4.5.5	Bentuk-Bentuk Normal Lainnya .....	58
4.6	Denormalisasi.....	58
<b>5</b>	<b>ALJABAR RELASIONAL .....</b>	<b>61</b>
5.1	Query dan Aljabar Relasional .....	62
5.2	Operasi Select .....	63
5.3	Operasi Project.....	64
5.4	Operasi Cartesian Product.....	64
5.5	Operasi Unio .....	65
5.6	Operasi Set Difference .....	66
5.7	Operasi Intersection .....	67
5.8	Operasi rename .....	67
5.9	Join.....	67
5.10	Fungsi Agregasi .....	69
5.11	Operasi Division .....	69
<b>6</b>	<b>BAHASA BASIS DATA .....</b>	<b>72</b>
6.1	Pendahuluan.....	73
6.2	Standarisasi SQL.....	73
6.3	Membangun Basisdata.....	75
6.3.1	Membuat BasisData .....	75
6.3.2	Membuat Tabel Data.....	75
6.3.3	Melakukan Perubahan pada Tabel .....	81
6.4	Maintenance Data pada Basisdata.....	84
6.4.1	Memasukan Data.....	84
6.4.2	Merubah Data.....	85
6.4.3	Menghapus Data .....	86
6.5	Mengakses Basisdata .....	86
6.5.1	Menganti Judul Kolom.....	87
6.5.2	Function pada SQL .....	88
6.5.3	Menentukan Kondisi .....	89
6.5.4	Menguruntukan Data.....	91
	<b>Daftar Pustaka .....</b>	<b>94</b>





# I PENGENALAN BASIS DATA



## Overview

---

Dalam kehidupan sehari-hari kita sering membahas mengenai data dan informasi. Informasi berasal dari kumpulan data yang disimpan secara terstruktur pada sebuah sistem yang dikenal dengan basis data (*database*). Pada bab awal ini akan dibahas tentang definisi, komponen sistem basis data, sistem *file*, abstraksi data, bahasa basis data, *database* administrator dan struktur sistem.



## Tujuan

---

1. Mahasiswa mengetahui dan mengerti konsep basis data.
2. Mahasiswa mengetahui mengenai komponen-komponen sistem basis data.
3. Mahasiswa mengetahui mengenai abstraksi data dan *Database Language*.

## 1.1 Data dan Informasi

Definisi data dan informasi dari beragam sumber maka dapat disimpulkan definisi dari data dan informasi sebagai berikut:

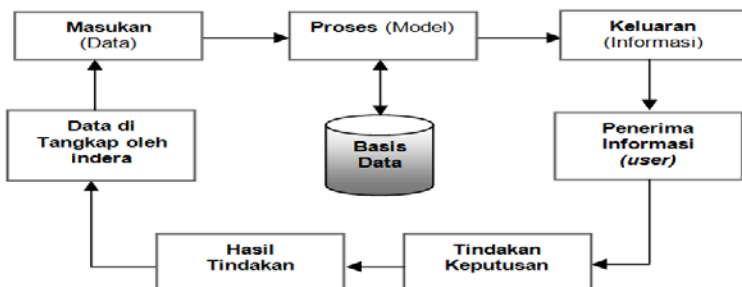
- **Data**, adalah respresantasi untuk mewakili nilai fakta dunia nyata. Representasi data dapat berupa nilai numerik, alphanumerik, gambar, suara, video, dan lain-lain. Fakta dunia nyata dapat berupa barang, kejadian, aktivitas, perasaan dan semua tentang dunia nya.
- **Informasi**, adalah data yang telah dikelola dalam bentuk tertentu untuk memberikan makna atau arti bagi penerimanya.

Kualitas data dan informasi dipengaruhi oleh hal-hal sebagai berikut:

- Benar merepresentasikan dunia nyata, dan
- Tepat waktu, dan
- Tepat penggunaan.

Pentingnya data dan informasi dapat dijelaskan dalam beberapa hal sebagai berikut:

- Data dan Informasi sebagai sebuah sumber daya penting, sama seperti Sumber Daya Manusia, Keuangan, dll.
- Data dan Informasi diperlakukan sebagai asset penting perusahaan/organisasi.
- Data dan Informasi nilainya akan bertambah bila dikelola dengan “baik dan semestinya”.
- Kegagalan dalam pengelolaan data akan berakibat ketimpangan dalam pengelolaan sumberdaya lainnya dalam organisasi dan berakibat negatif bagi jalannya organisasi atau bisnis



Gambar 1-1 Siklus informasi



**Siklus informasi** adalah merupakan gambaran secara umum mengenai proses terhadap data sehingga menjadi informasi yang bermanfaat bagi pengguna. Informasi yang menghasilkan informasi berikutnya. Demikian seterusnya proses pengolahan data menjadi informasi. Siklus informasi digambarkan pada gambar 1-1 dibawah ini.

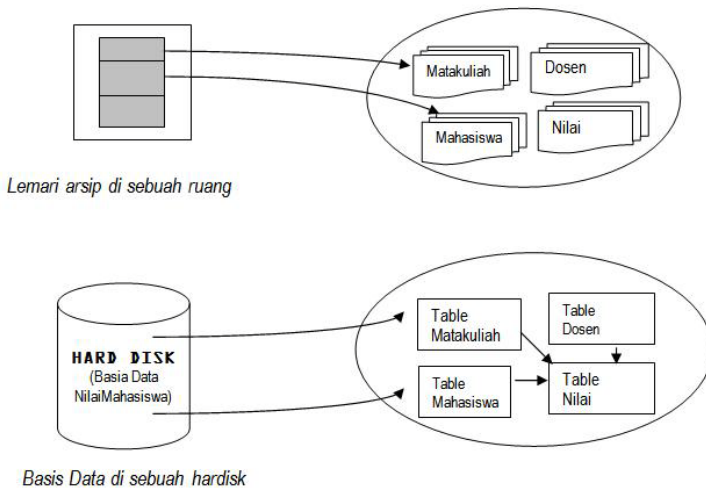
Data merupakan bentuk mentah yang belum dapat bercerita banyak, sehingga perlu diolah lebih lanjut. Data ditangkap sebagai *input*, diproses melalui suatu model membentuk informasi. Pemakai kemudian menerima informasi tersebut sebagai landasan untuk membuat suatu keputusan dan melakukan tindakan operasional yang akan membuat sejumlah data baru. Data baru tersebut selanjutnya menjadi *input* pada proses berikutnya, begitu seterusnya sehingga membentuk suatu siklus informasi/*Information Cycle* (Tata Sutabri, 2004: 17).

## 1.2 Terminologi dan Konsep Basis Data

Basis data terdiri dari 2 kata, yaitu basis & data. **Basis** dapat diartikan sebagai markas / gudang, tempat berkumpul. Sedangkan **data** adalah fakta yang mewakili suatu objek seperti manusia, barang, hewan peristiwa, keadaan dan sebagainya, yang direkam dalam bentuk angka, huruf simbol, teks gambar, bunyi atau kombinasinya.

Basis data sendiri dapat di definisikan dalam sejumlah sudut pandang seperti :

1. **Himpunan kelompok data/arsip** yang saling berhubungan yang diorganisasi sedemikian rupa agar kelak dapat dimanfaatkan kembali dengan cepat & mudah.
2. **Kumpulan data yang saling berhubungan yang disimpan** secara bersama sedemikian rupa dan tanpa pengulangan/ penumpukan (**redundansi**), untuk memenuhi berbagai kebutuhan.
3. **Kumpulan file/tabel/arsip yang saling berhubungan yang disimpan** dalam media **penyimpanan elektronik**.



**Gambar 1-2 Lemari Arsip dan Basis data**

Basis data dan lemari arsip sesungguhnya memiliki prinsip kerja dan tujuan yang sama. Prinsip utamanya adalah **pengaturan data/arsip**. Tujuan utama basis data dan lemari arsip adalah **kemudahan dan kecepatan dalam pengambilan kembali data/ arsip**. Perbedaan basis data dengan lemari arsip hanya terletak pada media penyimpanan yang digunakan. Lemari arsip menggunakan lemari sebagai media penyimpanannya, maka basis data menggunakan media penyimpanan elektronik seperti disk (disket, harddisk). Tidak semua bentuk penyimpanan data secara elektronik bisa disebut basis data. Basisdata adalah pengaturan, pemilahan, pengelompokkan, pengorganisasian data yang akan kita simpan sesuai fungsi/jenisnya. Pemilahan/ pengelompokkan ini dapat berbentuk sejumlah file/tabel terpisah atau dalam bentuk pendefinisian kolom-kolom/field-field data dalam setiap file/tabel.

Tujuan dibangunnya basis data adalah sebagai berikut :

- **Kecepatan & kemudahan (speed)**  
Dengan memanfaatkan basis data, memungkinkan kita untuk dapat menyimpan data atau melakukan perubahan/ manipulasi terhadap data atau menampilkan kembali data tersebut secara lebih cepat dan mudah.

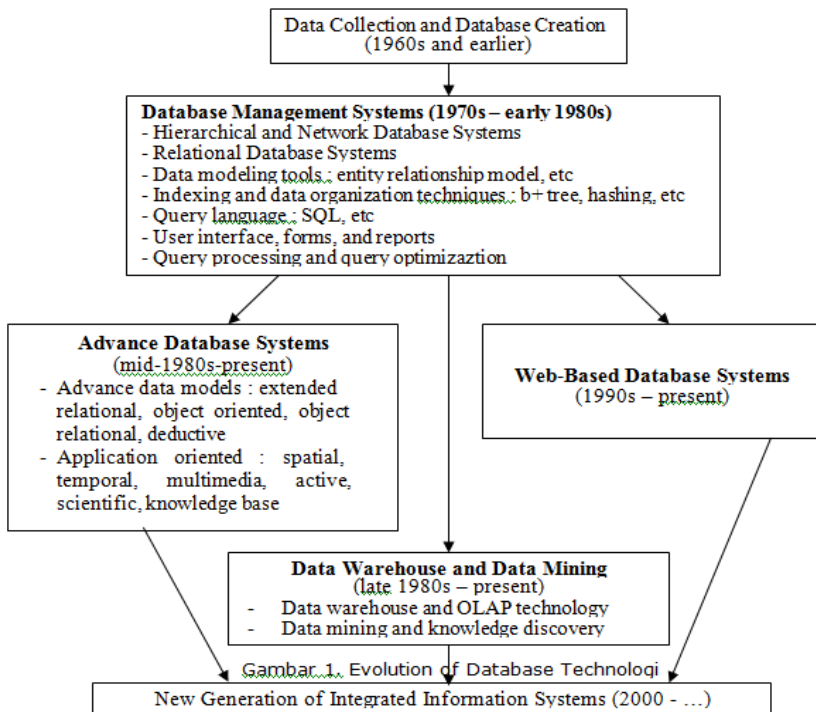
- **Efisiensi ruang penyimpanan (*space*)**  
Karena keterkaitan yang erat antara kelompok data dalam sebuah basisdata, maka redundansi (pengulangan) pasti akan selalu ada, sehingga akan memperbesar ruang penyimpanan. Dengan basisdata, efisiensi ruang penyimpanan dapat dilakukan dengan menerapkan sejumlah pengkodean, atau dengan membuat relasi-relasi antar kelompok data yang saling berhubungan.
- **Keakuratan (*accuracy*)**  
Pengkodean atau pembentukan relasi antar data bersama dengan penerapan aturan/batasan (*constraint*), domain data, keunikan data, dsb, yang secara ketat dapat diterapkan dalam sebuah basis data, sangat berguna untuk menekan ketidak akuratan penyimpanan data.
- **Ketersediaan (*availability*)**  
Dengan pemanfaatan jaringan komputer, maka data yang berada di suatu lokasi/cabang dapat juga diakses (*tersedia/available*) bagi lokasi/cabang lain.
- **Kelengkapan (*completeness*)**  
Kelengkapan data yang disimpan dalam sebuah database bersifat relatif, bisa jadi saat ini dianggap sudah lengkap, tetapi belum tentu pada suatu saat dianggap lengkap. Untuk mengakomodasi kelengkapan data, seperti
- **Keamanan (*security*)**  
Aspek keamanan dapat diterapkan dengan ketat, dengan begitu kita dapat menentukan pemakai basis data serta obyek-obyek didalamnya, serta jenis-jenis operasi apa saja yang boleh dilakukannya.
- **Kebersamaan pemakaian (*shareability*)**  
Basis data yang dikelola dengan aplikasi multi user dapat memenuhi kebutuhan ini.

#### Alasan mengapa mempelajari basisdata :

- **Perpindahan dari komputasi ke informasi**
- Himpunan elemen data semakin banyak dan beragam, sebagai contoh
  - ✓ perpustakaan digital.
  - ✓ Video interaktif
  - ✓ kebutuhan untuk memperluas DBMS
- DBMS mencakup bidang ilmu lain, sebagai contoh:
  - ✓ System operasi, bahasa pemrograman, teori komputasi, AI, logika, multimedia.

### 1.3 Perkembangan Basis Data

Perkembangan Basis Data dapat dipetakan dalam diagram pada gambar 1-3 dibawah ini.



**Gambar 1-3 Perkembangan Database**

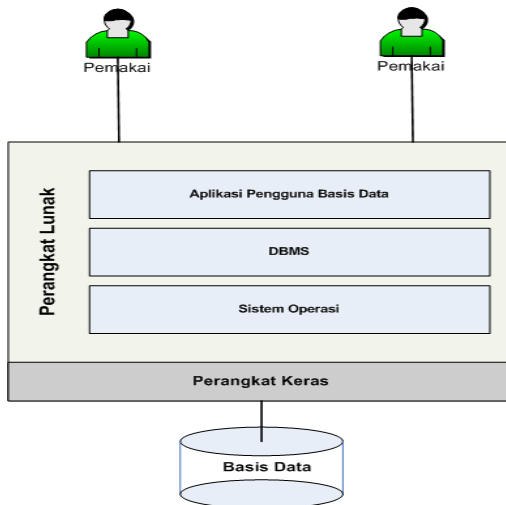
## 1.4 Sistem Basis Data (DBMS)

Perangkat lunak yang digunakan untuk mengelola dan memanggil kueri (*query*) basis data disebut sistem manajemen basis data (*Database Management System*, DBMS). DBMS memiliki karakteristik sebagai berikut:

- *Software program*
- *Supplements operating sistem*
- *Manages data*
- *Queries data and generates reports*
- *Data security*

Sistem basis data adalah sistem yang terdiri atas kumpulan *file-file* yang saling berhubungan dan dikelola oleh program (DBMS) yang memungkinkan beberapa pemakai dan atau program lain yang memiliki otoritas untuk mengakses dan memanipulasi data tersebut. Kelebihan pemakaian DBMS adalah:

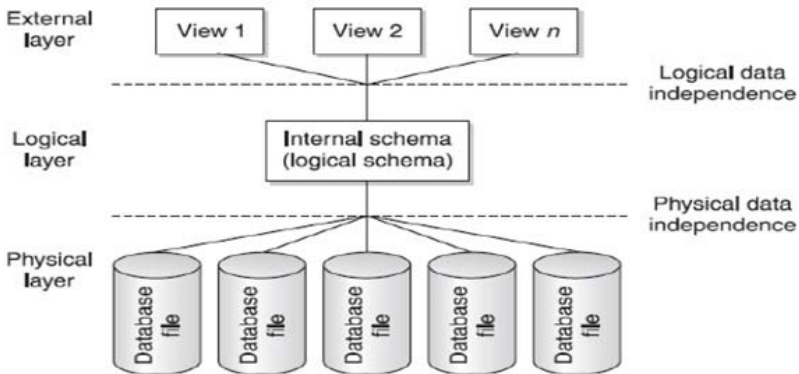
- *Data berdiri sendiri (Data Independence)*
- *Pengaksesan data efisien (Efficient data access)*
- *Integritas data dan keamanan terjamin (Data integrity and security)*
- *Administrasi data (Data administration)*
- *Dapat diakses bersamaan (Concurrent access )*
- *Recovery saat terjadi kegagalan (Crash recovery)*
- *Mengurangi waktu pembangunan aplikasi (Reduced application development time)*



**Gambar 1-4 Komponen DBMS**

Komponen-komponen pada sebuah sistem basis data gambar1-4 terdiri antara lain:

- Perangkat keras
- Sistem operasi
- Basis data
- DBMS (*Database Management System*)
- Pemakai
- Aplikasi lain



**Gambar 1-5 Abstraksi Data.**

Tujuan utama dari sistem basis data adalah untuk menyediakan fasilitas untuk *view* data secara abstrak bagi penggunanya. Namun bagaimana sistem menyimpan dan mengelola data tersebut, hanya diketahui oleh sistem itu sendiri. Abstraksi data merupakan level dalam bagaimana melihat data dalam sebuah sistem basis data. Berikut ini tiga level abstraksi data:

1. **Level fisik**

Merupakan level terendah pada abstraksi data yang menunjukkan bagaimana sesungguhnya data disimpan. Pada level ini pemakai melihat data sebagai gabungan dari struktur dan datanya sendiri.

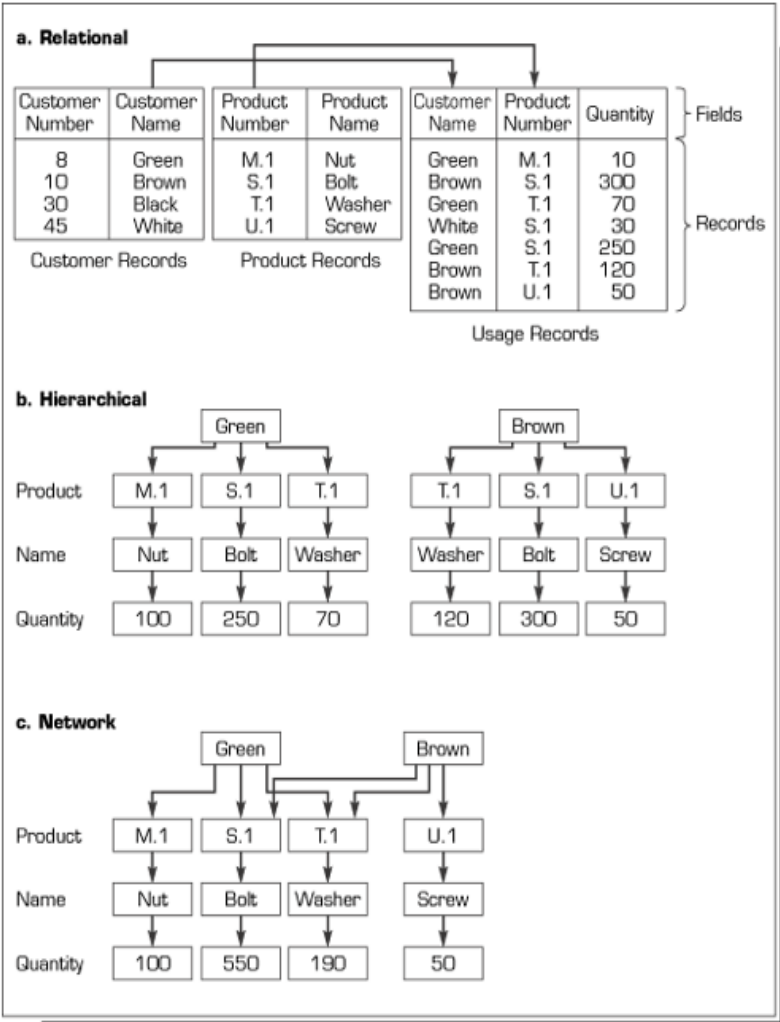
2. **Level logik**

Merupakan level berikutnya pada abstraksi data, menggambarkan data apa yang disimpan pada basis data dan hubungan apa saja yang ada di antara data tersebut.

3. **Level view**

Merupakan level tertinggi dari abstraksi data yang hanya menunjukkan sebagian dari basis data. Banyak *user* dalam sistem basis data tidak akan terlibat dengan semua data atau informasi yang ada atau yang disimpan. Para *user* umumnya hanya membutuhkan sebagian data atau informasi dalam basis data yang kemunculannya di mata *user* diatur oleh aplikasi *end user*.

1.5 Model Basis Data



Gambar 1-2 Contoh Model Database



Basis Data memiliki banyak model, penggambaran model basis data digambarkan pada gambar 1-5 dengan penjelasan dibawah ini:

- a. **Relational**  
Model ini direpresentasikan dalam tabel dua dimensi, tabel-tabel tersebut memiliki hubungan yang disebut dengan relasi. Model ini memiliki fleksibilitas dan kecepatan yang tinggi.
- b. **Hierarchical**  
Memiliki struktur pohon dimana *field* hanya memiliki satu buah induk (*parent*), masing-masing *parent* memiliki banyak *child* (anak). Model ini memiliki kecepatan yang baik.
- c. **Network**  
*Relationship* dibuat menggunakan *linked list (pointer)*. Berbeda dengan model *hierarchical* satu anak dapat memiliki beberapa induk. Model ini memiliki fleksibilitas yang tinggi.
- d. **Object oriented**  
*Object Oriented Database* adalah sebuah sistem *database* yang menggabungkan semua konsep *object oriented* seperti pewarisan, abstraksi, enkapsulasi, dll. Model ini dapat berinteraksi dengan baik dengan bahasa pemrograman berorientasi objek seperti java dan C++.

## 1.6 Basis Data Relasional

Model Relasional merupakan model yang paling sederhana sehingga mudah digunakan dan dipahami oleh pengguna. Model ini menggunakan sekumpulan tabel berdimensi dua ( yang disebut relasi atau tabel ), dengan masing-masing relasi tersusun atas tupel atau baris dan atribut. DBMS yang bermodelkan relasional biasa disebut RDBMS (*Relational Data Base Management System*). Model database ini dikemukakan pertamakali oleh EF codd, seorang pakar basisdata. Model ini sering disebut juga dengan database relasi.

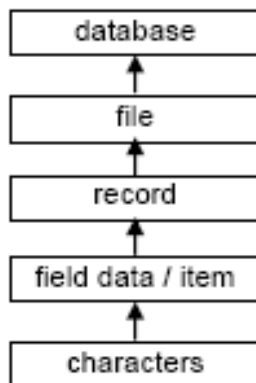
Model database hirarki dan jaringan merupakan model database yang tidak banyak lagi dipakai saat ini, karena adanya berbagai kelemahan dan hanya cocok untuk struktur hirarki dan jaringan saja. Artinya tidak mengakomodir untuk berbagai macam jenis persoalan dalam suatu sistem database.

Model database relasi merupakan model database yang paling banyak digunakan saat ini, karena paling sederhana dan mudah digunakan serta yang paling penting adalah kemampuannya dalam mengakomodasi berbagai kebutuhan pengelolaan database. Sebuah database dalam model ini disusun

dalam bentuk tabel dua dimensi yang terdiri dari baris (*record*) dan kolom (*field*), pertemuan antara baris dengan kolom disebut **item data** (*data value*), table-table yang ada di hubungan (*relationship*) sedemikian rupa menggunakan **field-field kunci** (*Key field*) sehingga dapat meminimalkan duplikasi data.

### Tingkatan Data Dalam Database Relasi

Dalam suatu sistem database relasi, data yang tersimpan dalam DBMS mempunyai tingkatan-tingkatan gambar 1-6, sebagai berikut :



**Gambar 1-6 Tingkatan data dalam Database Relasional**

- **Karakter (Characters)**

Merupakan bagian terkecil dalam database, dapat berupa karakter numerik (angka 0 s.d 9), huruf (A - Z, a - z) ataupun karakter-karakter khusus, seperti \*, &, %, # dan lain-lain.

- **Field atau Attribute**

Merupakan bagian dari record yang menunjukkan suatu item data yang sejenis, Misalnya : field nama, file NIM dan lain sebagainya. Setiap field harus mempunyai nama dan tipe data tertentu. Isi dari field di sebut Data Value. Dalam table database, field ini disebut juga kolom.

- **Record atau Tuple**

Tuple/Record adalah kumpulan data value dari attribute yang berkaitan sehingga dapat menjelaskan sebuah entity secara lengkap. Misal : Record

entity mahasiswa adalah kumpulan data value dari field nobp, nama, jurusan dan alamat per-barisnya. Dalam tabel database, Record disebut juga **baris**.

- **Table/Entity**

Entity merupakan sesuatu yang dapat diidentifikasi dari suatu sistem database, bisa berupa objek, orang, tempat, kejadian atau konsep yang informasinya akan disimpan di database. Misal. Pada sistem database akademik, yang menjadi entity adalah, mahasiswa, dosen, matakuliah dan lain-lain. Dalam aplikasi nantinya, penggunaan istilah Entity sering di samakan dengan istilah Tabel. (Entity = table). Disebut tabel, karena dalam merepresentasikan datanya di atur dalam bentuk baris dan kolom. Baris mewakili 1 record dan kolom mewakili 1 field. Dalam sistem database tradisional, entity/table ini disebut juga dengan file.

- **Database**

Kumpulan dari tabel-tabel yang saling berelasi, disusun secara logis, sehingga menghasilkan informasi yang bernilai guna dalam proses pengambilan keputusan.

Ada beberapa sifat yang melekat pada suatu tabel :

- ✓ Tidak boleh ada record yang sama (kembar)
- ✓ Urutan record tidak terlalu penting, karena data dalam record dapat diurut sesuai dengan kebutuhan.
- ✓ Setiap field harus mempunyai nama yang unik (tidak boleh ada yang sama).
- ✓ Setiap field mesti mempunyai tipe data dan karakteristik tertentu

Contoh produk DBMS terkenal yang menggunakan model relasional antara lain adalah :

1. DB2 (IBM)
2. Rdb/VMS (Digital Equipment Corporation)
3. Oracle (Oracle Corporation)
4. Informix (Informix Corporation)
5. Ingres (ASK Group Inc)
6. Sybase (Sybase Inc)
7. Dan masih banyak lagi

### 1.7. Bahasa Basis Data

Macam-macam perintah pada bahasa basis data dapat dikelompokkan menjadi:

1. *Data Definition Language (DDL)*

DDL adalah perintah-perintah yang biasa digunakan oleh pengguna basis data untuk mendefinisikan skema ke DBMS. Skema adalah deskripsi lengkap tentang struktur medan, rekaman, dan hubungan data pada basis data. Index merupakan suatu mekanisme yang lazim digunakan pada basis data, yang memungkinkan pengambilan data dapat dilakukan dengan cepat. DDL Digunakan untuk mespesifikasikan struktur/skema basis data yang menggambarkan/mewakili desain basis data secara keseluruhan. Hasil kompilasi perintah DDL adalah kamus data. Kamus data merupakan sebuah *file* yang berupa metadata, yaitu data yang mendeskripsikan data sesungguhnya. Kamus data ini akan selalu diakses pada suatu operasi basis data sebelum suatu *file* data yang sesungguhnya diakses.

2. *Interactive Data Manipulation Language (DML)*

DML adalah perintah-perintah yang digunakan untuk mengubah, manipulasi dan mengambil data pada basis data. Tindakan seperti menghapus, mengubah, dan mengambil data menjadi bagian dari DML.

DML merupakan bahasa yang memungkinkan *user* untuk mengakses atau memanipulasi data sebagaimana telah direpresentasikan oleh model data.

Terdapat dua macam DML, yaitu:

- Prosedural, mengharuskan *user* untuk menentukan data apa yang dibutuhkan dan bagaimana untuk mendapatkan data tersebut.
- Nonprosedural, mengharuskan pemakai untuk menentukan data apa yang dibutuhkan tanpa menyebutkan bagaimana mendapatkan data tersebut.

3. *Transaction control*

*Transaction control* adalah bahasa basis data yang mengatur transaksi yang dilakukan oleh Data Manipulation Language (DML). *Transaction control* ini memiliki peran yang sangat besar untuk menentukan dilakukan atau tidaknya perubahan-perubahan data yang ada pada basis data. Contoh dari *transaction control* ini adalah perintah *commit* dan *rollback*.

4. *Embedded and Dinamic SQL*, contoh C, C++, Java, Cobol, Pascal, etc.

Tidak semua DBMS memiliki fasilitas ini, salah satu contoh DBMS yang memiliki fasilitas ini adalah oracle dimana oracle dapat me-load class yang ditulis menggunakan bahasa pemrograman java kedalam database.

5. *Authorization*, untuk mendefinisikan hak akses spesifik terhadap objek-objek basis data.



## Rangkuman

1. **Data**, adalah representasi untuk mewakili nilai fakta dunia nyata. Representasi data dapat berupa nilai numerik, alphanumerik, gambar, suara, video, dan lain-lain. Fakta dunia nyata dapat berupa barang, kejadian, aktivitas, perasaan dan semua tentang dunia nyata. **Informasi**, adalah data yang telah dikelola dalam bentuk tertentu untuk memberikan makna atau arti bagi penerimanya.
2. Data dan informasi akan saling berkesinambungan sehingga membentuk suatu siklus yang disebut *information cycle* (siklus informasi).
3. Basis data sendiri dapat di definisikan dalam sejumlah sudut pandang seperti :
  1. Himpunan kelompok data/arsip yang saling berhubungan yang diorganisasi sedemikian rupa agar kelak dapat dimanfaatkan kembali dengan cepat & mudah.
  2. Kumpulan data yang saling berhubungan yang disimpan secara bersama sedemikian rupa dan tanpa pengulangan/penumpukan (redundansi), untuk memenuhi berbagai kebutuhan.
  3. Kumpulan file/tabel/arsip yang saling berhubungan yang disimpan dalam media penyimpanan elektronik.
4. Tujuan dibangunnya basis data adalah sebagai berikut :
  - Kecepatan & kemudahan (*speed*)
  - Efisiensi ruang penyimpanan (*space*)
  - Keakuratan (*accuracy*)
  - Ketersediaan (*availability*)
  - Kelengkapan (*completeness*)
  - Keamanan (*security*)
  - Kebersamaan pemakaian (*shareability*)
4. Pengelolaan data dan informasi menggunakan DBMS memiliki keuntungan jika dibandingkan dengan menyimpannya menggunakan *file*.
5. Basis data adalah penyimpanan kumpulan informasi secara sistematis dalam sebuah komputer sehingga dapat diperiksa menggunakan suatu program komputer untuk memperoleh informasi dari basis data tersebut. Perangkat lunak yang digunakan untuk mengelola dan memanggil kueri

(*query*) basis data disebut sistem manajemen basis data (*Database Management System*, DBMS).

6. Terdapat tiga level abstraksi data yaitu dari yang paling dasar adalah level fisik, level logik, dan level view.
7. Basis data relasional memiliki satu struktur “logik” yang disebut *Relation* (relasi). Struktur relasi merupakan struktur data 2-dimensi dan pada level “fisik” berupa *table* (tabel).

## 2 *ENTITY RELATIONSHIP* MODEL



---

### Overview

---

Model Entity-Relationship adalah model data konseptual tingkat tinggi untuk perancangan basis data. Model data konseptual adalah himpunan konsep yang mendeskripsikan struktur basis data, transaksi pengambilan dan pembaruan basis data.

Model ER adalah data konseptual tak tergantung DBMS dan platform perangkat keras tertentu. Model ER dikemukakan oleh Chen [1976]. Sejak itu, telah memperoleh banyak perhatian dan perluasan. Konsep utama dari pemodelan ini berakar pada Entitas entitas dan relasi antar entitas. Pada bab ini akan di bahas pengertian entitas, atribut dan relasi, derajat himpunan relasi, kardinalitas relasi, serta pengenalan *key* sebagai salah satu *constraint* dalam ER.



---

### Tujuan

---

1. Mahasiswa memahami definisi salah pemodelan konseptual basis data menggunakan ER.
2. Mahasiswa memahami Konsep Entitas, Atribut, Relasi dan Kardinalitas relasi.

Hal pertama yang harus dilakukan dalam membuat basis data adalah mendesain tabel-tabel yang akan digunakan untuk menyimpan data sesuai bisnis proses yang kita inginkan. Proses konseptual adalah pandangan secara konsep tentang basis data. Pandangan konseptual ini tentunya harus bisa diimplementasikan kedalam bentuk tabel, karena basis data relasional hanya mengenal tabel.

Model Entity-Relationship adalah model data konseptual tingkat tinggi untuk perancangan basis data. Model data konseptual adalah himpunan konsep yang mendeskripsikan struktur basis data, transaksi pengambilan dan pembaruan basis data. Model ER adalah data konseptual tak tergantung DBMS dan platform perangkat keras tertentu. Model ER dikemukakan oleh Chen [1976]. Sejak itu, telah memperoleh banyak perhatian dan perluasan. Model ER adalah persepsi terhadap dunia nyata sebagai terdiri objek-objek dasar yang disebut entitas dan keterhubungan (*relationship*) antar entitas-entitas itu. Konsep paling dasar di model ER adalah entitas, relationship dan atribut.

Komponen-komponen utama model ER adalah:

1. **Entitas (*entity*)**, Entitas memodelkan objek-objek yang berada diperusahaan/lingkungan.
2. **Relationship**. Relationship memodelkan koneksi/hubungan di antara entitas-entitas.
3. **Atribut-atribut (*properi-properti*)**, memodelkan properti-properti dari entitas dan relationship.
4. **Konstrain-konstrain (*batasan-batasan*)** integritas, konstrain-konstrain ketentuan validitas.

## 2.1 Entitas (*entity*) dan Himpunan Entitas (*Entitas Sets*)

Entitas merupakan individu yang mewakili sesuatu yang nyata (eksistensinya) dan dapat dibedakan dari sesuatu yang lain. Sebuah kursi yang kita duduki, seseorang yang menjadi pegawai di sebuah perusahaan dan sebuah mobil yang melintas di depan kita adalah entitas.

Sekelompok entitas yang sejenis dan berada dalam lingkup yang sama membentuk sebuah himpunan entitas (*entity sets*). Sederhananya, entitas menunjuk pada individu suatu objek, sedang himpunan entitas menunjuk pada rumpun (*family*) dari individu tersebut.

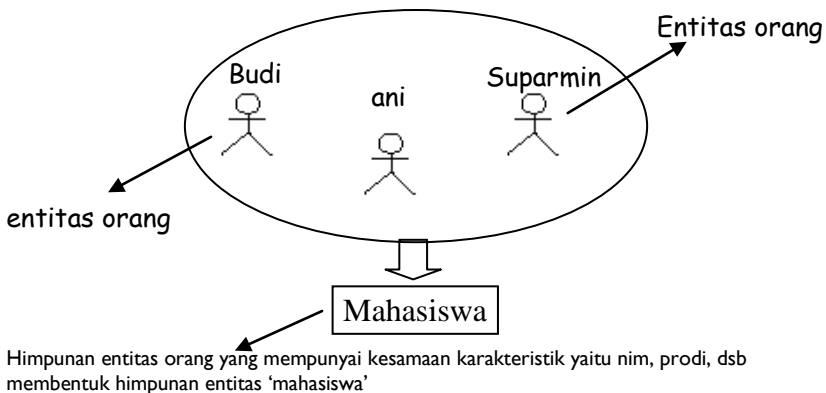
Dalam berbagai pembahasan/literature, penyebutan himpunan entitas (yang kurang praktis) ini seringkali digantikan dengan sebutan entitas saja. Karena itu sering ditemui, penggunaan istilah entitas (*entity*) di sebuah literature



sebenarnya menunjuk pada himpunan entitas. Entitas adalah objek yang dirasa penting di sistem tersebut, yg bisa berupa :

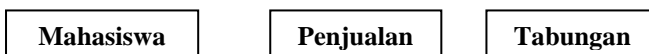
- Objek Konkrit  
Contoh : Orang, Mobil, Buku
- Objek Abstrak  
Contoh : Penjualan, Jadwal, Pinjaman, Tabungan

Budi adalah salah satu contoh dari entitas. Sedangkan budi, ani, suparmin merupakan himpunan entitas orang. Dapat kita katakan bahwa **Himpunan Entitas (Entity Set)**: Sekelompok entitas yang sejenis dan berada dalam lingkup yang sama. Kumpulan entitas orang dengan karakteristik mempunyai nim, prodi, dsb bisa kita katakan merupakan himpunan entitas mahasiswa. **Entitas menunjuk kepada pada individu** suatu objek sedangkan **himpunan entitas menunjuk pada rumpun (family) dari individu tersebut**.



**Gambar 2-1 Himpunan Entitas Mahasiswa**

Sebuah entitas/himpunan entitas dapat di gambarkan/dinotasikan dengan **sebuah gambar persegi panjang**. Berikut merupakan contoh entitas mahasiswa, jadwal dan pinjaman.



**Gambar 2-2 Contoh Himpunan Entitas**

Setiap entitas mempunyai **atribut** yang melekat pada entitas tersebut. Berikut gambaran konseptual basis data (\* entitas dan atribut) yang direfleksikan kedalam bentuk fisik dari basis data (\* tabel dan kolom).



**Gambar 2-3 Gambaran Himpunan entitas di Tabel**

## 2.2 Atribut

Setiap entitas pasti memiliki atribut yang mendeskripsikan karakteristik (property) dari entitas tersebut.

Penentuan / pemilihan atribut-atribut yang relevan bagi sebuah entitas merupakan hal penting lainnya dalam pembentukan model ER. Contoh: nim, nama, alamat, kode.

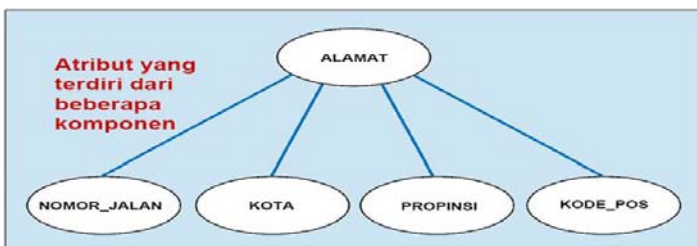
Setiap atribut mempunyai *domain value set* yaitu batasan batasan yg dibolehkan bagi suatu atribut.

Tipe-tipe atribut dapat dibedakan.

- **Simple dan Composite**

Atribut **Simple** yaitu suatu atribut yang **tidak bisa dibagi** menjadi bagian yg lebih kecil lagi. Contoh atribut *simple* adalah Jenis Kelamin.

Atribut **Composite** yaitu suatu atribut yang **dapat di bagi** menjadi beberapa bagian. Contoh atribut *composite* Nama dapat di bagi menjadi nama depan dan nama belakang.



**Gambar 2-4 Contoh Atribut Komposit**

- **Single value dan multivalued**

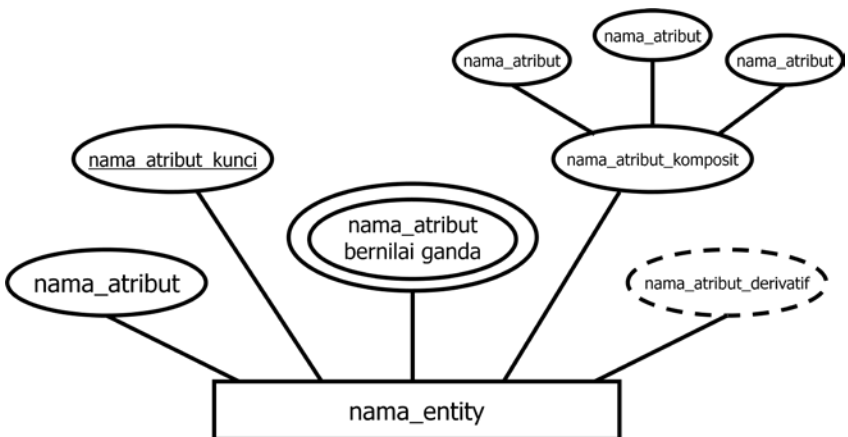
Atribut **Single value** yaitu suatu atribut yang bisa di isi paling banyak 1 nilai untuk setiap baris data. Contoh atribut *single value* adalah Jenis Kelamin.

Atribut **Multivalued** yaitu suatu atribut yang bisa lebih dari 1 nilai yang sejenis untuk setiap baris data. Contoh atribut multivalued value adalah Alamat, No telp dan hobi. Ketiga atribut tersebut bisa berisi lebih dari 1. Contoh untuk 1 entitas orang bisa mempunyai lebih dari 1 nilai untuk atribut hobi yang isinya musik, olahraga begitu juga untuk telp dan alamat (\* karena bisa mempunyai > 1 no telp dan > 1 alamat)

- **Derived attribute**

**Derived Attribute** yaitu suatu atribut yang nilainya didapatkan dari hasil pengolahan atribut lain. Contoh atribut *derived* adalah umur yaitu didapatkan dari perhitungan tanggal lahir dan tanggal sekarang. IPK yang didapatkan dari penjumlahan nilai di bagi dengan jumlah sks yang diambil.

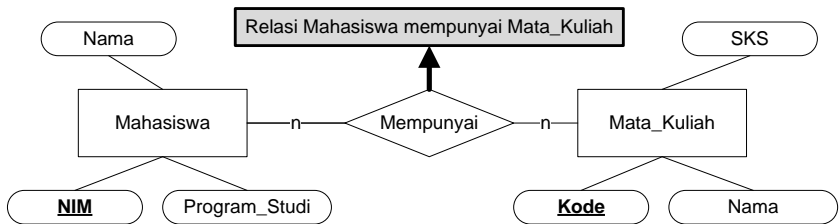
Notasi **atribut** digambarkan dengan gambar **elips**. **Atribut kunci** biasa di beri tanda # atau **garis bawah**. Contoh himpunan entitas mahasiswa mempunyai atribut nim sebagai *key*, prodi, nama, dsb



**Gambar 2-5 Entitas dengan Atribut**

### 2.3 Relasi

ER menggambarkan entitas-entitas dengan atributnya yang saling berelasi. **Relasi** menggambarkan hubungan antara entitas satu dengan entitas yang lain sesuai dengan proses bisnisnya. Notasi relasi didalam diagram ER digambarkan dengan notasi **belah ketupat**.



**Gambar 2-6 Relasi di gambarkan dengan belah ketupat**

Relasi menunjukkan adanya hubungan di antara sejumlah entitas yang berasal dari himpunan entitas yang berbeda, sebagai contoh ERD gambar 2-6 sebagai berikut:

Misalnya, entitas seorang Mahasiswa dengan

nim = '980001' dan

nama = 'Ali Akbar' (yang ada di himpunan entitas Mahasiswa)

mempunyai relasi dengan entitas sebuah Mata\_Kuliah dengan

kode = 'IK-330' dan

nama = 'Basis Data'.

Relasi diantara kedua entitas tadi mengandung arti bahwa mahasiswa tersebut sedang mengambil/mempelajari mata kuliah tersebut di sebuah perguruan tinggi yang ditinjau.

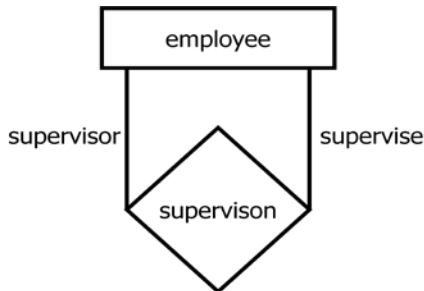
Kumpulan semua relasi diantara entitas-entitas yang terdapat pada himpunan entitas-himpunan entitas tersebut membentuk himpunan relasi (*relationship sets*).

Sebagaimana istilah himpunan entitas yang banyak sekali disingkat menjadi entitas, istilah himpunan relasi jarang sekali digunakan dan lebih sering disingkat dengan istilah relasi saja.

## 2.4 Derajat Himpunan Relasi

Jika dilihat dari **jumlah entitas** yang dihubungkan oleh sebuah relasi, maka kita bisa membagi menjadi 3 macam:

- **Unary** (Hanya me-relasi-kan 1 entitas)



**Gambar 2-7 Contoh Derajat Relasi *Unary***

Relasi di atas menggambarkan entitas **employee** yang ber-relasi dengan entitas **employee**. Entitas karyawan bisa merupakan supervise biasa tetapi bisa juga merupakan supervisor. Relasi yang terjadi yaitu relasi **employee supervision** untuk **employee** (\* entitas manajer adalah salah satu karyawan juga). Perhatikan kardinalitas relasinya, 1 **employee** hanya bekerja untuk 1 manajer, tetapi 1 manajer bisa mempunyai banyak bawahan.

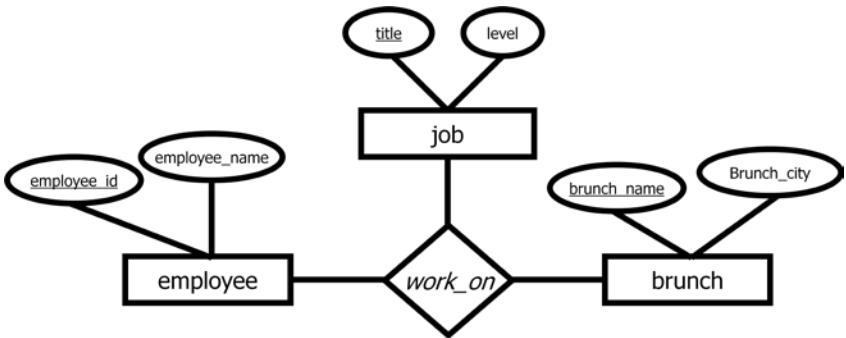
- **Binary** (Me-relasi-kan 2 entitas)



**Gambar 2-8 Contoh Derajat Relasi *Binary***

Relasi di atas menggambarkan entitas pelanggan yang ber-relasi dengan entitas pinjaman. 1 pelanggan bisa mempunyai banyak nomor pinjaman, dan 1 nomor pinjaman hanya untuk 1 pelanggan.

▪ **Ternary** (Me-relasi-kan 3 entitas)



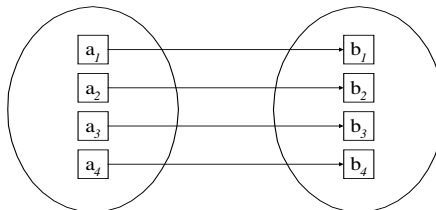
**Gambar 2-9 Contoh Derajat Relasi Ternary**

Relasi di atas menggambarkan entitas **employee** yang ber-relasi dengan entitas **branch** dan entitas **job** melalui relasi **work\_on**. 1 employee bekerja di sebuah title tertentu dan juga bekerja di sebuah brunch\_name tertentu. Ada 3 entitas yang terlibat dari relasi di atas

## 2.5 Kardinalitas Relasi

Kardinalias relasi menggambarkan banyaknya jumlah maksimum entitas dapat ber-relasi dengan entitas pada himpunann entitas yang lain. Pada himpunan **relasi biner**, pemetaan kardinalitas relasi dapat berupa salah satu dari pilihan berikut :

▪ **Satu ke Satu**

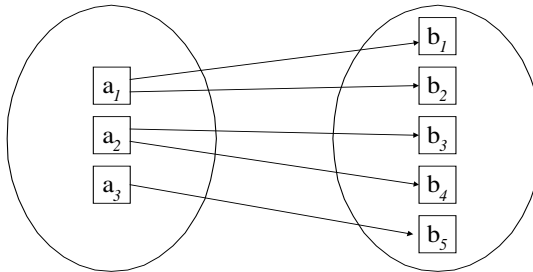


**Gambar 2-10 Relasi dengan Kardinalitas 1 ke 1**

Relasi di atas menggambarkan bahwa untuk setiap entitas di himpunan entitas A berpasangan dengan maksimal 1 entitas di himpunan entitas B. Dari A ke

B kardinalitasnya maksimal 1, dan dari B ke A kardinalitasnya maksimal 1. Oleh karena itu relasi ini berkardinalitas 1 ke 1.

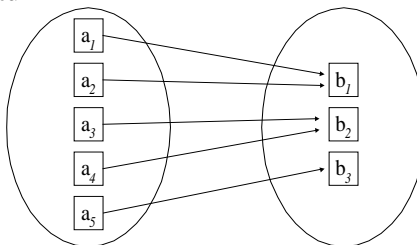
- **Satu ke Banyak**



**Gambar 2-11 Relasi dengan Kardinalitas 1 ke Banyak**

Relasi di atas menggambarkan bahwa untuk setiap entitas di himpunan entitas A berpasangan dengan banyak entitas di himpunan entitas B. Dari A ke B kardinalitasnya maksimal adalah banyak, dan dari B ke A kardinalitasnya maksimal 1. Oleh karena itu relasi ini berkardinalitas 1 ke banyak.

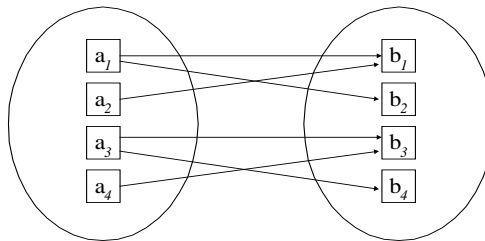
- **Banyak ke Satu**



**Gambar 2-12 Relasi dengan Kardinalitas Banyak ke 1**

Relasi di atas menggambarkan bahwa untuk setiap entitas di himpunan entitas A berpasangan dengan maksimal 1 entitas di himpunan entitas B. Dari A ke B kardinalitasnya maksimal adalah 1, dan dari B ke A kardinalitasnya maksimal adalah banyak. Oleh karena itu relasi ini berkardinalitas banyak ke 1.

### ▪ Banyak ke Banyak



**Gambar 2-13 Relasi dengan Kardinalitas Banyak ke Banyak**

Relasi di atas menggambarkan bahwa untuk setiap entitas di himpunan entitas A berpasangan dengan maksimal banyak entitas di himpunan entitas B. Dari A ke B kardinalitasnya maksimal adalah banyak, dan dari B ke A kardinalitasnya maksimal adalah banyak. Oleh karena itu relasi ini berkardinalitas banyak ke banyak.

## 2.6 Key

Penggunaan *key* merupakan cara untuk membedakan suatu entitas didalam himpunan entitas dengan entitas lain. *Key* dipilih karena unik, untuk setiap entitas sehingga bisa di bedakan dari entitas yang lain. Kita bisa mendefinisikan *key* sebagai **satu atau gabungan dari beberapa atribut yang dapat membedakan semua row dalam relasi secara unik.**

Macam *key* ada 3 yaitu :

### ▪ Superkey

*Superkey* yaitu satu atau lebih atribut (kumpulan atribut) yang dapat membedakan satriap baris data dalam sebuah relasi secara unik. Contoh super *key* yaitu =

- Nim, nama, alamat, kota
- Nim, nama, alamat
- Nim, nama
- Nim

### ▪ Candidate key

Kumpulan atribut minimal yang dapat membedakan setiap baris data dalam sebuah relasi secara unik. Contoh Nim



### ▪ **Primary key**

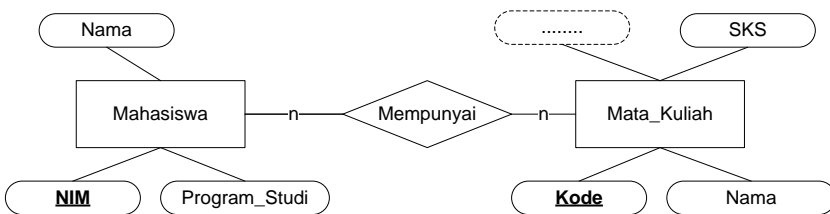
*Primary key* merupakan salah satu dari *candidate key* yang terpilih. Alasan pemilihan *primary key* :

- Lebih sering di jadikan acuan
- Lebih ringkas
- Jaminan keunikan *key* lebih baik

Contoh dari *primary key* adalah Nim

## 2.7 Diagram ER

Merupakan diagram model konseptual untuk menggambarkan struktur *logis* dari basisdata berbasis grafis.



**Gambar 2-14 Contoh Diagram ER**

Notasi yang digunakan di Diagram ER adalah :

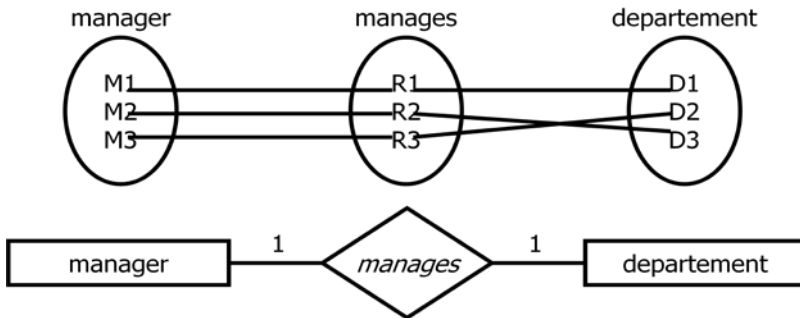
**Garis** : Link yang menghubungkan antara Entitas dengan atribut, dan entitas dengan relasi atau entitas

**Elips dobel** : Menunjukkan atribut yang *multivalued*

**Elips dengan garis terputus** : Menunjukkan atribut turunan

## 2.8 Constraint Kardinalitas

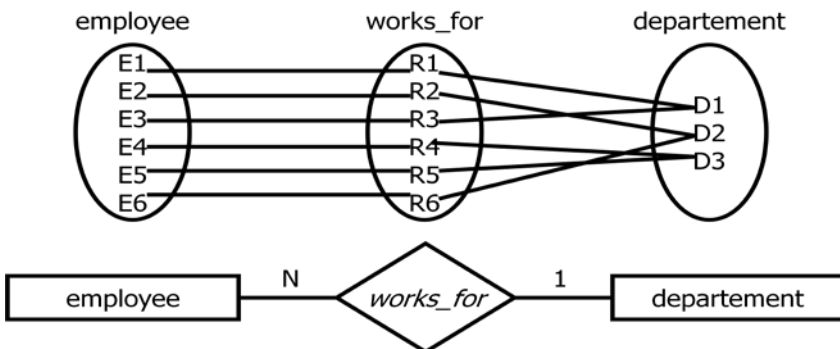
Dalam menggambarkan kardinalitas pada Diagram ER, digunakan **1** yang menunjukkan **Satu** atau **n** yang menunjukkan **Banyak**.



Gambar 2-15 Relasi 1 ke 1

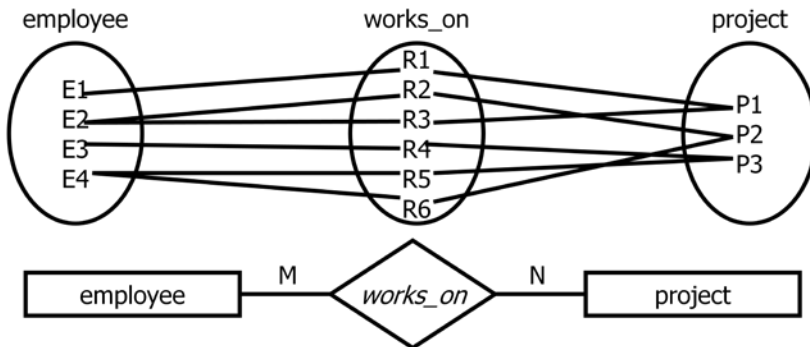
Menjelaskan jumlah keterhubungan satu entity dengan entity yang lainnya gambar 2-16. **(1:1)** : satu entitas pada tipe entitas A berhubungan dengan paling banyak satu entitas pada tipe entitas B dan juga sebaliknya.

Contoh : seorang manager hanya memimpin satu departemen dan begitu sebaliknya.



Gambar 2-16 Relasi 1 ke banyak

(1 : N / N : 1) : suatu entitas di A dihubungkan dengan sejumlah entitas di B  
gambar 2-17. Contoh : banyak karyawan berkerja untuk satu departement atau satu departement memiliki banyak karyawan yang bekerja untuknya.



**Gambar 2-18 Relasi Banyak ke Banyak**

(M : N) : setiap entitas A dapat berhubungan dengan banyak entitas B dan sebaliknya setiap entitas B juga dapat berhubungan dengan banyak entitas A  
gambar 2-18. Contoh : satu proyek mempunyai banyak karyawan, satu karyawan boleh bekerja di beberapa proyek.

## 2.9 Tahapan Pembuatan ER Diagram

Dapat disimpulkan untuk membuat ER Diagram, dapat mengikuti urutan tahapan berikut:

1. Mengidentifikasi dan menetapkan seluruh himpunan entity yang akan terlibat.
2. Menentukan atribut-atribut dari setiap entity.
3. Menentukan atribut primary key dari setiap entity.
4. Menentukan relationship antar entity.
5. Menentukan atribut-atribut dari setiap relationship (jika ada).
6. Menentukan Cardinality Rasio.
7. Menentukan Participation Constraint.



## Rangkuman

1. Model Entity-Relationship adalah model data konseptual tingkat tinggi untuk perancangan basis data. Model data konseptual adalah himpunan konsep yang mendeskripsikan struktur basis data, transaksi pengambilan dan pembaruan basis data. Model ER adalah data konseptual tak tergantung DBMS dan platform perangkat keras tertentu. Model ER dikemukakan oleh Chen [1976].
2. Diagram ER Merupakan diagram **model konseptual** untuk menggambarkan struktur *logis* dari basisdata berbasis grafis
3. Komponen-komponen utama model ER adalah:
  - ✓ **Entitas (entity)**, Entitas memodelkan objek-objek yang berada di perusahaan/lingkungan.
  - ✓ **Relationship**. Relationship memodelkan koneksi/hubungan di antara entitas-entitas.
  - ✓ **Atribut-atribut (properi-properiti)**, memodelkan properti-properiti dari entitas dan relationship.
  - ✓ **Konstrain-konstrain (batasan-batasan)** integritas, konstrain-konstrain ketentuan validitas.
4. **Entitas** diartikan sebagai '**objek**' di dunia nyata yang bisa dibedakan dengan '**objek**' yang lain. Notasi entitas digambarkan dengan Persegi panjang.
5. **Relasi** diartikan sebagai **hubungan** yang terjadi diantara **satu entitas dengan entitas yang lainnya**. Notasi relasi di gambarkan dengan belah ketupat.
6. Setiap entitas mempunyai atribut yang berisi **karakteristik** yang mendeskripsikan dari entitas tersebut. Notasi atribut di gambarkan dengan elips.
7. Derajat himpunan relasi ada 3 macam, yaitu *unary*, *binary* dan *ternary*.
8. Kardinalitas relasi menggambarkan **banyaknya jumlah maksimum entitas dapat berelasi dengan entitas pada himpunan entitas yang lain**.
9. Pada himpunan **relasi biner**, pemetaan kardinalitas relasi dapat berupa salah satu dari berikut ini
  - a. Satu ke Satu
  - b. Satu ke Banyak atau Banyak ke Satu
  - c. Banyak ke Banyak

10. *Key* adalah satu atau gabungan dari beberapa atribut yang dapat membedakan semua *row* dalam relasi secara unik. Macam *key* dibedakan jadi 3 = *super key*, *candidate key*, *primary key*.

### 3 KONVERSI ER KE BASIS DATA REALTIONAL



#### Overview

---

---

Pemodelan ER adalah pemodelan konseptual sebuah basis data relasional. Desain ER bisa di konversikan kedalam bentuk tabel fisik yang akan di simpan di dalam basis data. Didalam bab ini akan di bahas lebih lanjut dari ER yaitu tentang *weak entity*, spesialisasi, agregasi dan penurunan Diagram ER ke bentuk tabel. Penekanan pada bab ini adalah kapan dan bagaimana kita menggunakan *weak entity*, spesilisasi dan agregasi dan bagaimana menurunkan konseptual ER kedalam tabel.



#### Tujuan

---

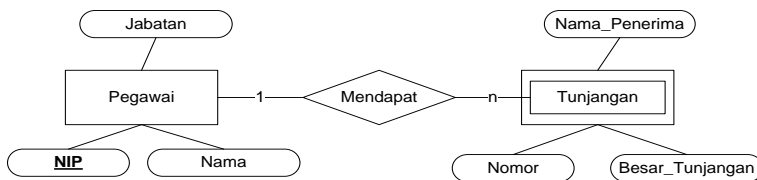
---

1. Mahasiswa memahami konsep *Weak Entity*, Spesialisasi dan Agregasi
2. Mahasiswa memahami konsep penurunan dari Diagram ER ke Tabel

### 3.1 Himpunan Entitas Lemah

Secara umum, Himpunan Entitas Lemah tidak memiliki *primary key* dan selalu bergantung pada entitas lain. **Notasi entitas lemah** digambarkan dengan **double persegi panjang**, sedangkan **relasi** untuk himpunan entitas lemah digambarkan dengan **double diamond**. **Diskriminator / key parsial** adalah atribut – atribut yg dpt membedakan entitas – entitas yang terdapat di himpunan entitas lemah. Diskriminator tidak sama dengan *primary key*. Konsep diskriminator hanya di pakai pada himpunan entitas lemah. **Primary key pada Himpunan Entitas lemah** ada 2 yaitu **primary key dari entitas kuat** yg berelasi **dan diskriminator / key parsialnya**.

**Diskriminator** di notasikan dengan garis bawah yang putus putus.



**Gambar 0-1 Contoh Himpunan Entitas Lemah**

Relasi di atas menggambarkan bahwa seorang pegawai mendapatkan fasilitas tunjangan dari perusahaan tempat dia bekerja. Tunjangan dalam hal ini adalah entitas lemah. Tunjangan sebagai entitas tidak bisa berdiri sendiri, tunjangan harus bergantung pada entitas pegawai (\* tidak akan ada tunjangan jika tidak ada pegawai).

Kardinalitas relasi yang terjadi pada himpunan entitas lemah biasanya merupakan banyak ke 1 atau 1 ke banyak dengan kardinalitas 1 di himpunan entitas yang lebih kuat.

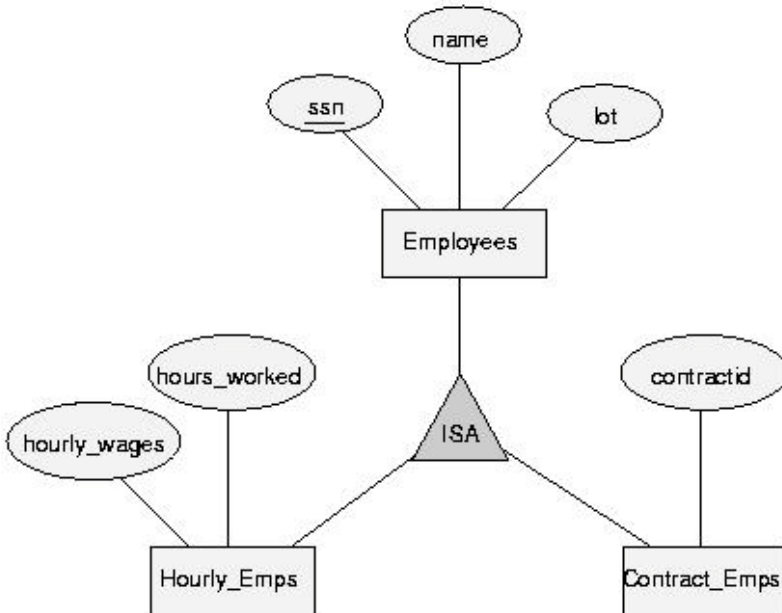
### 3.2 Spesialisasi dan Generalisasi

**Spesialisasi** merupakan proses desain *top-down* dengan mendesain *subgrouping* didalam didalam himpunan entitas yang berbeda dari himpunan entitas. Tujuan dari spesialisasi adalah memberikan gambaran konseptual tentang perbedaan karakteristik dari himpunan entitas yang hampir serupa dengan konsep *sub grouping* / pengelompokan.

**Subgrouping** di atas menjadi himpunan entitas yang levelnya lebih rendah dan memiliki atribut tersendiri yang tidak dimiliki pada level di atasnya. Atribut

ini khas dan merupakan pembeda dari entitas di *subgroup* yang lain. **IS A** dinotasikan dengan gambar segitiga berlabel IS A.

Sifat dari **spesialisasi** adalah **inheritan atribut** yaitu atribut pada level tinggi secara otomatis akan di turunkan pada level di bawahnya.



**Gambar 0-2 Contoh Spesialisasi dan Generalisasi**

Contoh di atas menggambarkan bahwa entitas **employees** mempunyai 2 *subgroup* yaitu **Hourly\_Emps** dan **Contract\_Emps**. Kedua entitas pegawai tetap dan pegawai honorer sama-sama mempunyai atribut turunan yaitu nama dan ssn, name, lot dari **employees**. Perbedaan dari **Hourly\_Emps** dan **Contract\_Emps** terdapat di atribut yang melekat pada *subgroup*-nya. Atribut hourly\_wages dan hours\_worked hanya terdapat di himpunan entitas **Hourly\_Emps**, sedangkan atribut contactid kerja terdapat di himpunan entitas **Contract\_Emps**.

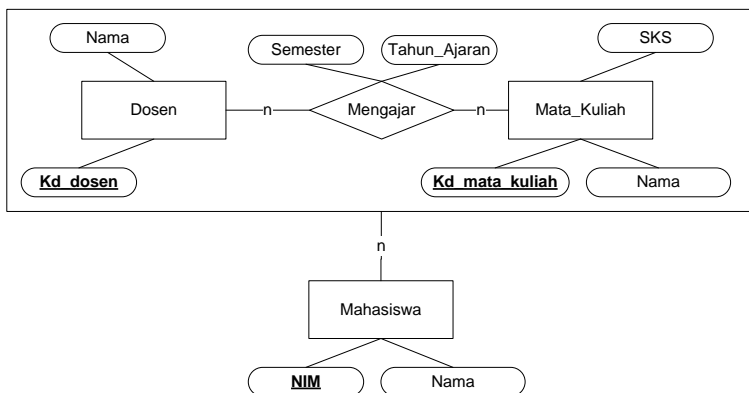
**Generalisasi** merupakan proses desain **bottom-up** dengan mengkombinasikan jumlah himpunan entitas yang digunakan secara bersama-sama. Spesialisasi dan generalisasi sama-sama digambarkan dengan notasi IS



A, yang membedakan adalah sudut pandangnya saja. Jika Spesialisasi kita mendefinisikan entitas secara umum kemudian mencari *subgroup* dari entitas tersebut, tetapi generalisasi memandang sebaliknya, dari adanya *subgroup subgroup* yang berbeda kemudian di cari entitas umum yang mewakili 2 himpunan entitas tersebut.

### 3.3 Agregasi

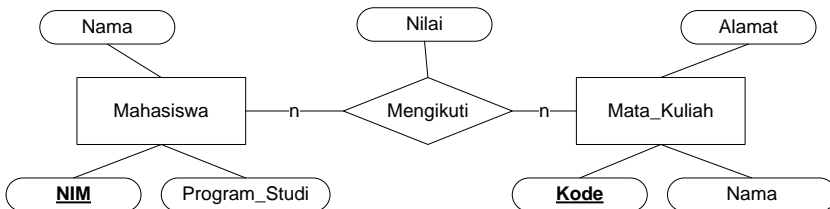
**Agregasi** adalah enkapsulasi dari entitas entitas yang berelasi (\*n-n). Pada umumnya terbentuk dari kardinalitas relasi banyak ke banyak. Didalam konsep agregasi terdapat istilah enkapsulasi relasi dari kedua entitas. Enkapsulasi di perlukan karena kedua himpunan entitas yang ber-relasi tersebut merupakan 1 kesatuan yang tidak bisa di pisah. Notasi agregasi di gambarkan dengan gambar persegi panjang yang membungkus himpunan entitas yang saling ber-relasi.



**Gambar 0-3 Contoh Agregasi**

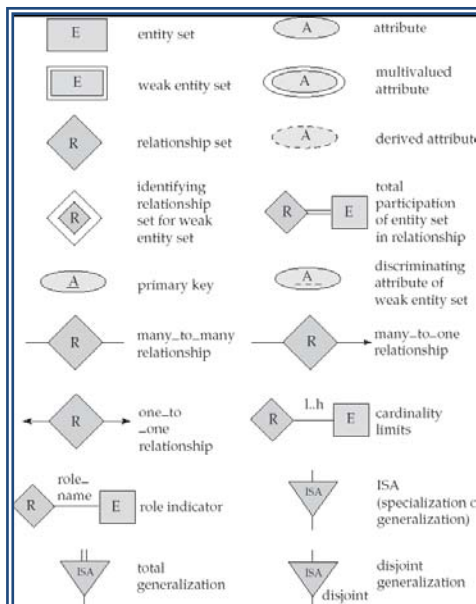
Gambar di atas menunjukkan relasi dosen mengajar sebuah mata kuliah dan mahasiswa mengambil mata kuliah yang diajarkan oleh dosen tertentu. Agregasi di perlukan dikarenakan tidak di mungkinkan mahasiswa untuk mengambil mata kuliah tanpa adanya dosen yang bersedia untuk mengajar mata kuliah tersebut. Dalam kasus di atas menekankan bahwa himpunan entitas dosen harus ber-relasi terlebih dahulu dengan himpunan entitas mata kuliah, kemudian relasinya di pandang sebagai 1 entitas yang ber-relasi dengan himpunan entitas mahasiswa lewat relasi mengambil. *Primary key* dari

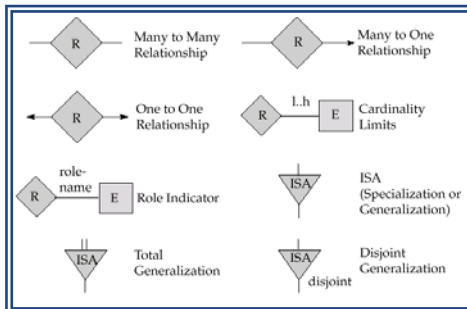
kedua himpunan entitas dosen dan mata kuliah akan secara implisit masuk ke relasi mengajar dengan di tambah 2 atribut deskriptif (\* semester dan thn\_ajaran). Relasi tersebut di anggap sebagai 1 entitas seperti gambar di bawah ini.



**Gambar 0-4 Relasi di pandang sebagai Himpunan Entitas**

### 3.4 Ringkasan notasi simbol di ER





**Gambar 0-5 Ringkasan Notasi pada Diagram ER**

### 3.5 Skema ER ke Tabel

**Penurunan skema** dimaksudkan untuk mengubah sebuah konsep hubungan entitas dan relasi kedalam bentuk fisik tabel tabel yang berelasi. Inti dari *Entity Relationship* adalah menggambarkan hubungan di dunia nyata kedalam bentuk entitas entitas yang saling ber-relasi, dari diagram ini bisa di buat kedalam bentuk tabel yang langsung di implementasikan kedalam basis data.

Secara umum penurunan diagram ER ke tabel memiliki aturan sebagai berikut :

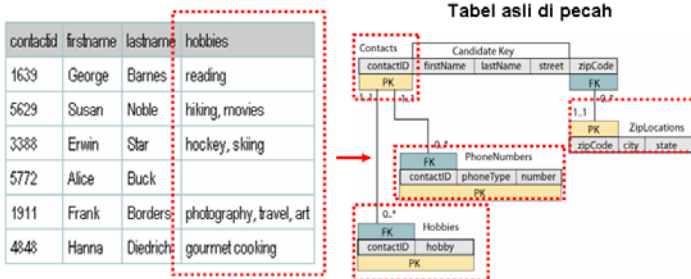
- Setiap **himpunan entitas** menjadi **Tabel** (\* baik himpunan entitas kuat atau lemah)
- Setiap **atribut** menjadi **kolom** di tabel
- **Kardinalitas relasi** akan menentukan jumlah tabel yang terbentuk (\* akan di bahas di bawah lebih detail)

### 3.6 Representasi Atribut menjadi Kolom

Pada atribut bertipe **simple** , **single** dan **derived** direpresentasikan **sama persis** seperti diagram ER. Tetapi untuk atribut **komposit** dan **multivalued** mempunyai aturan tersendiri.

**Atribut komposit** akan dipecah dengan membuat atribut terpisah untuk masing masing komponennya. Contoh atribut **nama** pada tabel mahasiswa, di pecah menjadi 2 kolom yaitu **nama depan** dan **nama belakang**.

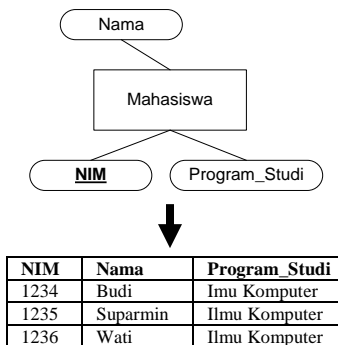
**Atribut multivalued** mengharuskan untuk di pecah menjadi 2 Tabel. Atribut **multivalued** M dari entitas E direpresentasikan oleh tabel terpisah EM. Perhatikan gambar di bawah yang menunjukkan bagaimana penurunan sebuah atribut **multivalued** :



**Gambar 0-6 Atribut *multivalued* di pecah menjadi entitas baru**

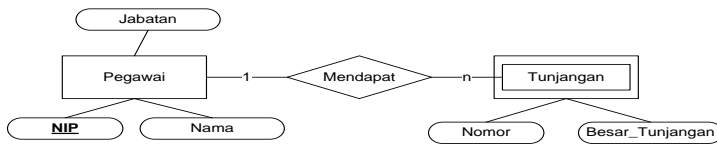
### 3.7 Representasi Himpunan Entitas sebagai Tabel

**Himpunan entitas kuat** di representasikan kedalam tabel dengan kolom sama persis dengan atribut yang sudah di definisikan di diagram ER. Perhatikan gambar di bawah ini :



**Gambar 0-7 Atribut himpunan entitas kuat di representasikan kedalam tabel**

**Himpunan entitas lemah** akan menjadi **tabel tersendiri** yang didalamnya ada kolom *primary key* yang merupakan identifikasi dari himpunan entitas kuat. Contoh di bawah menggambarkan himpunan entitas lemah di turunkan kedalam tabel.

**Tabel Mahasiswa**

NIP	Nama	Jabatan
1234	Budi	Programmer
1235	Suparmin	Analist
1236	Wati	Dokumentator

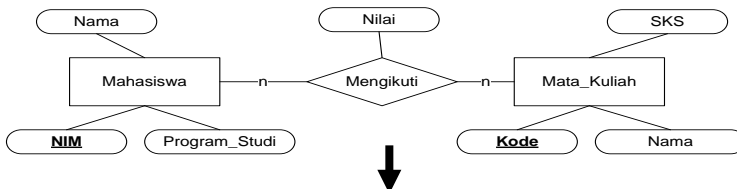
**Tabel Mengikuti**

NIP	Nomor	Besar_Tunjangan
1234	T01	1.000.000
1235	T02	1.500.000
1236	T03	1.500.000

**Gambar 0-8 Penurunan Himpunan Entitas Lemah ke tabel**

### 3.7.1 Representasi Relasi (\* pada kardinalitas N to N)

Relasi dari Himpunan Banyak ke Banyak direpresentasikan kedalam Tabel tersendiri dengan *primary key* dari 2 Entitas menjadi atribut di Tabel Relasi. Perhatikan relasi banyak ke banyak berikut dan contoh penurunan ke tabel :

**Tabel Mahasiswa**

NIM	Nama	Program_Studi
1234	Budi	Imu Komputer
1235	Suparmin	Ilmu Komputer
1236	Wati	Ilmu Komputer

**Tabel Mengikuti**

NIM	Kode	Program_Studi
1234	IK 330	Basis Data
1235	IK 330	Basis Data
1236	IK 331	Sistem Basis Data

**Tabel Mata\_Kuliah**

Kode	Nama	SKS
IK 330	Basis Data	2
IK 331	Sistem Basis Data	3

**Gambar 0-9 Penurunan Kardinalitas relasi N to N menjadi Tabel**

### Hubungan kardinalitas dengan tabel yang terbentuk

Kardinalitas relasi dari Himpunan Entitas yang saling ber-relasi akan menentukan banyaknya tabel yang bisa di buat. Adapun aturannya sebagai berikut :

- 1 ke 1  
Pilih *primary key* di 1 himpunan entitas untuk menjadi *foreign key* bagi himpunan entitas yang lain.
- 1 ke banyak / banyak ke 1  
*Primary key* pada Tabel berkardinalitas sedikit menjadi *foreign key* pada tabel berkardinalitas banyak.
- Banyak ke banyak  
Digambarkan pada bab 3.7.1

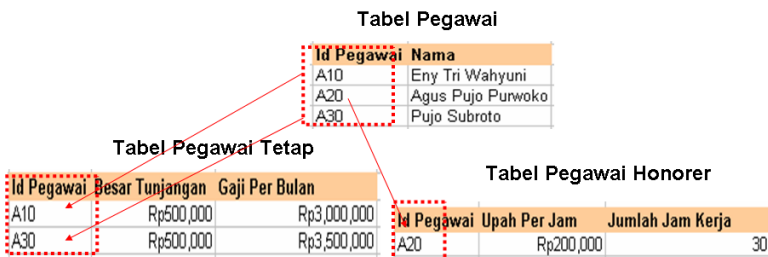
### 3.7.2 Representasi Spesialisasi (IS A)

Ada 2 pendekatan yang dipakai didalam menurunkan spesialisasi kedalam tabel.

#### Pendekatan 1

- Bentuklah tabel untuk level entitas yg lebih tinggi
- Bentuklah tabel untuk level entitas yg lebih rendah

(\*dengan memasukkan *primary key* pada level yg lebih tinggi ke tabel dengan level yang lebih rendah)



**Gambar 0-10 Representasi spesialisasi ke tabel metoda 1**

#### Pendekatan 2

- Bentuklah tabel untuk tiap himpunan entitas dengan semua atribut lokal dan turunan.
- Bisa jadi tabel pada level tinggi tidak perlu di simpan jika spesialisasi adalah total. Jika diperlukan bisa dibuat **view** yang menggabungkan tabel-tabel spesialisasi.

**Tabel Pegawai Tetap**

<b>Id Pegawai</b>	<b>Nama</b>	<b>Besar Tunjangan</b>	<b>Gaji Per Bulan</b>
A10	Eny Tri Wahyuni	Rp500,000	Rp3,000,000
A30	Pujo Subroto	Rp500,000	Rp3,500,000

**Tabel Pegawai Honorer**

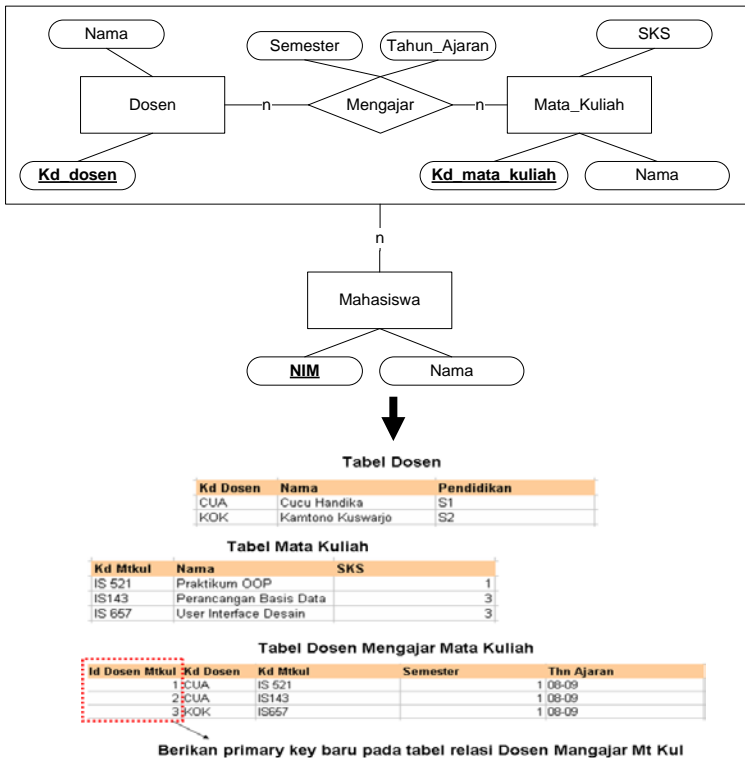
<b>Id Pegawai</b>	<b>Nama</b>	<b>Upah Per Jam</b>	<b>Jumlah Jam Kerja</b>
A20	Agus Pujo Purwo	Rp200,000	30

**Gambar 0-11 Representasi spesialisasi ke tabel metoda 1**

### 3.7.3 Representasi Agregasi

Untuk merepresentasikan agregasi, buatlah tabel yang terdiri dari :

- **Foreign key** dari **himpunan entitas yang berhubungan**
- Setiap atribut deskriptif
- Atribut baru untuk *primary key* di tabel relasi



**Gambar 0-12 Representasi Agregasi untuk tabel mata kuliah, dosen dan Dosen mengajar mt kul**



Tabel Mahasiswa

NIM	Nama	Prodi	Alamat	Kota	Ipk	Umur
30107001	Bambang Sumantri	Manajemen Informatika	Jl Telekomunikasi 1 No3	Bandung	3	20
30207001	Susi Lo	Teknik Komputer	Jl Sukapura 43	Sukarta	3.5	20
30307001	Sumarno	Komputerisasi Akuntansi	Jl Bojongsoang 34	Bandung	2.75	20
30307002	Frans Sahilatua	Komputerisasi Akuntansi	Griya Bandung Asri 1 D277	Bandung	2.5	19

Tabel Mahasiswa Mengambil Mtkul

NIM	Id Dosen Mtkul	Semester	Thn Ajaran	Nilai
30107001	1		1 08-09	A
30307001	1		1 08-09	B
30107001	2		1 08-09	A

**Gambar 0-13 Representasi Agregasi untuk tabel Mahasiswa dan Mahasiswa Mengambil Mtkul**



## Rangkuman

1. Himpunan Entitas Lemah tidak memiliki *primary key* dan selalu bergantung pada entitas lain.
2. Notasi entitas lemah adalah doble persegi panjang, sedangkan relasinya *double diamond*.
3. Jika konsep *primary key* di pakai didalam himpunan entitas kuat, maka **diskriminator** dipakai sebagai pembeda antar entitas di dalam himpunan entitas lemah. Diskriminator di gambarkan sebagai garis bawah yang terputus putus.
4. **Spesialisasi** merupakan proses desain *top-down*; dengan mendesain *subgrouping* didalam himpunan entitas yang berbeda dari himpunan entitas.
5. **Generalisasi** merupakan proses desain bottom-up; mengkombinasikan jumlah himpunan entitas yang digunakan secara bersama-sama.
6. **Agregasi** adalah enkapsulasi dari entitas entitas yang berelasi (\*n-n). Notasi agregasi adalah Persegi Panjang yang membungkus himpunan entitas biner yang saling ber-relasi.
7. Secara umum penurunan diagram ER ke tabel memiliki aturan sebagai berikut :
  - Setiap himpunan entitas menjadi Tabel  
(\* baik himpunan entitas kuat atau lemah)
  - Setiap atribut menjadi kolom di tabel
  - Kardinalitas relasi akan menentukan jumlah tabel yang terbentuk  
(\* akan di bahas di bawah lebih detail)
8. **Atribut komposit** akan dipecah dengan membuat atribut terpisah untuk masing masing komponennya.
9. **Atribut multivalued** mengharuskan untuk di pecah menjadi 2 Tabel.
10. **Himpunan Weak Entity** akan menjadi tabel tersendiri yang didalamnya ada kolom *primary key* yang merupakan identifikasi dari *strong entity*.
11. Kardinalitas relasi menentukan berapa banyak tabel yang terbentuk.

## 4 NORMALISASI



### Overview

---

Bab ini akan membahas konsep normalisasi *database* berikut konsep-konsep lain yang mendasarinya. Dalam bab ini juga akan ditampilkan contoh penerapan normalisasi untuk tabel-tabel sederhana dalam kasus *database* akademik di sebuah perguruan tinggi.



### Tujuan

---

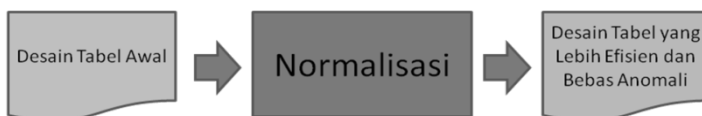
1. Mahasiswa memahami definisi dan tujuan normalisasi
2. Mahasiswa mampu mengidentifikasi *super key*, *candidate key* dan *primary key* dari sebuah *table*
3. Mahasiswa mampu mengidentifikasi *functional dependency* yang ada pada sebuah tabel, termasuk partial dan transitive FD
4. Mahasiswa mengenal bentuk normal pertama, ke dua, ke tiga dan BCNF serta mampu melakukan normalisasi dengan menerapkan bentuk-bentuk normal tersebut
5. Mahasiswa mengenal sekilas tentang bentuk-bentuk normal lain dan memahami konsep denormalisasi

## 4.1 Definisi Normalisasi

Normalisasi adalah langkah-langkah sistematis untuk menjamin bahwa struktur *database* memungkinkan untuk *general purpose query* dan bebas dari *insertion*, *update* dan *deletion anomalies* yang dapat menyebabkan hilangnya integritas data (E.F. Codd, 1970)

## 4.2 Tujuan Normalisasi

Pada dasarnya normalisasi dilakukan untuk memperbaiki desain tabel yang kurang baik sehingga penyimpanan data menjadi lebih efisien dan bebas anomali data. Untuk memperjelas pemahaman tentang proses normalisasi, perhatikan diagram berikut:



**Gambar 4-1 Diagram Normalisasi**

Intinya, normalisasi dilakukan terhadap desain tabel yang sudah ada dengan tujuan untuk meminimalkan redundansi (pengulangan) data dan menjamin integritas data dengan cara menghindari 3 Anomali Data: *Update*, *Insertion* dan *Deletion Anomaly*.

### 4.2.1 Update Anomaly

**Tabel 4-1 Contoh Update Anomaly**

NIM	Nama_Mhs	Kd_Jur	Nama_Jur	Kode_MK	Nama_MK	SKS	Nilai
1-01	Tukimin	TE	Elektro	TE-001	Elektronika	3	A
1-01	Tukimin	TE	Elektro	DU-001	English	2	A
2-01	Jamilah	IF	Informatika	IF-001	Algoritma	3	B
2-01	Jamilah	IF	Informatika	DU-001	English	2	C
2-02	Maemunah	IF	Informatika	IF-002	Database	2	A

Tabel di atas adalah contoh tabel yang memiliki desain yang kurang baik. Perhatikan bahwa jika kita ingin meng-*update* jumlah sks mata kuliah English dari 2 menjadi 3 sks, maka kita harus mengupdate lebih dari 1 *record*, yaitu baris 2 dan 4.

Jika hanya salah satu baris saja yang di-*update*, maka data menjadi tidak konsisten (ada mata kuliah English dengan 2 sks dan ada mata kuliah English dengan 3 sks) . Kondisi seperti inilah yang disebut dengan *update anomaly*.

### 4.2.2 Insertion Anomaly

**Tabel 4-2 Contoh Insert Anomaly**

NIM	Nama Mhs	Kd_Jur	Nama_Jur	Kode_MK	Nama_MK	SKS	Nilai
1-01	Tukimin	TE	Elektro	TE-001	Elektronika	3	A
1-01	Tukimin	TE	Elektro	DU-001	English	2	A
2-01	Jamilah	IF	Informatika	IF-001	Algoritma	3	B
2-01	Jamilah	IF	Informatika	DU-001	English	2	C
2-02	Maemunah	IF	Informatika	IF-002	Database	2	A

Pada tabel yang sama seperti contoh di atas, terjadi pula *insertion anomaly*. Misalkan terdapat mahasiswa baru dengan nim 1-02 bernama ‘Zubaedah’ dengan kode jurusan ‘TE’ dan nama jurusan ‘Elektro’.

Data mahasiswa tersebut tidak dapat dimasukkan ke dalam tabel sebab dia belum mengambil kuliah apapun (misalnya karena belum melakukan registrasi). Kondisi inilah yang disebut dengan *insertion anomaly*.

### 4.2.3 Deletion Anomaly

**Tabel 4-3 Contoh Delete Anomaly**

NIM	Nama Mhs	Kd_Jur	Nama_Jur	Kode_MK	Nama_MK	SKS	Nilai
1-01	Tukimin	TE	Elektro	TE-001	Elektronika	3	A
1-01	Tukimin	TE	Elektro	DU-001	English	2	A
2-01	Jamilah	IF	Informatika	IF-001	Algoritma	3	B
2-01	Jamilah	IF	Informatika	DU-001	English	2	C
2-02	Maemunah	IF	Informatika	IF-002	Database	2	A

Pada contoh tabel di atas terjadi *deletion anomaly*. Perhatikan bahwa jika kita menghapus data mahasiswa bernama ‘Maemunah’ maka kita harus menghapus data pada baris ke 5, hal ini akan mengakibatkan kita juga kehilangan data mata kuliah ‘Database’. Kondisi inilah yang disebut dengan *deletion anomaly*.

Selain 3 anomali di atas, ada beberapa konsep yang mendasari normalisasi. Adapun konsep-konsep penting yang mendasari normalisasi adalah konsep mengenai *super key*, *candidate key*, *primary key*, *functional dependency* dan tentu saja bentuk-bentuk normal yang menjadi acuan kita dalam melakukan normalisasi terhadap desain sebuah tabel. Pemahaman terhadap konsep-konsep ini sangat penting dan akan dibahas di beberapa sub bab berikutnya.

### 4.3 Konsep Keys

Konsep tentang *key* adalah konsep yang penting untuk memahami keterkaitan antar atribut data dalam tabel dan akan sangat berguna dalam proses normalisasi. Dalam setiap tabel, terdapat 3 macam *key*:

a) **Super key**

*Super key* adalah satu atribut atau gabungan atribut (kolom) pada tabel yang dapat membedakan semua baris secara unik.

b) **Candidate key**

*Candidate key* disebut juga dengan *minimal super key*, yaitu *super key* yang tidak mengandung *super key* yang lain. Setiap *candidate key* pasti merupakan *super key*, namun tidak semua *super key* akan menjadi *candidate key*.

c) **Primary key**

*Primary key* adalah salah satu *candidate key* yang dipilih (dengan berbagai pertimbangan) untuk digunakan dalam DBMS. Tiap tabel hanya memiliki 1 *primary key*, namun *primary key* tersebut bisa saja dibentuk dari beberapa atribut (kolom).

Untuk memperjelas pemahaman kita terhadap 3 macam *key* di atas, perhatikan contohnya pada tabel mata\_kuliah di bawah ini:

**Tabel 4-4 Tabel Mata Kuliah**

Kode_MK	Nama_MK	Semester	SKS
DU-001	English	2	2
DU-002	Kalkulus	1	3
IF-001	Algoritma	1	3
IF-002	Database	2	3
IF-003	Artificial Intelligence	5	2
TE-001	Elektronika	4	3

Beberapa *super key* dari tabel di atas adalah:

1. (*kode\_mk*)

Dari 6 baris data yang ada pada tabel di atas tak ada satupun yang memiliki *kode\_mk* yang sama.

2. (*nama\_mk*)

Demikian pula dengan *nama\_mk*, masing-masing baris data memiliki *nama\_mk* yang unik. Tidak ada satupun baris data yang memiliki kolom *nama\_mk* dengan nilai yang sama persis.

3. (*kode\_mk, nama\_mk, semester*)

Walaupun beberapa baris data memiliki kolom semester dengan nilai yang sama (misalnya baris 1&4, baris 2&3) namun tidak ada satupun baris data yang memiliki kombinasi kode\_mk, nama\_mk dan semester yang sama persis.

4. (kode\_mk, nama\_mk, sks)

Kombinasi kode\_mk, nama\_mk dan sks juga digolongkan sebagai *super key* dengan alasan yang kurang lebih sama dengan poin 3.

5. (kode\_mk, nama\_mk, semester, jml\_temu)

Kombinasi kode\_mk, nama\_mk, semester dan jml\_temu juga digolongkan sebagai *super key* dengan alasan yang kurang lebih sama dengan poin 3 dan 4.

Sedangkan yang **bukan super key** adalah:

1. (sks)

Perhatikan bahwa kolom sks tidak bisa membedakan baris data secara unik, contohnya baris data 2,3, 4 dan 6 sama-sama memiliki kolom sks bernilai 3.

2. (semester)

Kolom semester juga tidak bersifat unik, contohnya baris data 1 dan 4 sama-sama memiliki kolom semester bernilai 2

3. (semester, sks)

Kombinasi semester dan sks juga tidak membedakan tiap baris data secara unik, contohnya baris data ke 2 dan 3 sama-sama memiliki kolom semester bernilai 1 dan sama-sama memiliki kolom sks bernilai 3

*Candidate key* dari tabel mata\_kuliah dipilih dari *super key* yang sudah ada. *Super key* yang akan menjadi *candidate key* adalah *super key* yang tidak mengandung *super key* lain di dalamnya.

Perhatikan 5 *super key* yang sudah kita peroleh dari analisis sebelumnya:

1. (kode\_mk)

2. (nama\_mk)

3. (kode\_mk, nama\_mk, semester)

4. (kode\_mk, nama\_mk, sks)

5. (kode\_mk, nama\_mk, semester, jml\_temu)

*Super key* yang hanya terdiri dari satu atribut data pasti akan menjadi *candidate key* sebab tidak mungkin mengandung *super key* yang lain. Oleh karena itu *super key* pada poin 1 dan 2 otomatis menjadi *candidate key*. *Super key* pada poin 3 tidak menjadi *candidate key* sebab dalam kombinasi (kode\_mk, nama\_mk, semester) terdapat *super key* yang lain yaitu (kode\_mk). Dengan demikian, poin 4 dan 5 juga bukan *candidate key*.

Dari analisis ini, kita memperoleh 2 buah *candidate key* yaitu (kode\_mk) dan (nama\_mk). Salah satu dari beberapa *candidate key* ini akan dipilih untuk digunakan dalam DBMS sebagai *primary key*. Ada beberapa pertimbangan untuk memilih *primary key*, di antaranya adalah jaminan keunikan yang lebih kuat, representasi yang lebih baik dan lain-lain.

#### 4.4 Functional Dependencies

*Functional dependency* (FD) atau kebergantungan fungsional adalah *constraint* atau batasan/ ketentuan antara 2 buah himpunan atribut pada sebuah tabel.

Jika A dan B adalah himpunan atribut dari tabel T, kebergantungan fungsional antara A dan B biasanya dinyatakan dalam notasi  $A \rightarrow B$ . Notasi  $A \rightarrow B$  berarti:

- A menentukan B
- B secara fungsional bergantung kepada A.

$A \rightarrow B$  jika memenuhi syarat berikut ini :

Pada sebuah tabel T, jika ada dua baris data atau lebih dengan nilai atribut A yang sama maka baris-baris data tersebut pasti akan memiliki nilai atribut B yang sama Namun hal ini tidak berlaku sebaliknya.

Untuk lebih jelasnya perhatikan tabel berikut ini:

**Tabel 4-5 Contoh Tabel**

NIM	Nama_Mhs	Kd_Jur	Nama_Jur	Kode_MK	Nama_MK	SKS	Nilai
1-01	Tukimin	TE	Elektro	TE-001	Elektronika	3	A
1-01	Tukimin	TE	Elektro	DU-001	English	2	A
2-01	Jamilah	IF	Informatika	IF-001	Algoritma	3	B
2-01	Jamilah	IF	Informatika	DU-001	English	2	C
2-02	Maemunah	IF	Informatika	IF-002	Database	2	A

Contoh kebergantungan fungsional yang terdapat pada tabel di atas:

- $NIM \rightarrow Nama\_mhs$



Untuk setiap baris data, jika  $NIM = 1-01$  pasti  $Nama\_mhs = 'Tukimin'$ , walaupun belum tentu semua mahasiswa yang bernama Tukimin memiliki  $NIM = 1-01$

- $NIM \rightarrow Kd\_jur$

Untuk setiap baris data, jika  $NIM = 1-01$  pasti  $Kd\_jur = 'TE'$ , walaupun tidak semua baris data dengan  $kd\_jur = 'TE'$  memiliki kolom  $NIM$  bernilai 1-01

- $NIM \rightarrow Nama\_Jur$

Untuk setiap baris data dengan kolom  $NIM$  bernilai 1-01 pasti memiliki kolom  $Nama\_Jur = 'Elektro'$ , walaupun tidak semua orang di jurusan Elektro memiliki  $NIM = 1-01$ . Demikian pula tidak semua baris data pada tabel dengan kolom  $Nama\_Jur = 'Elektro'$  memiliki kolom  $NIM = 1-01$

Penulisan kebergantungan fungsional dari 3 poin di atas dapat diringkas menjadi  $(NIM) \rightarrow (nama\_mhs, kd\_jur, nama\_jur)$

Dengan demikian, dari tabel tersebut dapat kita simpulkan beberapa kebergantungan fungsional (FD) sebagai berikut:

- $FD1: (nim) \rightarrow (nama\_mhs, kd\_jur, nama\_jur)$
- $FD2: (kd\_jur) \rightarrow (nama\_jur)$
- $FD3: (kode\_mk) \rightarrow (nama\_mk, sks)$
- $FD4: (nim, kode\_mk) \rightarrow (nilai)$

Ada beberapa jenis kebergantungan fungsional, di antaranya yaitu:

- a. *Partial Functional dependency*
- b. *Transitive Functional dependency*
- c. *Multivalued Functional dependency*

Ketiganya adalah konsep penting dalam normalisasi, namun dalam sub bab ini kita hanya akan membahas mengenai *Partial Functional dependency* dan *Transitive Functional dependency*.

#### 4.4.1 Partial Functional Dependency

*Partial Functional dependency* atau kebergantungan fungsional parsial terjadi bila:

- $B \rightarrow A$
- B adalah bagian dari *candidate key*

Dengan kata lain jika  $(B,C)$  adalah *candidate key* dan  $B \rightarrow A$  maka A bergantung secara parsial terhadap  $(B,C)$  atau  $(B,C)$  menentukan A secara parsial.

Untuk lebih jelasnya perhatikan tabel berikut ini:

**Tabel 4-6 Tabel Nilai**

NIM	Nama_Mhs	Kode_MK	Nilai
1-01	Tukimin	TE-001	A
1-01	Tukimin	DU-001	A
2-01	Jamilah	IF-001	B
2-01	Jamilah	DU-001	C
2-02	Maemunah	IF-002	A

Pada tabel di atas perhatikan bahwa:

1. *Super key* : (nim,kode\_mk), (nim,nama\_mhs,kode\_mk) dan (nim,nama\_mhs,kode\_mk,nilai)
2. Dari *super key* yang sudah diperoleh pada poin 1, maka dipilih *super key* yang akan menjadi *candidate key* yaitu (nim,kode\_mk)
3. FD: (nim)  $\rightarrow$  (nama\_mhs)

Dari analisis poin 2 dan 3 maka dapat disimpulkan bahwa terjadi kebergantungan fungsional parsial dimana (nama\_mhs) bergantung kepada (nim,kode\_mk) secara parsial atau dapat juga dikatakan bahwa (nim,kode\_mk) menentukan (nama\_mhs) secara parsial.

#### 4.4.2 Transitive Functional dependency

*Transitive Functional dependency* atau kebergantungan fungsional transitif terjadi jika:

- $A \rightarrow B$
- $B \rightarrow C$

Jika  $A \rightarrow B$  dan  $B \rightarrow C$  maka  $A \rightarrow C$ . Dengan kata lain A bergantung secara transitif terhadap C melalui B atau A menentukan C secara transitif melalui B. Untuk lebih jelasnya perhatikan contoh tabel berikut ini:

**Tabel 4-7 Tabel Mahasiswa**

NIM	Nama_Mhs	Kd_Jur	Nama_Jur
1-01	Tukimin	TE	Elektro
1-01	Tukimin	TE	Elektro
2-01	Jamilah	IF	Informatika
2-01	Jamilah	IF	Informatika
2-02	Maemunah	IF	Informatika

FD1: (nim)  $\rightarrow$  (nama\_mhs, kd\_jur, nama\_jur)

FD2: (kd\_jur)  $\rightarrow$  (nama\_jur)

Dengan demikian dapat disimpulkan bahwa (nama\_jur) bergantung secara transitif terhadap (nim) melalui (kd\_jur) atau dapat juga dikatakan bahwa (nim)  $\rightarrow$  (nama\_jur) secara transitif melalui (kd\_jur).

## 4.5 Bentuk Normal dan Langkah-Langkah Normalisasi

Bentuk Normal adalah sekumpulan kriteria yang harus dipenuhi oleh sebuah desain tabel untuk mencapai tingkat/level bentuk normal tertentu. Parameter yang biasanya digunakan dalam menentukan kriteria bentuk normal adalah *Functional dependency & The Three Keys*.

Masing-masing bentuk normal memiliki kriteria dan level tertentu yang tidak mungkin dicapai tanpa memenuhi kriteria bentuk normal level yang berada di bawahnya. Makin tinggi level bentuk normal yang dicapai maka kualitas desain tabel tersebut dinyatakan makin baik dan semakin kecil peluang terjadinya anomali dan redundansi data.

Normalisasi dilakukan dengan cara menerapkan Bentuk-Bentuk Normal secara bertahap dari level terendah sampai level yang dikehendaki. Walaupun ada beberapa bentuk normal namun jika desain tabel tertentu sudah memenuhi kriteria 3<sup>rd</sup> NF atau BCNF maka desain tabel itu biasanya dianggap sudah ‘cukup normal’.

### 4.5.1 Bentuk Normal Pertama (1st Normal Form)

Bentuk normal pertama atau *First Normal Form* (1<sup>st</sup> NF) adalah bentuk normal yang memiliki level terendah.

Kriteria 1<sup>st</sup> NF:

- **Tidak ada atribut (kolom) pada tabel yang bersifat multi-value**  
Sebuah atribut disebut bersifat *multivalued* jika dalam sebuah baris data pada kolom tersebut terdapat lebih dari satu nilai. Misalnya kolom telepon yang berisi ‘0813xx, 022xxx’ dan seterusnya.
- **Tidak memiliki lebih dari satu atribut dengan domain yang sama**  
Sebuah tabel dikatakan memiliki lebih dari satu atribut dengan domain yang sama jika pada tabel tersebut terdapat lebih dari satu kolom yang digunakan untuk menyimpan data yang jenisnya sama. Misalnya kolom *telepon1*, *telepon2*, *telepon3* dan seterusnya.

Untuk lebih jelasnya perhatikan 2 versi contoh tabel T berikut ini:

**Table 4-8 Versi pertama**

NIM	Nama_Mhs	Telp_1	Telp_2	Kd_Jur	Nama_Jur	Kode_MK	Nama_MK	SKS	Nilai
1-01	Tukimin	0813xx	022xxx	TE	Elektro	TE-001	Elektronika	3	A
2-01	Jamilah	0812xx	021xxx	IF	Informatika	IF-001	Algoritma	3	B
2-02	Maemunah	0852xx	031xxx	IF	Informatika	IF-002	Database	2	A

Tabel T versi pertama ini memiliki 2 atribut dengan domain yang sama yaitu kolom `telp_1` dan `telp_2`. Hal ini menunjukkan bahwa tabel T versi pertama ini belum memenuhi syarat 1<sup>st</sup> NF.

#### Tabel 4-9 Versi ke dua

NIM	Nama_Mhs	Telepon	Kd_Jur	Nama_Jur	Kode_MK	Nama_MK	SKS	Nilai
1-01	Tukimin	0813xx, 022xxx	TE	Elektro	TE-001	Elektronika	3	A
2-01	Jamilah	0812xx, 021xxx	IF	Informatika	IF-001	Algoritma	3	B
2-02	Maemunah	0852xx, 031xxx	IF	Informatika	IF-002	Database	2	A

Tabel T versi ke dua ini juga belum memenuhi syarat 1<sup>st</sup> NF karena kolom telepon bersifat *multivalue*.

Solusi agar tabel T memenuhi syarat 1<sup>st</sup> NF adalah dengan melakukan pemecahan tabel atau dekomposisi tabel. Namun perlu diingat, dekomposisi tabel harus dilakukan dengan cermat agar data tetap konsisten (perubahan hanya terjadi pada struktur tabel tapi tidak terjadi perubahan pada data)

Perhatikan bahwa (nim) → (telepon). Dengan demikian, kita dapat memecah tabel T menjadi tabel T-1 dan tabel T-2 berikut ini:

#### Tabel 4-10 Contoh Tabel T-1

NIM	Nama_Mhs	Kd_Jur	Nama_Jur	Kode_MK	Nama_MK	SKS	Nilai
1-01	Tukimin	TE	Elektro	TE-001	Elektronika	3	A
2-01	Jamilah	IF	Informatika	IF-001	Algoritma	3	B
2-02	Maemunah	IF	Informatika	IF-002	Database	2	A

#### Tabel 4-11 Contoh Tabel T-2

NIM	Telepon
1-01	0813xx
1-01	022xxx
2-01	0812xx
2-01	021xxx
2-02	0852xx
2-02	031xxx

Baik Tabel T1 maupun tabel T2 tidak memiliki atribut bersifat *multivalue*. Tabel T1 dan T2 juga tidak memiliki lebih dari satu atribut dengan domain yang sama. Dengan demikian dapat disimpulkan bahwa tabel T1 dan T2 telah memenuhi syarat 1<sup>st</sup> NF dan siap untuk diperiksa apakah memenuhi syarat bentuk normal level berikutnya (2<sup>nd</sup> NF)

#### 4.5.2 Bentuk Normal Ke Dua (2nd Normal Form)

Kriteria 2<sup>nd</sup> NF:

- **Memenuhi 1<sup>st</sup> NF**  
Desain tabel yang tidak memenuhi syarat 1<sup>st</sup> NF sudah pasti tidak akan memenuhi syarat 2<sup>nd</sup> NF
- **Tidak ada *Partial Functional dependency***  
*Partial Functional dependency* terjadi bila (B,C) adalah *candidate key* dan  $B \rightarrow A$

Untuk lebih jelasnya perhatikan tabel T-1 hasil tahap sebelumnya:

**Tabel 4-12 Contoh T-1 hasil**

NIM	Nama_Mhs	Kd_Jur	Nama_Jur	Kode_MK	Nama_MK	SKS	Nilai
1-01	Tukimin	TE	Elektro	TE-001	Elektronika	3	A
2-01	Jamilah	IF	Informatika	IF-001	Algoritma	3	B
2-02	Maemunah	IF	Informatika	IF-002	Database	2	A

Perhatikan bahwa:

1. (nim, kode\_mk) adalah *candidate key*
2. FD1: (nim)  $\rightarrow$  (nama\_mhs, kd\_jur, nama\_jur)
3. FD2: (kode\_mk)  $\rightarrow$  (nama\_mk, sks)
4. FD3: (nim, kode\_mk)  $\rightarrow$  nilai

Berarti Terjadi *Partial Functional dependency*:

- FD 1: (nim, kode\_mk)  $\rightarrow$  (nama\_mhs, kd\_jur, nama\_jur) secara parsial
- FD 2: (nim, kode\_mk)  $\rightarrow$  (nama\_mk, sks) secara parsial

Walaupun tabel T-1 telah memenuhi syarat 1<sup>st</sup> NF namun karena terjadi *partial functional dependency* maka tabel T-1 belum memenuhi syarat 2<sup>nd</sup> NF.

Solusinya adalah dengan melakukan dekomposisi terhadap tabel T-1 dengan tetap menjaga agar datanya tetap konsisten. Hal ini dapat dilakukan dengan melakukan dekomposisi tabel sesuai FD1, FD2 dan FD3 yang telah kita analisis sebelumnya. Adapun hasil dekomposisi dari tabel T-1 adalah 3 tabel berikut ini:

**Tabel 4-13 Contoh Tabel T-1-1**

NIM	Nama_Mhs	Kd_Jur	Nama_Jur
1-01	Tukimin	TE	Elektro
2-01	Jamilah	IF	Informatika
2-02	Maemunah	IF	Informatika

**Tabel 4-14 Contoh Tabel T-1-2**

Kode_MK	Nama_MK	SKS
TE-001	Elektronika	3
DU-001	English	2
IF-001	Algoritma	3
IF-002	Database	2

**Tabel 4-15 Contoh Tabel T-1-3**

NIM	Kode_MK	Nilai
1-01	TE-001	A
1-01	DU-001	A
2-01	IF-001	B
2-01	DU-001	C
2-02	IF-002	A

Ketiga tabel hasil dekomposisi tersebut sudah tidak mengalami *partial functional dependency*. Dengan demikian ketiga tabel tersebut telah memenuhi syarat 2<sup>nd</sup> NF dan siap untuk diperiksa apakah memenuhi syarat bentuk normal level berikutnya (3<sup>rd</sup> NF).

Adapun Tabel T-2 (hasil dekomposisi pada tahap 1<sup>st</sup> NF) juga tidak mengalami *partial functional dependency* sehingga sudah memenuhi 2<sup>nd</sup> NF, tidak perlu didekomposisi lagi dan dapat langsung diperiksa apakah memenuhi 3<sup>rd</sup> NF bersama-sama dengan tabel T-1-1, T-1-2 dan T-1-3.

### 4.5.3 Bentuk Normal Ke Tiga (3rd Normal Form)

Umumnya jika sebuah tabel telah memenuhi syarat bentuk normal ke tiga (3<sup>rd</sup> NF) maka tabel tersebut sudah dianggap ‘cukup normal’. Bentuk normal ke 3 adalah bentuk normal yang biasanya menjadi syarat *minimum* bagi sebuah desain tabel walaupun akan lebih baik jika juga memenuhi BCNF.

Kriteria 3<sup>rd</sup> NF:

- **Memenuhi 2<sup>nd</sup> NF**  
Desain tabel yang tidak memenuhi syarat 2<sup>nd</sup> NF sudah pasti tidak akan memenuhi syarat 3<sup>rd</sup> NF
- **Tidak ada Transitive Functional dependency**  
*Transitive functional dependency* terjadi bila  $A \rightarrow B$  dan  $B \rightarrow C$

Untuk lebih jelasnya perhatikan tabel T-1-1 dari tahap sebelumnya:

**Tabel 4-16 Contoh tabel T-1-1**

NIM	Nama_Mhs	Kd_Jur	Nama_Jur
1-01	Tukimin	TE	Elektro
2-01	Jamilah	IF	Informatika
2-02	Maemunah	IF	Informatika

Perhatikan bahwa:

FD1: (nim)  $\rightarrow$  (nama\_mhs, kd\_jur, nama\_jur)

FD2: (kd\_jur)  $\rightarrow$  (nama\_jur)

Berarti Terjadi *Transitive FD*:

(nim)  $\rightarrow$  (nama\_jur) secara transitif melalui (kd\_jur)

Walaupun tabel T-1-1 telah memenuhi syarat 2<sup>nd</sup> NF namun karena terjadi *transitive functional dependency* maka tabel T1 belum memenuhi syarat 3<sup>rd</sup> NF. Solusinya adalah dengan melakukan dekomposisi terhadap tabel T-1-1 dengan tetap menjaga agar datanya tetap konsisten sesuai FD1 dan FD2. Adapun hasil dekomposisi dari tabel T-1-1 adalah 2 tabel berikut ini:

**Tabel 4-17 Contoh Tabel T-1-1-1**

NIM	Nama_Mhs	Kd_Jur
1-01	Tukimin	TE
2-01	Jamilah	IF
2-02	Maemunah	IF

**Tabel 4-18 Contoh Tabel T-1-1-2**

Kd_Jur	Nama_Jur
TE	Elektro
IF	Informatika
IF	Informatika

#### 4.5.4 Bentuk Normal Boyce Codd (BC Normal Form)

*Boyce Codd Normal Form* atau bentuk normal Boyce-Codd adalah bentuk normal yang levelnya di atas 3<sup>rd</sup> NF. Kriteria BCNF:

- **Memenuhi 3<sup>rd</sup> NF**  
Desain tabel yang tidak memenuhi syarat 3<sup>rd</sup> NF sudah pasti tidak akan memenuhi syarat BCNF
- **Untuk semua FD yang terdapat di tabel, ruas kiri dari FD tersebut adalah super key**  
Jika ada satu saja FD pada tabel dimana ruas kirinya bukan *super key* maka desain tabel tersebut belum memenuhi syarat BCNF. Solusinya adalah dengan melakukan dekomposisi tabel dan tetap mempertahankan konsistensi data seperti beberapa contoh pada sub bab sebelumnya

Jarang ada kasus dimana sebuah tabel memenuhi 3<sup>rd</sup> NF tapi tidak memenuhi BCNF. Umumnya sebuah tabel dikategorikan sudah ‘cukup normal’ jika sudah memenuhi kriteria BCNF. Jika tidak memungkinkan untuk memenuhi kriteria BCNF, maka 3<sup>rd</sup> NF juga sudah dianggap cukup memadai.

#### 4.5.5 Bentuk-Bentuk Normal Lainnya

Selain bentuk-bentuk normal yang sudah diperkenalkan pada beberapa sub bab sebelumnya, masih ada beberapa bentuk-bentuk normal lain. Beberapa diantaranya adalah sebagai berikut:

1. **Bentuk Normal ke-4 (4<sup>th</sup> NF)**  
diperkenalkan oleh Ronald Fagin pada tahun 1977
2. **Bentuk Normal ke-5 (5<sup>th</sup> NF)**  
diperkenalkan oleh Ronald Fagin pada tahun 1979
3. **Domain/Key Normal Form (DKNF)**  
diperkenalkan oleh Ronald Fagin pada tahun 1981
4. **Bentuk Normal ke-6 (6<sup>th</sup> NF)**  
diperkenalkan oleh Date, Darwen dan Lorentzos pada tahun 2002

#### 4.6 Denormalisasi

Denormalisasi adalah proses menggandakan data secara sengaja (sehingga menyebabkan redundansi data) untuk meningkatkan performa *database*, untuk meningkatkan kecepatan akses data atau memperkecil *query cost*.

Yang perlu diingat tentang denormalisasi adalah bahwa denormalisasi **tidak sama** dengan tidak melakukan normalisasi. Denormalisasi dilakukan setelah tabel dalam kondisi ‘cukup normal’ (mencapai level bentuk normal yang dikehendaki).



Salah satu contoh teknik Denormalisasi adalah *Materialized View* pada DBMS Oracle. *Materialized view* adalah teknik menggandakan data dengan cara membuat tabel semu berupa view fisik (yang benar-benar dituliskan di disk, bukan sebatas di *memory*). *Materialized view* biasanya dibuat dari hasil *join* beberapa tabel yang sering diakses tapi jarang diupdate.

*Materialized view* akan menyebabkan redundansi data, namun sebagai imbalannya kecepatan akses data meningkat drastis sebab data dapat langsung diakses melalui *materialized view* tanpa harus menunggu *query* menyelesaikan operasi *join* dari beberapa tabel.

Ada beberapa alasan melakukan denormalisasi:

1. Mempercepat proses *query* dengan cara meminimalkan *cost* yang disebabkan oleh operasi *join* antar tabel
2. Untuk keperluan *Online Analytical Process* (OLAP)
3. Dan lain-lain

Adapun konsekuensi denormalisasi adalah sebagai berikut:

1. Perlu ruang ekstra untuk penyimpanan data
2. Memperlambat pada saat proses insert, update dan delete sebab proses-proses tersebut harus dilakukan terhadap data yang redundant (ganda)

Dengan demikian dapat disimpulkan bahwa denormalisasi harus dilakukan dengan bijak sebab walaupun memiliki beberapa keuntungan namun juga memiliki konsekuensi yang patut diperhitungkan.



## Rangkuman

1. Normalisasi merupakan serangkaian langkah sistematis yang dilakukan terhadap desain tabel dan bertujuan untuk menghindari redundansi dan anomaly data sekaligus mengefisienkan penyimpanan data.
2. Tiga macam anomaly data: *update*, *insertion* dan *deletion anomaly*
3. *Super key* adalah sebuah atribut atau sekumpulan atribut yang dapat membedakan setiap baris data dalam tabel secara unik
4. *Candidate key* adalah minimal *super key* yang tidak mengandung *super key* yang lain
5. *Primary key* adalah *candidate key* yang dipilih untuk diimplementasikan pada DBMS
6. Setiap *primary key* pasti merupakan *candidate key*, setiap *candidate key* pasti merupakan *super key*. Hal ini tidak berlaku sebaliknya.
7.  $A \rightarrow B$  jika terpenuhi syarat bahwa untuk setiap baris data di tabel T, jika ada pasangan baris data pada kolom A memiliki nilai yang sama maka dijamin kolom B pada baris-baris tersebut juga akan memiliki nilai yang sama, namun tidak harus sebaliknya.
8. *Partial Functional dependency* terjadi bila sebuah atribut bergantung pada sebagian dari *candidate key*
9. *Transitive Functional dependency* terjadi bila sebuah atribut bergantung pada *key* melalui atribut lain
10. Normalisasi dilakukan secara bertahap, yaitu dengan menerapkan bentuk-bentuk normal dari mulai level terendah sampai mencapai level yang dikehendaki.
11. Syarat 1st NF: tidak ada atribut *multivalued* dan tidak ada lebih dari satu atribut dengan domain yang sama
12. Syarat 2nd NF: memenuhi syarat 1st NF dan tidak mengalami *partial functional dependency*
13. Syarat 3rd NF: memenuhi syarat 2nd NF dan tidak mengalami *transitive functional dependency*
14. Syarat BCNF: memenuhi syarat 3rd NF dan ruas kiri dari setiap *functional dependency*-nya adalah *super key*
15. Denormalisasi tidak sama dengan tidak melakukan normalisasi, tujuannya adalah untuk meningkatkan performa *database* sebagai imbalan dari munculnya redundansi (pengulangan data) pada *database*.

## 5 ALJABAR RELASIONAL



---

### Overview

---

**Bahasa *Query*** adalah bahasa yang dikhususkan untuk mengajukan pertanyaan atau *Query*, yang melibatkan data dalam sebuah *database*. Input dan output suatu *query* adalah relasi. *Query* dievaluasi dengan menggunakan contoh input relasi dan menghasilkan contoh output relasi. Bahasa *query* menggunakan operasi-operasi Aljabar relasional. Aljabar relasional mendefinisikan secara teoritis cara memanipulasi isi tabel dengan menggunakan delapan fungsi relasional: SELECT, PROJECT, JOIN, INTERSECT, UNION, DIFFERENCE, PRODUCT dan DIVIDE.



---

### Tujuan

---

1. Memahami dan mengerti tentang dasar-dasar Aljabar Relasional
2. Mengetahui keterkaitan antara Aljabar relasional dengan bahasa *query*
3. Memahami dan mengerti sintak dasar *query* dalam basis data.

## 5.1 Query dan Aljabar Relasional

Bahasa *query* (*Query language*) merupakan suatu bahasa yang menyediakan fasilitas bagi *user* untuk mengakses informasi dari basis data. Umumnya *query language* ini dibagi menjadi 2 (dua) kategori, yaitu:

1. Bahasa *Query* Prosedural
2. Bahasa *Query* Non Prosedural

Bahasa *query* Prosedural adalah dimana *user* menginstruksikan ke sistem agar membentuk serangkaian operasi dalam basis data untuk mengeluarkan hasil yang diinginkan. Yang termasuk dalam bahasa ini adalah Aljabar *Relational*

Bahasa *Query* Non-Prosedural adalah dimana *user* mendeskripsikan informasi yang diinginkan tanpa memberikan prosedur detail untuk menghasilkan informasi tersebut. Yang termasuk dalam bahasa ini adalah Kalkulus *Relational* Tuple.

Dalam bab ini akan dibahas Bahasa *Query* prosedural, bahasa ini memiliki sejumlah operasi yang menggunakan satu atau beberapa relasi/tabel sebagai inputan dan menghasilkan sebuah relasi/tabel sebagai outputnya. Aljabar *Relational* sebagai dasar dari bahasa ini merupakan suatu kumpulan operasi terhadap relasi dimana setiap operasi menggunakan satu atau lebih relasi untuk menghasilkan suatu relasi yang baru. Operasi-operasi dasar dalam *Aljabar Relational* sendiri dibagi menjadi 2 (dua) yaitu *Unary* dan *Binary*.

Operasi *Unary* adalah operasi yang hanya memerlukan satu operasi saja. Yang termasuk didalamnya adalah *Select*, *Project* dan *Rename*. Sedangkan operasi-operasi yang membutuhkan lebih dari satu relasi (sepasang relasi atau lebih) disebut *Binary*, contohnya adalah *Unio*, *Set Difference*, *Cartesian Product*.

Selanjutnya akan dibahas satu persatu dari masing-masing operasi-operasi diatas, baik yang termasuk dalam *unary* maupun *binary*. Sebagai acuan adalah menggunakan tabel- tabel akademik (tabel mahasiswa, tabel kuliah, tabel dosen dan tabel nilai).

## T\_kuliah

kode_mk	mata_kuliah	sks	semester
IK_330	Database	3	3
IK_331	Aljabar_Relational	2	3
IK_335	Kalkulus	3	1
IK_335	Citizenship	2	2

## T\_Dosen

Kd_dosen	Nama_dosen	Alamat	Tanggal_Lahir
1234	Mister A	Bandung	10-12-1976
1235	Mister B	Malang	09-11-1977
1236	Mister C	Bandung	10-07-1978
1237	Ibu A	Surabaya	05-12-1970

## T\_mahasiswa

NIM	Nama	Tempat_lahir	Tanggal_Lahir
3010001	Ahmad	Jakarta	10-12-1982
3010003	Nita	Surabaya	09-11-1982
3010054	Richard	Medan	10-07-1983
3010066	Amelia	Jakarta	05-12-1983

## T\_nilai

Kode_mk	NIM	Indeks
IK_330	3010001	A
IK_331	3010003	A
IK_335	3010003	C
IK_335	3010054	B

## 5.2 Operasi Select

Operasi ini digunakan untuk mengambil sejumlah baris data yang memenuhi predikat yang diberikan. Dimana predikat tersebut sesuai dengan kondisi yang ingin diperoleh dalam operasi ini. Sintaks untuk operasi ini adalah sebagai berikut,

$$\sigma_P = (E1)$$

Dimana P adalah predikat dari atribut di E1. Sehingga jika ingin mengambil bari-baris data mahasiswa yang lahir di kota 'Jakarta' dapat kita buat sintaks seperti dibawah ini

$$\sigma_{\text{tempatlahir} = \text{"Jakarta"}} (\text{mahasiswa})$$

Pada dasarnya predikat merupakan suatu ekspresi logik sehingga dapat menggunakan operator-operator logik seperti =, <, >, dan yang lainnya.

### 5.3 Operasi Project

Operasi ini digunakan untuk menampilkan *field-field* dari sebuah tabel atau relasi yang diinginkan. Sintaks untuk operasi ini adalah sebagai berikut,

$$\pi_{\langle \text{daftar atribut} \rangle} (\langle \text{nama tabel} \rangle) \text{ atau } \pi_s (E1)$$

dimana  $s$  atau daftar atribut yang berisi satu atau lebih *field* yang ingin ditampilkan dari  $E1$ .

Sehingga jika kita ingin menghasilkan tampilan yang berisi data NIM dan Nama\_mahasiswa maka *query*-nya adalah sebagai berikut,

$$\pi_{\langle \text{NIM, Nama\_mahasiswa} \rangle} (\text{mahasiswa})$$

Sebagai catatan, operasi yang dapat diprojektikan bukan hanya dari tabel tetapi bisa juga dari suatu operasi/*query*.

### 5.4 Operasi Cartesian Product

Operasi ini adalah operasi yang bisa digunakan untuk menggabungkan data dari dua buah tabel atau hasil *query*. Sintak yang digunakan adalah

$$E1 \times E2$$

Dimana semua *record* di  $E1$  akan digabungkan dengan *record* di  $E2$  dan hasilnya akan menampilkan semua *record* yang ada di  $E1$  dan  $E2$ .

Pada umumnya operasi *cartesian product* ini tidak berdiri sendiri, biasanya dikombinasikan atau digabung dengan operasi *select* dan *project* dengan semua ketentuannya sesuai dengan apa yang ingin ditampilkan sebagai hasil *query*-nya.

Sebagai contoh, saat akan menampilkan hasil dari tabel kuliah dan tabel nilai yang mendapatkan indeks “A”, maka operasinya dapat kita tuliskan sebagai berikut:

$$\sigma_{\text{indeks} = \text{“A”}} (t_{\text{nilai}} \times t_{\text{kuliah}})$$

Pada dasarnya untuk melihat adanya satu keterhubungan antara satu tabel dengan tabel yang lainnya adalah ditandai dengan adanya satu *field* yang sama. Misal antara  $t_{\text{kuliah}}$  dan  $t_{\text{nilai}}$  (tabel kuliah dan tabel nilai) sama-sama memiliki *field*  $\text{kode\_mk}$ .

Jika ingin menampilkan hasil dari suatu *query* dimana tidak ada relasi langsung antar tabel-tabel yang terkait (tidak ada *field* yang sama) maka bisa

kita libatkan tabel lain yang memiliki keterhubungan antara tabel tersebut. Contoh untuk menghasilkan list mahasiswa yang mengambil mata kuliah IK\_330. Tabel kuliah dan tabel mahasiswa tidak memiliki keterhubungan secara langsung tetapi keterhubungannya dapat kita lihat dari tabel nilai. Sehingga kita bisa membuat suatu *query* seperti dibawah ini:

$$\pi_{\langle \text{NIM} \rangle} (\sigma_{\text{t\_kuliah.kode\_mk}=\text{t\_nilai.kode\_mk} \wedge \text{kode\_mk} = \text{"IK\_330"}} (\text{t\_nilai} \times \text{t\_kuliah}))$$

Dari contoh diatas dapat terlihat bahwa Operasi *Cartesian Product* tidak berdiri sendiri tetapi melibatkan operasi-operasi yang lainnya untuk mendapatkan hasil *query* yang kita inginkan.

## 5.5 Operasi Unio

Operasi *Unio* adalah operasi yang menggabungkan semua baris dari dua buah tabel dan kedua tabel tersebut harus sesuai atau memiliki hasil proyeksi yang sama. Dimana akan menghasilkan tabel ketiga. Operasi *Unio* disimbolkan sebagai berikut :

**T1 U T2**

Contoh lainnya, adalah pada t\_dosen dan t\_mahasiswa terdapat *field* tempat\_lahir. Operasi *Projection* (proyeksi) untuk masing-masing tabel adalah

**$\Pi_{\text{tempat\_lahir}}(\text{t\_dosen})$   
dan  
 $\Pi_{\text{tempat\_lahir}}(\text{t\_mahasiswa})$**

Sehingga jika kita ingin memproyeksikan kedua tabel diatas maka kita bisa menggunakan operasi *Unio* dengan bentuk sebagai berikut :

**$\Pi_{\text{tempat\_lahir}}(\text{t\_dosen}) \cup \Pi_{\text{tempat\_lahir}}(\text{t\_mahasiswa})$**

Untuk melakukan suatu operasi *unio* tabel-tabel tersebut harus bisa *unio compatible*. Tabel disebut *Unio Compatible* jika :

1. T1 dan T2 harus memiliki jumlah atribut/*field* yang sama
2. *Field* dari T1 dan *field* dari T2 harus bersala dari domain yang sama.

## 5.6 Operasi Set Difference

Operasi ini merupakan kebalikan dari operasi *unio*, dimana terjadi pengurangan data ditabel pertama oleh data dari tabel kedua. Simbol dari operasi ini adalah sebagai berikut :

### $T1 - T2$

Dapat dikatakan juga bahwa operasi adalah operasi terhadap relasi yang terdiri dari semua baris di T1, tetapi tidak ada di T2. Operasi ini juga memiliki syarat yang sama dengan operasi *Unio* yaitu harus *Unio Compatible*.

Sebagai contohnya perhatikan tabel dibawah ini

T\_kuliah\_Ilkom

Kode_mk	Mata_kuliah	SKS	Semester
IK_002	Kalkulus	3	1
IK_003	Aljabar Linear	2	2
TU_001	<i>Database</i>	4	2
IK_004	Java	3	3

T\_kuliah\_PIlkom

Kode_mk	Mata_kuliah	SKS	Semester
IK_002	Kalkulus	3	1
IK_003	Aljabar Linear	2	2
TU_001	English 2	2	2
IK_335	<i>Software</i>	3	3

Operasi yang dikenakan pada 2 (dua) tabel diatas adalah

**$\Pi_{mata\_kuliah}(t\_kuliah\_ilkom) - mata\_kuliah(t\_kuliah\_pilkom)$**

Maka hasilnya *query* diatas adalah sebagai berikut:

Mata_kuliah
<i>Database</i>
Java

Dari contoh diatas didapatkan bahwa operasi ini akan menemukan tuple-tuple yang berada pada satu relasi tetapi tidak berada pada relasi yang Lainnya



### 5.7 Operasi *Intersection*

Operasi *intersection* adalah operasi yang mendapatkan atau menyatakan irisan dari dua buah tabel/*query*. Operasi ini disimbolkan dengan menggunakan lambang sebagai berikut :

$$T1 \cap T2$$

Selain simbol diatas dapat disimbolkan juga dengan bentuk *set difference* seperti dibawah ini :

$$T1 - (T1-T2)$$

$T1 \cap T2$  menghasilkan suatu relasi yang berisi instan – instan yang terjadi baik pada  $T1$  dan  $T2$ . Relasi  $T1$  dan  $T2$  harus *Unio – Compatible*.

### 5.8 Operasi *rename*

Dalam operasi himpunan *Cross – Product*, bisa menimbulkan terjadinya konflik penamaan , karena *Cross – Product* bisa menghasilkan suatu relasi dari 2 relasi dengan skema yang sama, sehingga skema hasil akan muncul *field* dengan nama yang sama.


Operasi *Rename* ( $\rho$ ) digunakan untuk menghindari terjadinya Konflik Penamaan tersebut. Operasi *Rename* dibutuhkan untuk melakukan penamaan kembali pada suatu tabel atau hasil proyeksi agar dapat menunjukkan acuan yang jelas dalam sebuah operasi yang lengkap, khususnya yang melibatkan dua/lebih *sumber data* yang sama.

Simbol untuk operasi *rename* adalah sebagai berikut:

$$\rho(R(\overline{F}), (E))$$

Dimana  $R$  = Relasi,  $F$  = Daftar Renaming,  $E$  = Aljabar Relasional

### 5.9 *Join*

Merupakan operasi yang digunakan untuk mengabungkan informasi dari dua atau lebih relasi (). Selain itu operasi *join* Memungkinkan kita untuk mengkom- binasikan informasi dari dua tabel atau lebih. *JOIN* memiliki kemampuan nyata untuk mendukung basis data relasional, memungkinkan

penggunaan tabel inde-penden yang dihubungkan melalui atribut yang sama. Dimana notasi untuk *join* adalah sebagai berikut:

**R |x|<kondisi join> S**

Kondisi *join* dalam bentuk:

**<kondisi> AND <kondisi> AND ... AND <kondisi>**

Operator pembandingan yang digunakan dalam operasi *join* yaitu  $\{=, <, \leq, >, \geq, \neq\}$

Operasi *join* sendiri memiliki beberapa tipe lagi, untuk yang pertama adalah *Natural join* dimana *Natural JOIN* menghubungkan tabel dengan memilih hanya record dengan nilai yang digunakan bersama-sama pada atribut yang sama. Operator ini akan menghasilkan tiga tahapan proses:

- *PRODUCT*
- *SELECT*
- *PROJECT*

Operasi *join* yang lain adalah *EquiJOIN* menghubungkan tabel didasarkan pada kondisi yang sama dengan membandingkan kolom tertentu setiap tabel. Hasil *equiJOIN* tidak menghilangkan kolom duplikat dan kondisi atau kriteria penggabungan tabel harus terdefinisi secara eksplisit.

Theta *JOIN* adalah *equiJOIN* yang membandingkan kolom tertentu setiap tabel menggunakan operator pembandingan selain operator sama dengan.

Pada *Outer JOIN* pasangan data yang tidak cocok akan tetap dipertahankan dan nilai untuk tabel lainnya yang tidak cocok akan dibiarkan kosong. *Outer join* sendiri terdiri dari tiga jenis, yaitu *left outer join*, *right outer join*, dan *full outer join*.

*Left outer join* akan menjadikan tabel/hasil *query* disebelah kiri simbol operasi, sehingga dapat melihat semua baris data di tabel/hasil *query* yang pertama baik yang memiliki relasi ataupun tidak dengan tabel/hasil *query* kedua.

Operasi *right outer join* merupakan kebalikan dari operasi *left outer join*. Sedangkan operasi *full outer join* merupakan gabungan dari operasi *left*

*outer join* dan *right outer join*. Dimana operasi ini akan menghasilkan semua baris data di kedua tabel/hasil *query* yang memiliki relasi ataupun tidak.

5.10 Fungsi Agregasi

Fungsi ini dapat dikatakan sebagai fungsi-fungsi yang akan melakukan penggabungan nilai dari suatu *query* dan kemudian menyajikannya sebagai sebuah nilai. Sedangkan fungsi-fungsi ini dapat dijalankan jika atau pada atribut-atribut yang bersifat numerik. Secara umum ada 5 fungsi agregasi, yaitu sebagai berikut :

- 1. *SUM* : menjumlah nilai dari suatu atribut
- 2. *AVERAGE* : mencari rata-rata nilai dari suatu atribut
- 3. *MAXIMUM* : mencari nilai paling besar dari suatu atribut
- 4. *MINIMUM* : mencari nilai paling besar dari suatu atribut
- 5. *COUNT* : menghitung jumlah record

Untuk fungsi yang terakhir yaitu fungsi *count* terdapat perluasan dalam fungsi tersebut yaitu yang diberi nama ***Count\_distict***. Fungsi *Count\_distict* akan menghitung nilai atribut yang unik.

5.11 Operasi Division

Operasi divisi cocok digunakan untuk query yang menginginkan adanya pernyataan “ untuk semua”. Misalkan untuk setiap nilai x di dalam R, perhatikan kumpulan nilai y yang muncul dalam instan R dengan nilai x tersebut. Jika kumpulan ini berisi semua nilai y dalam S, maka nilai x adalah hasil dari ***R / S***. Simbol dari operasi ini adalah “/” atau “÷”.

Contoh, perhatikan tabel dibawah ini:

$\pi_{A,B}(T1)$

A	B
a1	b1
a2	b1
a2	b2
a3	b2

$\pi_B(\sigma_{B=b1}(T1))$

$\pi_B(T1)$

B
b1
b2

$\pi_{A,B}(T1) / \pi_B(T1)$

B
b1

A
a2

$\pi_{A,B}(T1) / \pi_B(\sigma_{B=b1}(T1))$

A
a1
a2



## Rangkuman

1.  $\rho x(T1)$ , dimana  $x$  adalah nama baru untuk hasil T1 dan T2 adalah ekspresi aljabar-relasional, maka berikut ini adalah semua ekspresi aljabarrelasional :
  - a.  $T1 \cup T2$
  - b.  $T1 - T2$
  - c.  $T1 \times T2$
  - d.  $\sigma p(T1)$ , dimana  $p$  adalah sebuah predikat untuk atribut-atribut dalam  $T1$ .
  - e.  $\pi s(T1)$ , dimana  $s$  adalah daftar yang terdiri dari beberapa atribut dalam  $T1$ .
  - f.  $\rho x(T1)$ , dimana  $x$  adalah nama baru untuk hasil  $T1$
  - g. Operasi *join* terdiri dari *natural join*, *equiJoin*, *outer join*
  - h. *Outer join* dibagi menjadi *left outer join*, *right outer join* dan *full outer join* yang merupakan kombinasi dari *left* dan *right outer join*.
2. Operasi *join* terdiri dari *natural join*, *equiJoin*, *outer join*
3. *Outer join* dibagi menjadi *left outer join*, *right outer join* dan *full outer join* yang merupakan kombinasi dari *left* dan *right outer join*.
4. Operasi lainnya adalah *aggregate function* terdiri dari operasi *Sum*, *Average*, *Maximum*, *Minimum*, dan *Count*. Untuk operasi tambahan dari *count* adalah operasi yang dinamakan dengan *Count\_distinct*
5. Untuk Operasi terakhir dalam bahasan ini adalah operasi *Division* dimana bisa digunakan untuk operasi yang bersifat “ untuk semua”

## 6 BAHASA BASIS DATA



### Overview

---

*Structure Query Language* (SQL) merupakan bahasa standar komputer dalam proses pengolahan data pada basisdata terutama untuk RDBMS. SQL ini telah mengalami jalan panjang sehingga terbentuk standarisai perintah-perintah yang ada. Perintah-perintah SQL mencakup perintah untuk membentuk *database* dan struktur tabel, merubah *database* dan struktur tabel, *maintenance* konten data pada *database*.



### Tujuan

---

1. Mahasiswa memahami fungsi dari SQL
2. Mahasiswa memahami dan mengetahui sejarah terbentuknya SQL
3. Mahasiswa memahami dan mengerti bentuk-bentuk standarisasi SQL
4. Mahasiswa mampu membuat *database* dan struktur tabel dengan menggunakan perintah SQL
5. Mahasiswa mampu melakukan perubahan pada struktur tabel dengan menggunakan perintah SQL
6. Mahasiswa mampu untuk menambahkan data pada tabel, merubah data pada tabel dan menghapus data pada tabel dengan menggunakan perintah SQL
7. Mahasiswa mampu untuk menampilkan data dari dalam tabel dengan berbagai macam bentuk, kondisi dan urutan data.

## 6.1 Pendahuluan

*Relational Database Management System* (RDBMS) seperti Oracle, Microsoft SQL Server, PostgreSQL dan MySQL merupakan komponen utama dalam sebuah sistem informasi saat ini bahkan untuk sistem-sistem yang berbasis intranet dan internet

*Structure Query Language* (SQL) hadir dalam RDBMS sebagai bahasa untuk mengakses, *me-maintenance* kumpulan dari *table*-tabel data yang dihubungkan oleh masing-masing *key*.

Jadi SQL dapat diartikan sebagai sebuah bahasa komputer untuk basisdata dimana dapat melakukan proses-proses seperti pembuatan dan perubahan struktur basisdata, perubahan konten data, mendapatkan informasi dari basisdata, perubahan setting *security* dan memberikan hak akses kepada pengguna basisdata

Secara umum perintah-perintah SQL dibagi dalam tiga bagian utama yaitu:

1. *Data Definition Language* (DDL)
2. *Data Manipulation Language* (DML)
3. *Data Control Language* (DCL)

## 6.2 Standarisasi SQL

SQL muncul pertama kali diawal tahun 1970-an sebagai sebuah hasil penelitian pada lab IBM, San Jose, California yang dilakukan oleh Donald C Meserly dan Raymond F Boyce. Versi pertama ini diberikan nama SEQUEL (*Structure English Query Language*). Rencana awalnya SEQUEL ini akan digunakan dalam DB2 sebagai sebuah produk RDBMS dari IBM. Dikemudian hari IBM memberikan nama lain pada SEQUEL ini dengan nama SQL

Hasil penelitian tersebut menarik banyak perhatian, banyak perusahaan yang tertarik untuk membangun RDBMS berdasarkan pada SQL ini. Salah satunya adalah Oracle. Dan Oracle ini dinyatakan sebagai sebuah produk SQL komersial pertama.

Berdasarkan perkembangan hal diatas *American National Standard Institute* (ANSI) dan *International Standard Organization* (ISO) membentuk standar SQL yang dimulai tahun 1986. Standar SQL ini lebih dikenal dengan SQL86. Kemudian SQL86 ini diperbaharui lagi tahun 1992 yang diberi nama SQL92 dan standar berikutnya dikeluarkan tahun 1999 yang dikenal dengan SQL99.

Perkembangan saat ini server-server basisdata tidak mengacu sepenuhnya pada standar SQL tersebut. Masing-masing server basisdata mengembangkan dialek masing-masing. Beberapa dialek SQL ini diantaranya:

1. PL/SQL

Dibuat oleh Oracle sebagai bahasa *procedural* berbasis pada SQL yang mengacu pada bahasa pemrograman ADA. PL/SQL ini menjadi bagian utama dalam perkembangan Oracle saat ini.

2. *Transact SQL*

Dialek yang dikembangkan oleh Microsoft yang membuat sebuah produk DBMS yang diberi nama Microsoft SQL Server. *Transact SQL* ini merupakan hasil kerjasama dengan Sybase Adaptive Server.

3. PL/PgSQL

Seperti juga pada Oracle, PostgreSQL mengembangkan bahasa *procedural* tersendiri yang lebih jauh berkembang sehingga memungkinkan untuk mengakomodasi *Object Oriented Database Management System*

SQL92 membagi secara nyata perintah-perintah SQL ini dalam tiga kategori besar yang diberi nama kelompok : *Data Manipulation language* (DML), *Data Definition Language* (DDL) dan *Data Control Language* (DCL).

DML menyediakan perintah-perintah untuk melakukan proses manipulasi pada konten data yang ada dalam basisdata. Perintah-perintah ini diantaranya: *Select*, *Insert*, *Update* dan *Delete*.

DDL menyediakan perintah-perintah untuk mengakses objek-objek basisdata seperti *database*, *table*, *index* PL/*Transact* baik untuk membuat dengan perintah *Create*, merubah dengan perintah *Alter* dan menghapus dengan perintah *Drop*.

Sementara DCL menyediakan perintah-perintah untuk mengatur hak akses terhadap objek-objek data dan konten data untuk masing-masing pengguna dan *layer user* dalam basisdata.

Pada SQL99 pembagian kategori perintah didasarkan *class-class (object)* yang terkait satu dengan yang lainnya. Terdapat tujuh kelas dalam SQL99 yaitu: *SQLConnection*, *SQLControl*, *SQLData*, *SQLDiagnostic*, *SQLSchema*, *SQLSession*, *SQLTransaction*.

Standarisasi SQL berikutnya SQL:2003 yang memperkenalkan *Extended Markup Language* (XML) dan *auto-generator column values*. Tahun 2006 muncul lagi SQL:2006 yang ditambahkan kemampuan untuk lebih fleksibel dalam pemrograman dasidata didalam *web* dengan membuat *XQuery* yang



mengabungkan antara SQL dan XML. Terakhir tahun 2008 terdapat standar baru SQL:2008

### 6.3 Membangun Basisdata

Pembangunan basisdata pada SQL92 termasuk dalam kategori DDL dimana perintah-perintah dasarnya terdiri dari *CREATE*, *ALTER* dan *DROP*. Membangun basisdata berarti akan membuat basisdata sebagai wadah utama, tabel-tabel data yang saling berelasi dengan kolom kunci dari masing-masing tabel dan jika dibutuhkan dibuat *index* tabel.

#### 6.3.1 Membuat BasisData

Struktur umum penulisan perintah untuk membuat basisdata ini adalah *CREATE DATABASE nama\_database*

Misalkan di Politeknik telkom akan dibangun beberapa basisdata yang terpisah untuk akademik, keuangan dan SDM maka perintah SQL yang perlu dibuat adalah:

```
CREATE DATABASE Akademik
CREATE DATABASE Keuangan
CREATE DATABASE SDM
```

Struktur penulisan *CREATE DATABASE* ini untuk setiap dialek SQL berbeda-beda sehingga nantinya perlu disesuaikan dengan struktur penulisan SQL dari dialek tersebut.

#### 6.3.2 Membuat Tabel Data

Pada basisdata yang sudah dibentuk akan ditempatkan beberapa tabel yang saling terkait satu dengan yang lainnya. Keterkaitan antar tabel ini ditandai dengan adanya *key* (*Primary key* dan *Foreign key*).

Struktur umum penulisan perintah untuk membuat tabel yaitu :

```
CREATE TABLE nama_tabel ( [kolom_data] )
```

nama\_tabel = nama yang diberikan untuk basisdata tersebut

Kolom\_data digunakan untuk mendefisikan kolom-kolom data yang ada dalam tabel tersebut. kolom data ini dapat diuraikan kembali sbb:

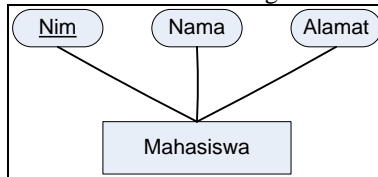
```
Kolom_data= definisi_kolom, [primary_key, konstrain]
```

Definisi\_kolom = [nama\_kolom] [tipe\_data] {Null, Not Null} {Option Kolom}

Primary\_key = *PRIMARY KEY* (kolom\_key [,kolom\_key])

Konstrain = *CONSTRAINT* {definisi konstrain}

Contoh akan dibuatkan tabel Mahasiswa sebagai berikut:



**Gambar 6-1 Entitas Mahasiswa**

**Tabel 6-1 Tabel Data Mahasiswa**

<u>NIM</u>	<u>NAMA</u>	Alamat
32121001	Aji	Jl. Abc 123
32121002	Bayu	Jl. Melati 10

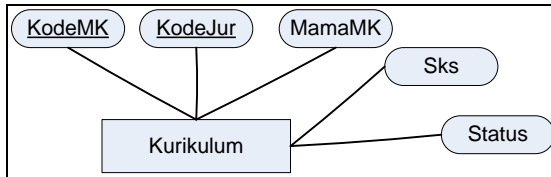
Perintah SQL nya sbb:

```
CREATE TABLE Mahasiswa (
  Nim CHAR(10) PRIMARY KEY,
  Nama VARCHAR(50) NOT NULL,
  Alamat VARCHAR(75) NULL
)
```

Atau dapat dituliskan sbb:

```
CREATE TABLE Mahasiswa (
  Nim CHAR(10) NOT NULL,
  Nama VARCHAR(50) NOT NULL,
  Alamat VARCHAR(75) NULL,
  PRIMARY KEY (Nim)
)
```

Contoh lain akan dibuat tabel kurikulum yang mempunyai *primary key* KodeMK dan KodeKur.

**Gambar 6-2 Entitas Kurikulum****Tabel 6-2 Data Kurikulum**

<u>KodeMK</u>	<u>KodeKur</u>	Nama Matakuliah	Sks	Status
SI101	SI08	Algoritma Pemrograman	2	1
SI102	SI08	Sistem Komputer	3	1

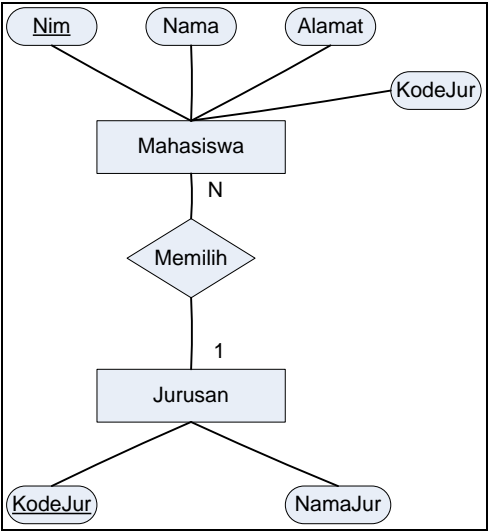
Perintah SQL nya sbb:

```
CREATE TABLE Kurikulum (
    KodeMK CHAR(5) NOT Null,
    KodeKur CHAR(4) NOT Null,
    NamaMK VARCHAR NOT Null,
    Sks INTEGER,
    STATUS CHAR(1),
    CONSTRAINT pk_kode_mk_kode_kur PRIMARY KEY
(kodeMK,KodeKur)
)
```

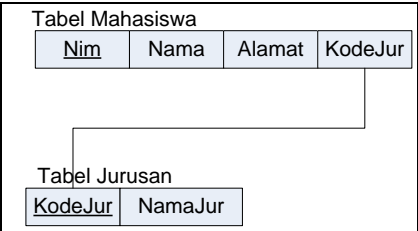
Atau dapat dituliskan sbb:

```
CREATE TABLE Kurikulum (
    KodeMK CHAR(5) NOT Null,
    KodeKur CHAR(4) NOT Null,
    NamaMK VARCHAR Not Null,
    Sks INTEGER,
    STATUS CHAR(1),
    PRIMARY KEY (KodeMK, KodeKur)
)
```

Kalau tadi sudah dibentuk satu tabel dengan *primary key*, berikut akan dibentuk tabel yang mempunyai *Foreign key*. Sebagai contoh akan dibuat tabel Jurusan dan tabel mahasiswa sebagai berikut:



Gambar 6-3 Entitas Mahasiswa dan Jurusan



Gambar 6-4 Relasi Tabel Mahasiswa dan Tabel Jurusan

Tabel 6-3 Tabel Data Jurusan

<u>KodeJur</u>	NamaJur
12	Teknik Komputer
22	Komputer Akuntansi
32	Manajemen Informatika

Tabel 6-4 Tabel Data Mahasiswa

<u>NIM</u>	NAMA	Alamat	<u>KodeJur</u>
32121001	Aji	Jl. Abc 123	32
32121002	Bayu	Jl. Melati 10	32
12121001	Sonny	Jl. Ahmad Yani 121	12

12121002	Putra	Jl. Terate 15	12
22121001	Ali	Jl. Suka 12	22

Perintah SQL nya sbb:

```
CREATE TABLE Jurusan (  
  KodeJur CHAR(2) PRIMARY KEY,  
  NamaJur VARCHAR(25) NOT NULL  
)
```

```
CREATE TABLE Mahasiswa (  
  Nim CHAR(10) PRIMARY KEY,  
  Nama VARCHAR(50) NOT NULL,  
  Alamat VARCHAR(75) NULL,  
  KodeJur CHAR(2) FOREIGN KEY REFERENCES  
  Jurusan(KodeJur)  
)
```

Aatau dapat dituliskan sbb:

```
CREATE TABLE Mahasiswa (  
  Nim CHAR(10) PRIMARY KEY,  
  Nama VARCHAR(50) NOT NULL,  
  Alamat VARCHAR(75) NULL,  
  KodeJur CHAR(2),  
  CONSTRAINT fk_mhs_jur FOREIGN KEY (KodeJur)  
  REFERENCES Jurusan(KodeJur)  
)
```

Dalam proses membuat tabel, harus ditentukan dengan tepat jenis data dan ukuran data yang digunakan untuk masing-masing kolom tersebut. Dalam contoh diatas baru beberapa tipe data yang terlihat ada *Varchar*, *Char* dan *integer*.

Masing-masing dialek SQL mendefinisikan jenis data masing-masing dan satu dengan yang lainnya banyak yang berbeda. Oleh sebab itu dalam pembuatan tabel perlu diperhatikan tujuan basisdata tersebut akan dibentuk untuk server basisdata apa.

Berikut akan ditampilkan tabel jenis data untuk Oracle sbb:

## Jenis Data

**Tabel 6-5 Jenis Data pada Oracle**

Type Data	Keterangan
BOOLEAN	Data logikal dengan nilai TRUE atau FALSE.
DATE	mulai 1 Januari 4712 SM sampai dengan 31 Desember 9999.
NUMBER [(p [,s])]	Tipe data numerik dengan p angka penting dan sejumlah s angka penting di belakang koma. Nilai p adalah integer dengan nilai maksimal 38 dan nilai s berada pada rentang - 84 sampai dengan 127. Nilai s negatif berarti pembulatan sampai dengan 10s terdekat.
FLOAT	Turunan dari NUMBER. Presisi sampai dengan 38 digit.
DOUBLE PRECISION	Sama dengan FLOAT.
REAL	Turunan dari number. Presisi sampai dengan 18 digit.
DEC [(p [,s])]	Sama dengan NUMBER [(p [,s])].
DECIMAL [(p [,s])]	Sama dengan NUMBER [(p [,s])].
NUMERIC [(p [,s])]	Sama dengan NUMBER [(p [,s])].
INTEGER [(n)]	Sama dengan NUMBER [(n,0)].
INT [(n)]	Sama dengan NUMBER [(n,0)].
SMALLINT [(n)]	Sama dengan NUMBER [(n,0)].
BINARY_INTEGER	Tipe variabel ini digunakan menyimpan nilai mulai dari -2.147.483.647 s/d 2.147.483.647
NATURAL	Bagian dari binary integer, mampu menyimpan mulai dari 0 s/d 2.147.483.647.
NATURALN	Bagian dari binary integer, mampu menyimpan mulai dari 0 s/d 2.147.483.647. Tipe data ini tidak boleh bernilai NULL.
POSITIVE	Bagian dari binary integer, mampu menyimpan mulai dari 1 s/d 2.147.483.647
POSITIVEN	Bilangan integer dengan rentang nilai 1 sampai dengan 2147483647. Tipe data ini tidak boleh bernilai NULL.
SIGNTYPE	Tipe data bilangan yang bernilai -1, 0 atau 1.
PLS_INTEGER	Bilangan integer dengan rentang nilai -2147483647 sampai 2147483647.
VARCHAR2(n)	Data karakter dengan panjang tidak tetap. Nilai n <i>minimum</i> sama dengan 1 dan maksimum sama dengan 32767 byte.
VARCHAR(n)	Sama dengan VARCHAR2(n).
CHAR [(n)]	Data karakter dengan panjang tetap sebesar n byte. Nilai n maksimum adalah 32767. Nilai n <i>minimum</i> dan juga nilai default adalah 1.
STRING(n)	Sama dengan VARCHAR2(n).

CHARACTER [(n)]	Sama dengan CHAR(n).
LONG [(n)]	Data karakter dengan panjang tidak tetap. Nilai n maksimum sama dengan 32760 byte.
NCHAR [(n)]	Data karakter dengan panjang tetap. Panjang maksimum sama dengan 32767 byte. maksimum bergantung pada national character set yang dipakai. Nilai default adalah 1.
NVARCHAR2(n)	Data karakter dengan panjang tidak tetap. Panjang maksimum sama dengan 32767 byte. Nilai n maksimum bergantung pada national character set yang dipakai.
RAW(n)	Data binary dengan panjang tidak tetap. Nilai n maksimum sama dengan 32767 byte.
LONG RAW [(n)]	Data binary dengan panjang tidak tetap. Nilai n maksimum sama dengan 32760 byte.
ROWID	Identitas baris pada suatu tabel- <i>index</i> yang dinyatakan dengan string heksa desimal. Identitas tersebut menunjukkan posisi baris data. Tipe data ini merupakan balikan dari kolom palsu ROWID.
UROWID [(n)]	Identitas baris pada suatu tabel- <i>index</i> yang dinyatakan dengan string heksa desimal. Nilai n adalah ukuran kolom UROWID. Nilai n maksimum adalah 4000 byte.
BFILE	Tipe data large object untuk data <i>file</i> .
BLOB	Tipe data large object untuk karakter binary.
CLOB	Tipe data large object untuk karakter satu byte.
NCLOB	Tipe data large object untuk karakter multi byte.

### 6.3.3 Melakukan Perubahan pada Tabel

Perubahan pada tabel sering terjadi sehubungan dengan perubahan pada rancangan basisdata yang dilakukan. Perubahan ini dapat dilakukan untuk menambah kolom, merubah kolom, menghapus kolom, menambahkan *key* dan menghapus *key* (*primary key*, *foreign key*)

Perubahan pada tabel dilakukan dengan menggunakan perintah sbb:

```
ALTER TABLE Nama_tabel
[Add kolom] [Alter Kolom] [drop kolom]
```

#### Menambahkan Kolom

Pada tabel mahasiswa akan ditambahkan kolom tahun masuk. Perintah SQLnya sbb:

```
ALTER TABLE mahasiswa
ADD TahunMasuk integer
```

Berikut akan ditambahkan dua kolom sekaligus untuk yaitu kolom StatusAkademik dan Kelas Kuliah. Perintah SQLnya sbb:

```
ALTER TABLE Mahasiswa  
  ADD StatusAkd CHAR(1),  
  Kelas Char(2)
```

### **Merubah Kolom**

Akan dilakukan perubahan pada panjang data Alamat dari 75 karakter menjadi 100 karakter. Perintah SQLnya sb:

```
ALTER TABLE Mahasiswa  
  ALTER Column Alamat VARCHAR(100)
```

### **Menghapus Kolom**

Akan dilakukan penghapusan kolom kelas pada tabel Mahasiswa karena akan dipindahkan pada tabel lainnya. Perintah SQLnya sbb:

```
ALTER TABLE Mahasiswa  
  DROP Column kelas
```

### **Menghapus *Primary key***

Penghapusan *primary key* mungkin saja terjadi untuk dilakukan perubahan pada tabel tersebut yang sifatnya sementara yang kemudian akan dibentuk kembali *primary key* yang barunya. Karena prinsip utama pada RDBMS setiap tabel harus mempunyai *primary key*.

Misalkan akan dihapus *primary key* pada tabel mahasiswa, perintah SQLnya sbb:

```
ALTER TABLE Mahasiswa  
  DROP CONSTRAINT pk_mahasiswa
```

*CONSTRAINT* pk\_mahasiswa didasarkan pada pembentukan struktur tabel diawal

```
CREATE TABLE Mahasiswa (  
  Nim CHAR(10) NOT NULL,  
  Nama VARCHAR(50) NOT NULL,
```



```
Alamat VARCHAR(75) NULL,  
KodeJur CHAR(2),  
CONSTRAINT pk_mahasiswa PRIMARY KEY (Nim)  
)
```

### **Menambahkan *Primary key***

Akan ditambahkan *primary key* untuk tabel mahasiswa yang tadi telah dihapus *primary key*. *Primary key* nya tetap pada kolom Nim. Perintah SQLnya sbb:

```
ALTER TABLE Mahasiswa  
ADD CONSTRAINT pk_mahasiswa PRIMARY KEY (Nim)
```

### **Menghapus *Foreign Key***

Penghapusan *foreign key* berarti memutus hubungan antar tabel yang sudah saling terkait. Perintah SQL yang dibuat hampir sama dengan penghapusan *primary key*. Perintah SQL untuk melakukan penghapusan *foreign key* adalah sbb:

```
ALTER TABLE Mahasiswa  
DROP CONSTRAINT fk_mhs_jur
```

*CONSTRAINT* pk\_mhs\_jur didasarkan pada pembentukan struktur tabel diawal

```
CREATE TABLE Mahasiswa (  
Nim CHAR(10) PRIMARY KEY,  
Nama VARCHAR(50) NOT NULL,  
Alamat VARCHAR(75) NULL,  
KodeJur CHAR(2),  
CONSTRAINT fk_mhs_jur FOREIGN KEY (KodeJur) REFERENCES  
Jurusan(KodeJur) )
```

### **Menambahkan *Foreign Key***

Akan ditambahkan *foreign key* untuk tabel mahasiswa yang tadi telah dihapus hubungan dengan tabel jurusan nya. Perintah SQLnya sbb:

```
ALTER TABLE Mahasiswa  
ADD CONSTRAINT fk_mhs_jur FOREIGN KEY (KodeJur) REFERENCES  
Jurusan(KodeJur)
```

## 6.4 Maintenance Data pada Basisdata

Pada bagian atas sudah dibentuk tabel-tabel data yang ada dalam sebuah basisdata. Langkah berikutnya adalah akan mengisi tabel-tabel data tersebut dengan konten data. Konten data ini datang dari proses transaksi yang yang di masukan lewat sebuah form yang biasanya ditangani lewat sebuah program aplikasi.

Perintah SQL yang diganakan dalam *maintenance* data ini adalah yang termasuk dalam Data *Manipulation Language* (DML) yaitu : *INSERT*, *UPDATE*, *DELETE* dan *SELECT*.

### 6.4.1 Memasukan Data

Struktur umum penulisan perintah untuk menambahkan data adalah sbb:

*INSERT INTO* nama\_tabel (kolom\_data) *VALUES* (isi\_data)

Kolom\_data adalah nama-nama kolom yang akan diisikan nilai datanya. Kolom data ini lebih baik dituliskan sesuai dengan data yang akan diisikan. Perhatikan jika akan mengisi data kolom yang bernilai *NOT Null* harus diisi pada saat perintah *INSERT* dilakukan. Kolom\_data dapat diabaikan untuk tidak ditulis jika data yang akan diisikan untuk seluruh tabel.

Berikut akan diisikan data untuk tabel Jurusan dan Mahasiswa sesuai dengan tabel berikut:

**Tabel 6-6 Data Jurusan**

<u>KodeJur</u>	NamaJur
12	Teknik Komputer
22	Komputer Akuntansi
32	Manajemen Informatika

**Tabel 6-7 Data Mahasiswa**

<u>NIM</u>	NAMA	Alamat	<u>KodeJur</u>
32121001	Aji	Jl. Abc 123	32
32121002	Bayu	Jl. Melati 10	32
12121001	Sonny	Jl. Ahmad Yani 121	12
12121002	Putra	Jl. Terate 15	12
22121001	Ali	Jl. Suka 12	22

Perintah perintah SQLnya sbb untuk menambahkan data pada tabel Jurusan:

```
INSERT INTO Jurusan (KodeJur,>NamaJur) VALUES ('12','Teknik
Infomatika');
INSERT INTO Jurusan (KodeJur,>NamaJur) VALUES ('22','Komputer
Akuntansi');
INSERT INTO Jurusan (KodeJur,>NamaJur) VALUES ('32','Manajemen
Informatika');
```

Atau dapat dituliskan perintah SQL sbb:

```
INSERT INTO Jurusan VALUES ('12','Teknik Infomatika');
INSERT INTO Jurusan VALUES ('22','Komputer Akuntansi');
INSERT INTO Jurusan VALUES ('32','Manajemen Informatika');
```

Perintah perintah SQLnya sbb untuk menambahkan data pada tabel mahasiswa:

```
INSERT INTO Mahasiswa (Nim,>Nama,>Alamat,>KodeJur)
VALUES ('32121001','Aji','Jl. Abc 123','32');
INSERT INTO Mahasiswa (Nim,>Nama,>Alamat,>KodeJur)
VALUES ('32121002','Bayu','Jl. Melati 10','32');
INSERT INTO Mahasiswa (Nim,>Nama,>Alamat,>KodeJur)
VALUES ('12121001','Sonny','Jl. Ahmad Yani 121','12');
INSERT INTO Mahasiswa (Nim,>Nama,>Alamat,>KodeJur)
VALUES ('12121002','Putra','Jl. Terate 15','12');
INSERT INTO Mahasiswa (Nim,>Nama,>Alamat,>KodeJur)
VALUES ('22121001','Ali','Jl. Suka 12','22');
```

### 6.4.2 Merubah Data

Struktur umum penulisan perintah untuk merubah data adalah sbb:

```
UPDATE nama_tabel SET nama_kolom = nilai_baru_kolom WHERE
kondisi
```

Perintah *update* ini akan mengubah isi kolom data yang jika tidak disikan kondisi *Where* data maka akan mengubah seluruh kolom yang ada di seuruh baris. Oleh sebab itu untuk membuat perubahan data harus selalu diperhatikan kondisi data yang akan dilakukan perubahan.

Misalnya akan dilakukan perubahan nama pada nama=Aji menjadi Aji Santoso. Maka kondisi yang harus dibuat adalah Nim sebagai *key* pada tabel tersebut, *Where* Nim='32121001'.

Perintah SQLnya untuk merubah data adalah:

```
UPDATE Mahasiswa SET Nama='Aji Santoso'  
WHERE Nim='32121001';
```

Berikutnya ingin dirubah nama bayu dengan nama Bayu Adji dan alamat menjadi Jl. Sukarame 15

```
UPDATE Mahasiswa  
SET Nama='Bayu Adji', Alamat='Jl. Sukarame 15'  
WHERE Nim='32121002';
```

### 6.4.3 Menghapus Data

Struktur umum penulisan perintah untuk menghapus data adalah sbb:

```
DELETE FROM nama_tabel WHERE kondisi
```

Perintah *delete* akan menghapus satu baris data atau lebih tergantung pada kondisi *Where* yang diberikan. Sebaiknya setiap melakukan proses penghapusan data maka kondisi penghapusan harus terdefinisi dengan baik, jika tidak maka akan kehilangan data.

```
DELETE FROM Mahasiswa WHERE Nim='12121002';
```

Perintah SQL *DELETE FROM* Mahasiswa akan mengkosongkan data pada tabel Mahasiswa

## 6.5 Mengakses Basisdata

Bagian atas sudah dijelaskan untuk melakukan *maintenance* data pada tabel. Berikutnya data dalam tabel tersebut akan dipanggil, diolah sehingga menjadi informasi yang berguna bagi pemakainya.

Untuk mengakses data digunakan perintah *SELECT*. Perintah ini akan mengembalikan kelompok baris-baris data satu baris atau lebih (mungkin juga tidak mengembalikan baris) yang berasal dari satu tabel, beberapa tabel, view dan *temporary* tabel yang ada dalam sebuah basisdata.

Untuk latihan select ini akan digunakan tabel Mahasiswa, Jurusan dan Kurikulum seperti berikut ini:

**Tabel 6-8 Data Jurusan**

<u>KodeJur</u>	NamaJur
12	Teknik Komputer

22	Komputer Akuntansi
32	Manajemen Informatika

**Tabel 6-9 Data Mahasiswa**

<u>NIM</u>	<u>NAMA</u>	<u>Alamat</u>	<u>KodeJur</u>
32121001	Aji	Jl. Abc 123	32
32121002	Bayu	Jl. Melati 10	32
12121001	Sonny	Jl. Ahmad Yani 121	12
12121002	Putra	Jl. Terate 15	12
22121001	Ali	Jl. Suka 12	22

Struktur dasar penulisan perintah select adalah sbb:

```
SELECT nama_kolom
FROM nama_tabel
[WHERE kondisi]
[ORDER BY mode_urutan_data]
```

Nama kolom merupakan nama-nama kolom data yang akan ditampilkan sebagai hasil dari perintah select ini. Jika diinginkan untuk ditampilkan seluruh kolom maka dapat digantikan dengan karakter bintang (“\*”)

Contoh akan ditampilkan data Jurusan, perintah SQL nya sbb:

```
SELECT KodeJur, NamaJur FROM Jurusan
```

Hasil yang didapat dari perintah diatas adalah:

**Tabel 6-10 Output Select Jurusan**

KodeJur	NamaJur
12	Teknik Infromatika
22	Komputer Akuntansi
32	Manajemen Informatika

```
SELECT * From Jurusan
```

➔ Hasilnya akan sama dengan tabel diatas.

### 6.5.1 Menganti Judul Kolom

Output pada tabel diatas memperlihatkan judul kolom sama dengan nama kolom, hal ini biasanya tidak baik saat judul kolom tersebut mengandung singkatan sehingga kurang informatif. Untuk mengganti judul

kolom dapat menggunakan perintah *AS*. Perintah ini akan memberikan nilai alias pada kolom\_data tersebut. Perintah SQL nya sbb:

```
SELECT KodeJur AS Kode, KodeJur AS Jurusan  
FROM Jurusan  
Sehingga hasil yang didapat adalah sbb:
```

**Tabel 6-11 Output Select Jurusan**

Kode	Jurusan
12	Teknik Informatika
22	Komputer Akuntansi
32	Manajemen Informatika

Penulisan alias pada kolom dapat juga dituliskan tanpa *keyword AD*, sehingga perintah SQL nya menjadi sbb:

```
SELECT KodeJur Kode, KodeJur AS Jurusan  
FROM Jurusan
```

### 6.5.2 Function pada SQL

Fungsi-fungsi pada SQL terbagi dalam beberap bagian atau kelompok,diantaranya :

1. Fungsi untuk agregasi
  - a. *AVG()* – mengembalikan nilai rata
  - b. *COUNT()* – mengembalikan jumlah data
  - c. *FIRST()* – mengembalikan nilai awal data
  - d. *LAST()* – mengembalikan nilai terakhir dari data
  - e. *MAX()* – mengembalikan nilai terbesar dari data
  - f. *MIN()* – mengembalikan nilai terkecil dari data
  - g. *SUM()* – mengembalikan hasil penjumlahan dari data
2. Fungsi untuk skalar
  - a. *UCASE()* – mengkonversi nilai kolom menjadi huruf besar semua
  - b. *LCASE()* - mengkonversi nilai kolom menjadi huruf kecil semua
  - c. *MID()* – mengambil sebagian dari suatu string
  - d. *LEN()* – mengembalikan panjang string
  - e. *ROUND()* – membulatkan nilai bilangan pada nilai terdekat

Fungsi-fungsi ini mungkin akan berbeda dari suatu dialek SQL dengan dialek SQL lainnya.

Contoh penggunaan function SQL sbb:

SELECT COUNT (Nim) FROM Mahasiswa

SELECT MAX(KodeJur) FROM Jurusan

### 6.5.3 Menentukan Kondisi

Dalam perintah *Select* terdapat *klausa Where* yang digunakan untuk menentukan hasil sesuai dengan data yang dibutuhkan. Kondisi ini perlu dihindangi tidak semua data akan ditampilkan.

Struktur penulisan *where* pada *select* sbb:

```
SELECT nama_kolom
FROM nama_tabel
[WHERE kondisi]
```

Kondisi = nama\_kolom operator nilai\_data

*Operator* yang dapat digunakan pada penulisan kondisi adalah sbb:

**Tabel 6-12 Operator Kondisi**

Operator	Keterangan
=	Sama dengan
<>	Tidak sama dengan
>	Lebih besar dari
<	Lebih kecil dari
>=	Lebih besar sama dengan dari
<=	Kebih kecil sama dengan dari
BETWEEN	Nilai diantara
LIKE	Mencocokkan sesuai dengan pola data yang diberikan
IN	Mencocokkan data dengan yang ada

Selain *operator* diatas juga digunakan operator relasi seperti *AND*, *OR* dan *NOT*

Berikut beberapa contoh penulisan kondisi:

WHERE Nim=' 32121001'

WHERE Sks=2

```
WHERE Nama ='Aji'
```

Untuk kolom yang jenis datanya adalah karakter (*CHAR*, *VARCHAR*) nilai datanya harus diapit oleh tanda kutip seperti dibagian atas.

Berikut perintah lengkap menggunakan kondisi dalam menampilkan data.

```
SELECT Nim, Nama, Alamat
FROM Mahasiswa
WHERE Nim='32121001'
```

Hasil yang didapat adalah sbb:

**Tabel 6-13 Output Select Mahasiswa menggunakan Where**

NIM	NAMA	Alamat
32121001	Aji	Jl. Abc 123

```
SELECT Nim, Nama
FROM Mahasiswa
WHERE KodeJur='32'
```

Hasil yang didapat adalah sbb:

**Tabel 6-14 Output Select Mahasiswa menggunakan Where**

NIM	NAMA
32121001	Aji
32121002	Bayu

Operator *Like* digunakan untuk mencari data karakter/string dengan pola-pola tertentu. Pembuatan pola diwakili dengan karakter persen ('%') untuk mengartikan satu atau lebih karakter pada string dan tanda garis bawah ('\_') yang mewaliki satu karakter saja.

Berikut contoh penggunaan *Like*:

**Tabel 6-15 Contoh Perintah Like**

Perintah Like	Arti
WHERE Nama LIKE 'A%'	Nama yang berawalan A
WHERE Nama LIKE '%i'	Nama yang berakhiran i



WHERE Nama LIKE 'AL_'	Nama yang mempunyai awala AL dan huruf ketiganya bebas
-----------------------	--

#### 6.5.4 Menguruntukan Data

Klausa *ORDER BY* digunakan untuk menguruntukan data output. Urutan data ini dapat diset baik secara *Ascending* (urutan dari kecil ke besar) atau *Descending* (urutan dari besar ke kecil)

Struktur penulisan *ORDER BY* sebagai berikut:

```
SELECT nama_kolom
FROM nama_Tabel
ORDER BY nama_kolom ASC | DESC
```

Nilai *Default* untuk pengurutan ini adalah *ASC*, sehingga untuk menuruntukan dari kecil ke besar sebenarnya tidak perlu dituliskan *ASC* nya. Berikut contoh perintah SQL menggunakan *ORDER BY*

```
SELECT Nim, Nama
FROM Mahasiswa
ORDER BY Nama
```

**Tabel 6-16 Output Order By Tabel Mahasiswa**

NIM	NAMA
32121001	Aji
22121001	Ali
32121002	Bayu
12121001	Sonny

```
SELECT Nim, Nama
FROM Mahasiswa
ORDER BY Nama DESC
```

**Tabel 6-17 Output Order By DESC Tabel Mahasiswa**

NIM	NAMA
12121001	Sonny
32121002	Bayu
22121001	Ali
32121001	Aji

```
SELECT KodeJur As Jurusan,Nim, Nama
```

FROM Mahasiswa  
ORDER BY Jurusan, Nama

**Tabel 6-18 Output Order By Dua Kolom**

Jurusan	Nim	Nama
12	12121001	Sonny
22	22121001	Ali
32	32121001	Aji Santoso
32	32121002	Bayu Adji



## Rangkuman

1. *Structure Query Language* (SQL) hadir dalam RDBMS sebagai bahasa untuk mengakses, *me-maintenance* kumpulan dari *table*-tabel data yang dihubungkan oleh masing-masing *key*.
2. SQL muncul pertama kali diawal tahun 1970-an sebagai sebuah hasil penelitian pada lab IBM, San Jose, California yang dilakukan oleh Donald C Meserly dan Raymond F Boyce.
3. *American National Standard Institute* (ANSI) dan *International Standard Organization* (ISO) membentuk standar SQL yang dimulai tahun 1986.
4. Beberapa standar SQL yang sudah dibuat adalah: SQL86, SQL92, SQL99, SQL:2003, SQL2006 dan SQL:2008
5. Secara umum perintah-perintah SQL dibagi dalam tiga bagian utama yaitu:
  - a. *Data Definition Language* (DDL)
  - b. *Data Manipulation Language* (DML)
  - c. *Data Control Language* (DCL)
6. DML menyediakan perintah-perintah untuk melakukan proses manipulasi pada konten data yang ada dalam basidata. Perintah-perintah ini diantaranya: *Select*, *Insert*, *Update* dan *Delete*.
7. DDL menyediakan perintah-perintah untuk mengakses objek-objek basidata seperti *database*, *table*, *index* PL/Transact baik untuk membuat dengan perintah *Create*, merubah dengan perintah *Alter* dan menghapus dengan perintah *Drop*.
8. DCL menyediakan perintah-perintah untuk mengatur hak akses terhadap objek-objek data dan konten data untuk masing-masing pengguna dan *layer user* dalam basidata.

## Daftar Pustaka

1. Raghu Ramakrishnan / Johannes Gehrke “*Database Management System*” Second edition.
2. Silberschatz-Korth-Sudarshan: *Database Sistem Concepts*, Fourth Edition.2001
3. Handbook Telkom Polytechnic : *Perancangan Basis Data*
4. Oracle *Database 10g: Administration Workshop I*, volume I: Student Guide