**TUGAS MENYUSUN BAB TEORI SQL QUERY**

****

**Disusun oleh**

**HILMAN ABU DZARRIN**

**2110147040**

**BASIS DATA D4LJPJJ**

**POLITEKNIK ELEKTRONIKA NEGERI SURABAYA**

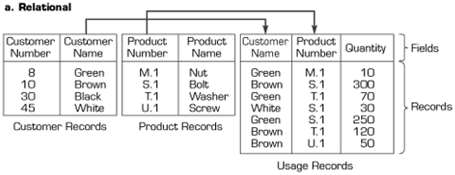
**2015**

**BAB I**

## 1.1 Model Basis Data

* *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.

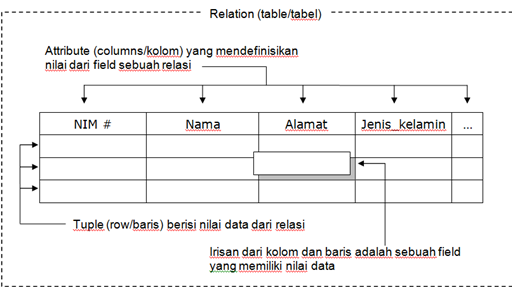


Basis data relasional memiliki satu struktur “lojik” yang disebut *Relation* (relasi). Struktur relasi merupakan strukture data 2-dimensi dan pada level “fisik” berupa *table* (tabel). Attribute merepresentasikan elemen dari data yang berkaitan dengan relasi. Sebagai contoh, relasi Mahasiswa memiliki atribut-atribut seperti nim, nama, tempat lahir. tanggal\_ lahir, dan alamat.

Pada buku-buku teks mengenai perancangan basis data, sebuah relasi dinotasikan secara konvensional dengan *Relation*(attribute1, attribute2, ..) dengan aturan : nama dari relasi dan atribut-atribut dari relasi yang berada didalam tanda kurung.

**Contohnya:** Mahasiswa(NIM, Nama, Alamat, Jenis\_kelamin …).

Nilai data dari atribut dari sebuah relasi akan disimpan dalam tuple atau *row* (baris) dari tabel. Gambar dibawah merupakan ilustrasi dari sebuah relasi.



## 

## 1.2 Entitas dan Himpunan Entitas

Definisi entitas adalah objek yang dirasa penting di sistem tersebut, yg bisa berupa :

* + **Objek Konkrit**

Contoh : Orang, Buku

* + **Objek Abstrak**

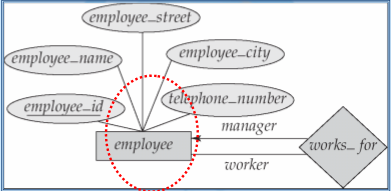
Contoh : Jadwal, Pinjaman, Tabungan

Bambang adalah salah satu contoh dari entitas. Sedangkan bambang, susi, sumarno 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 mahasiwa. Entitas menunjuk kepada pada individu suatu objek sedangkan himpunan entitas menunjuk pada rumpun (family) dari individu tersebut.

## 1.3 Derajat Himpunan Relasi

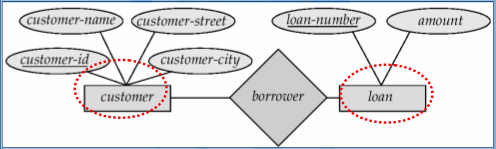
Jika dilihat dari **jumlah entitas** yang dihubungkan oleh sebuah relasi, maka kita bisa membagi menjadi 3 macam:

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



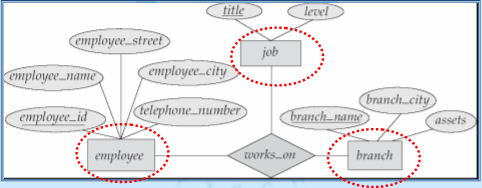
Relasi di atas menggambarkan entitas karyawan yang ber-relasi dengan entitas karyawan. Entitas karyawan bisa merupakan karyawan biasa tetapi bisa juga merupakan manajer. Relasi yang terjadi yaitu relasi karyawan bekerja untuk manajer (\* entitas manajer adalah salah satu karyawan juga). Perhatikan kardinalitas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

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



Relasi di atas menggambarkan entitas pelangan 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)



Relasi di atas menggambarkan entitas karyawan yang ber-relasi dengan entitas cabang dan entitas pekerjaan melalui relasi bekerja\_di. 1 karyawan bekerja di sebuah id pekerjaan tertentu dan juga bekerja di sebuah cabang tertentu. Ada 3 entitas yang terlibat dari relasi di atas

## 1.4 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 satiap 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***

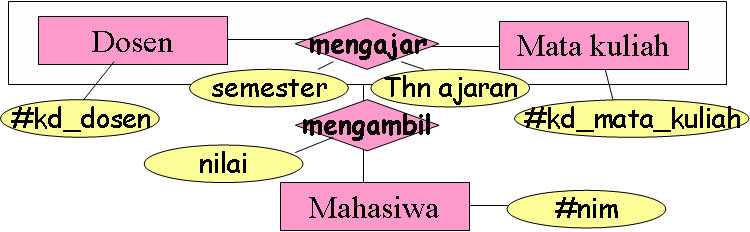
*Primary key* merupakan salah satu dari *candidate key* yang terpilih. Alasan pemilihan *primary key* :

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

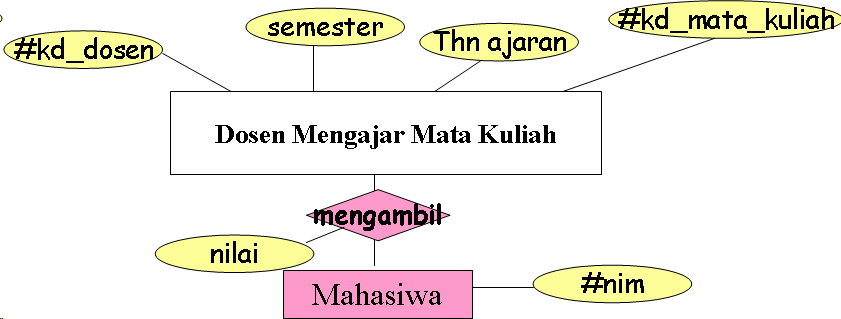
Contoh dari *primary key* adalah Nim

## 1.5 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.



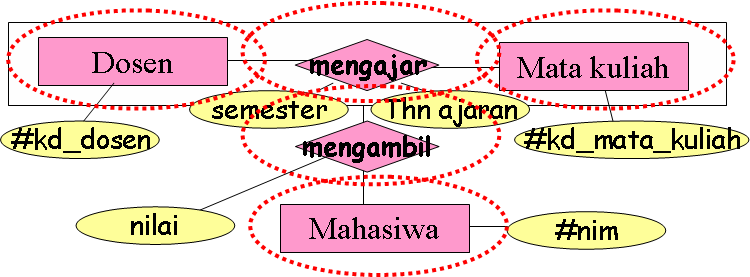
Gambar di atas menunjukkan relasi dosen mengajar sebuah mata kuliah dan mahasiswa mengambil mata kuliah yang diajarkan oleh dosen tertentu. Agregasi di perlukan dikarenakan tidak di mungkinkan mahasiwa untuk mengambil mata kuliah tanpa adanya dosen yang bersedia untuk mengajar mata kuliah tersebut. Dalam 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 mahasiwa 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.

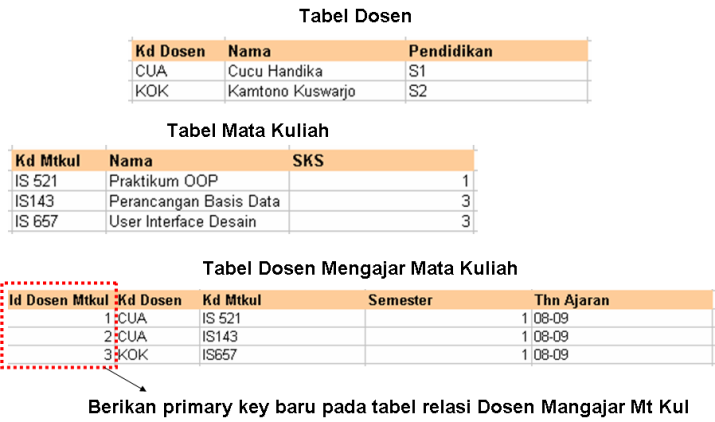


## 1.5.1 Representasi Agregasi

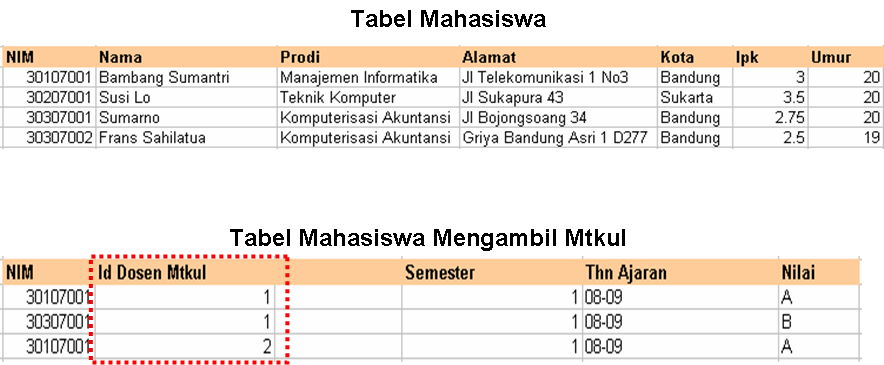
Untuk merepresentasikan agregasi, buatlah tabel yang terdiri dari :

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





Representasi Agregasi untk tabel mata kuliah, dosen dan Dosen mengajar Mtkul



Representasi Agregasi untuk tabel Mahasiwa dan Mahasiwa Mengambil Mtkul

**1.6 Operasi aritmatika**

**1.6.1 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,

**σ 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

**σ tempatlahir= “Jakarta” (mahasiswa)**

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

**1.6.2 Operasi Project**

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

**π <daftar atribut > (<nama tabel>)** atau **π 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,

**π <NIM,Nama\_mahasiswa > (mahasiswa)**

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

**1.6.3 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 X 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:

**σindeks= “A” (t\_nilai x t\_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\_kuliah dan t\_nilai (tabel kuliah dan tabel nilai) sama-sama memiliki *field* 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 IS\_001. 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:

**π <NIM > (σt\_kuliah.kode\_mk=t\_nilai.kode\_mk п kode\_mk = “IS\_001” (t\_nilai xt\_ 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.

**1.6.4 Operasi** **Unio**

Operasi *Unio* adalah operasi yang menggabungkan semua baris dari dua buah tabel dan kedua tabel tersebut harus sesuai atau memiliki hasil projeksi yang sama. Dimana akan mengasilkan 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

**Πtempat\_lahir(t\_dosen)**

**dan**

**Πtempat\_lahir(t\_mahasiswa)**

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

**Πtempat\_lahir(t\_dosen) U Πtempat\_lahir(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.

**1.6.5 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*.

**1.6.6 Opera****i Intersection**

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

**T1 ∩ T2**

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

**T1 – (T1-T2)**

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

**1.6.7 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* (**ρ**) 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, khusnya yang melibatkan dua/lebih *sum*ber data yang sama.

Simbol untuk operasi *rename* adalah sebagai berikut:

****

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

## 1.7 Function pada SQL

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

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

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

Contoh penggunaan function SQL sbb:

SELECT *COUNT* (Nim) FROM Mahasiswa

SELECT MAX(KodeJur) FROM Jurusan

## 1.8 Join

Merupakan operasi yang digunakan untuk mengabungkan informasi dari dua atau lebih relasi (Join ). 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 pembandinganyang digunakan dalam operasi *join* yaitu {=, <, ≤, >, ≥, ≠}

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 Equi*JOIN* menghubungkan tabel didasarkan pada kondisi yang sama dengan memban-dingkan kolom tertentu setiap tabel. Hasil equi*JOIN* tidak menghilangkan kolom dupli-kat dan kondisi atau kriteria penggabungan tabel harus terdefinisi secara eksplisit.

Theta *JOIN* adalah equi*JOIN* yang membanding-kan kolom tertentu setiap tabel menggunakan operator pembanding 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 ditabel/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*. Sedangakan operasi *full* *outer* *join* meriupakan 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.

## 1.9 Nama Alias

Nama alias dapat diberikan untuk nama kolom dan juga untuk nama tabel. Pemberian nama alias menggunakan *keyword* *AS*. Nama alias ini sangat diperlukan untuk menyingkat penulisan-penulisan perintah *query* yang komplek dan banyak.

Struktur penulisan nama tabel adalah sbb:

SELECT nama\_kolom

FROM nama\_tabel AS nama\_alias

Contoh perintah SQL untuk penggunaan alias ini adalah sbb:

SELECT Nim, Nama, NamaJur

FROM Mahasiswa AS M, Jurusan AS J

WHERE M.kodeJur = J.KodeJur

Contoh lain sbb:

SELECT Nim, Nama, M.kodeJur, NamaJur

FROM Mahasiswa AS M, Jurusan AS J

WHERE M.kodeJur = J.KodeJur

Perintah INNER *JOIN* juga dapat menggunakan alias sbb:

SELECT Nim, Nama, NamaJur

FROM Mahasiswa AS m

INNER *JOIN* Jurusan AS j

ON m.kodeJur = j.KodeJur

Pembuatan alias selain menggunakan *keyword* *AS* juga dapat menggunakan spasi sebagai pemisah dari nama tabel dan nama alias. Contoh sbb:

SELECT Nim, Nama, NamaJur

FROM Mahasiswa m

INNER *JOIN* Jurusan j

ON m.kodeJur = j.KodeJur

## 1.10 Subquery

Sub*query* adalah sebuah bentuk perintah *SELECT* yang mengembalikan nilai yang ada kepada perintah lain berupa perintah *SELECT*, *INSERT*, *UPDATE* dan *DELETE* atau dengan kata lain sub*query* adalah *query* dalam *query*.

*Subquery* merupakan alternative dalam membuat perintah SQL yang menggunakan *JOIN*, dimana hal ini dibuat untuk meningkatkan *performance* terhadap perintah *query* tersebut. namun demikian, tergantung kepada data jika kalau ingin dibandingkan mana yang lebih baik antara *subquery* dengan *JOIN* ini.

*Subquery* dapat ditempatkan pada perintah *SELECT* setelah klausa *Select*, *From*, *Where*, *Group* *BY* dan *Having*. Umumnya perintah sub *Query* ini mengikuti format sebagai berikut:

* *WHERE* *ekspresi* [*NOT*] *IN* **(***subquery***)**
* *WHERE* *ekspresi operator perbandingan* [*ANY* | *ALL*] **(***subquery***)**
* *WHERE* [*NOT*] *EXISTS* **(***subquery***)**

**Operator IN**

*Subquery* dengan menggunakan *operator* *IN* adalah akan me-*list* hasil dari *subquery* untuk dibandingkan dengan ekspresi *where* yang diberikan. *Subquery* akan dijalankan terlebih dahulu baru kemudian *query* pemanggilnya akan dijalankan.

Berikut contoh perintah *Subquery* untuk menampilkan nama penerbit untuk buku yang berjenis *business* dengan menggunakan *operator* *IN*

SELECT NamaPen

FROM Penerbit

WHERE kodePen IN

(SELECT kodePen

FROM Buku

WHERE jenis='*business*')

Jika operator *IN* ditambahkan operator *NOT* maka akan menjadi NOT *IN*. hal ini akan memberikan nilai kebalikan dari hasil yang didapat dengan menggunakan perintah *IN*

SELECT NamaPen

FROM Penerbit

WHERE kodePen NOT IN

(SELECT kodePen

FROM Buku

WHERE jenis='*business*')

**Operator *EXISTS***

Operator *Exists* ini akan melakukan pemeriksaan terhadap hasil subqery apakah menghasilkan baris-data atau tidak, jika *subquery* tersebut menghasilkan baris data maka akan mengembalikan nilai true dan sebaliknya jika tidak menghasikan data maka akan mengembalikan nilai false

Berikut contoh perintah Sub*query* untuk menampilkan nama penerbit untuk buku yang berjenis *business* dengan menggunakan operator *EXISTS*

SELECT NamaPen

FROM Penerbit p

WHERE *EXISTS*

(SELECT \*

FROM Buku b

WHERE b.KodePen= p.KodePen

AND jenis='*business*')

Jika operator *EXISTS* ditambahkan operator *NOT* maka akan menjadi *NOT* *EXISTS*. hal ini akan memberikan nilai kebalikan dari hasil yang didapat dengan menggunakan perintah *EXISTS*

SELECT NamaPen

FROM Penerbit p

WHERE NOT *EXISTS*

(SELECT \*

FROM Buku b

WHERE b.KodePen= p.KodePen

AND jenis='*business*')

**Operator Komporasi**

*Subquery* memungkinkan juga menggunakan operator komparasi yang terdiri dari =, < >, >, > =, <, ! >, ! <, or < =. Berbeda dengan operator sebelum penggunaaan operator ini sub*query* harus menghasilkan data tunggal yang menjadi parameter untuk *query* pemanggilnya.

Berikut contoh perintah *Subquery* untuk menampilkan nama penerbit untuk buku yang berjenis *business* dengan menggunakan operator komparasi =

SELECT NamaPeng

FROM Pengarang

WHERE KotaTinggal =

(SELECT Kota

FROM Penerbit

WHERE namaPen='Algodata Infosistems')

**Operator ANY dan ALL**

Penggunaan operator ini dipadukan dengan operator relasi sehingga nanti akan terbentuk >*ANY*, =*ANY* , <> *ANY*, >*ALL* dan <>*ALL*

>ANY mengadung arti bahwa akan mengambil nilai lebih besarnya dari nilai terendah yang ada dalam list tersebut. contoh >*ANY* (1,2,3) maka nilai lebih besarnya adalah 1

>*ALL* mengandung arti bahwa akan mengambil nilai lebih besarnya dari nilai tertinggi yang ada dalam lsit tersebut. Contoh >*ALL* (1,2,3) maka nilai lebih besarnya adalah 3

=*ANY* pada dasarnya sama dengan fungsi operator *IN* yaitu akan menyamakan apa yang ada dala list sebagai hasil dari sub*query*nya. <>*ANY* tidak serta merta sama dengan *NOT* *IN*, hal ini berbeda karena <>*ANY* menghasilkan not = a or not = b , sementara *NOT* *IN* menghasilkan not = a and not = b. <>*ALL* pengertianya sama dengan *NOT* *IN*

Berikut contoh perintah Sub*query* untuk menampilkan nama penerbit untuk buku yang berjenis *business* dengan menggunakan operator <>*ANY*

SELECT NamaPeng

FROM Pengarang

WHERE KotaTinggal <>ANY

(SELECT Kota

FROM Penerbit)

## 1.11 Bahasa Basis Data

1. ***Data Definition Language* (DDL)**

*Data Definition Language* ini berfungsi untuk menspesifikasikan skema basis data. Degan bahasa ini *user* dimungkinkan untuk membuat tabel baru, indeks, mengubah struktur tabel, menetukan stuktur penyimpanan tabel, dan masih banyak lagi. Hasil dari kompilasi DDL adala kumpulan tabel yang tersimpan pada sebuah *file* khusus yang disebut dengan kamus data (*data dictionary*) atau data *directory*. Kamus data merupakan sebuah *file* yang berupa metadata, yaitu data tentang data. Kamus data ini akan selalu diakses pada suatu operasi basis data sebelum suatu *file* data yang sesungguhnya diakses.

1. ***Interactive Data Manipulation Language* (DML)**

Level abstraksi yang telah dibahas sebelumnya tidak hanya berlaku pada definisi atau struktur data tetapi juga pada manipulasi data. Manipulasi data itu sendiri dapat berupa:

* + 1. pengambilan informasi yang tersimpan pada basis data.
    2. Pemasukan informasi baru ke dalam basis data.
    3. Penghapusan informasi dari basis data.
    4. Modifikasi informasi yang tersimpan pada basis data

Pada level fisik, kita harus mendefinisikan algoritma yang memungkinkan pengaksesan yang efisien terhadap data. Pada level yang lebih tinggi, yang dipentingkan bukan hanya efisiensi akses tetapi juga efisiensi interaksi *user* dengan sistem.

DML merupakan bahasa yang memungkinkan *user* untuk mengakses atau memanipulasi data sebagaimana telah direpresentasikan oleh model data. Terdapat dua macam DML, yaitu:

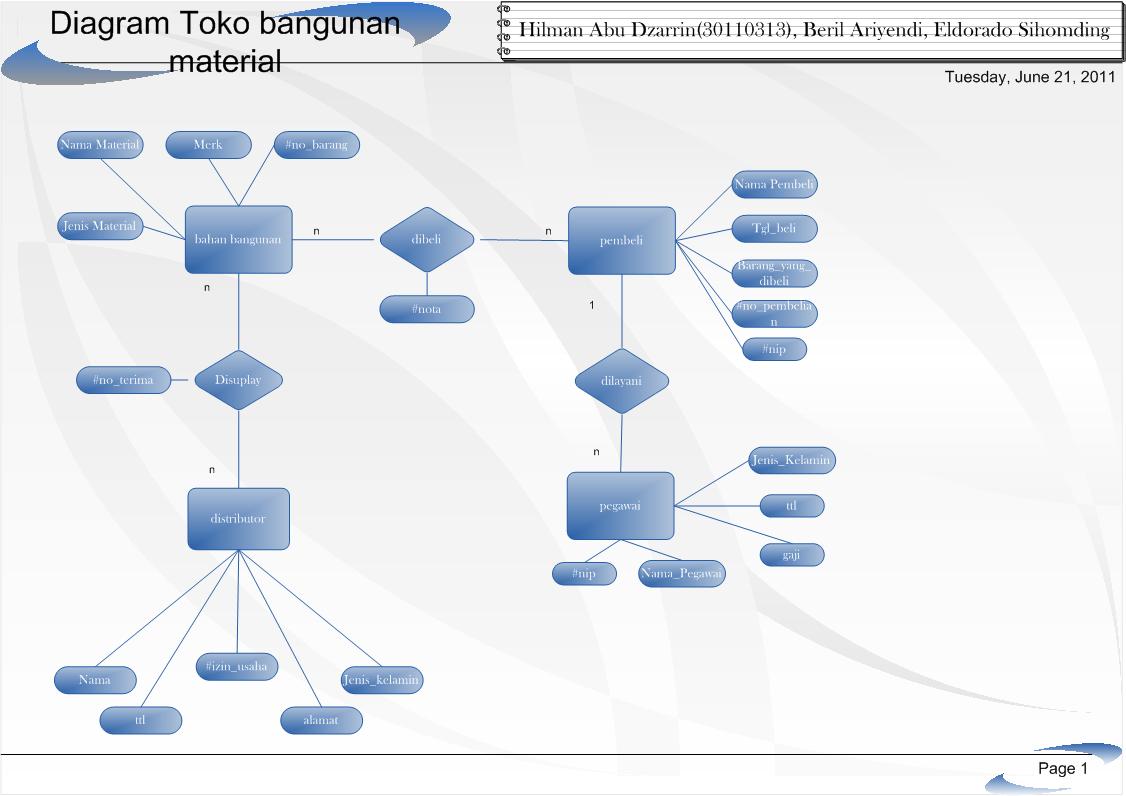
* Prosedural, mengharuskan *user* untuk menentukan data apa yang dibutuhkan dan bagaimana untuk mendapatkan data tersebut.
* Nonprosedural, mengharuskan pemakai untuk menentukan data apa yang dibutuhkan tanpa menyebuntukan bagaimana mendapatkan data tersebut.

**BAB II**

### STUDI KASUS:

**“TOKO BAHAN BANGUNAN MATERIAL”**

# Diagram ER



# Skema Relasi

-distributor :(#izin\_usaha, Nama, ttl, alamat, Jenis\_kelamin)

-pegawai :(#nip, Nama\_pegawai, Jenis\_kelamin, gaji, alamat, ttl)

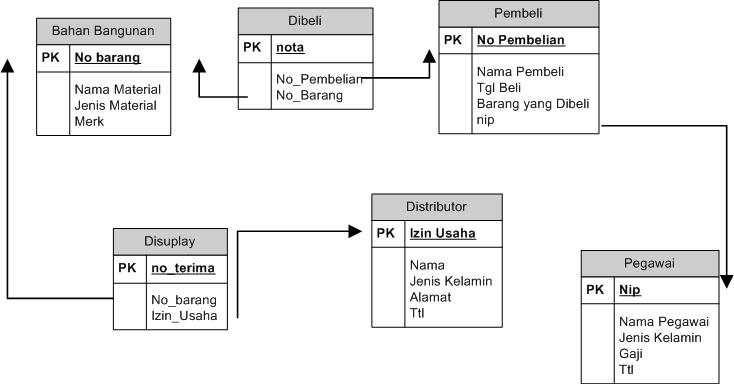
-pembeli :(#no\_pembelian, @nip, Nama\_pembeli, Barang\_yang\_dibeli, Tgl\_beli)

-bahan bangunan :(#no\_barang,@ no\_pembelian, Nama\_material, Jenis\_material, Merk)

-disuplay :( @izin\_usaha, @no\_barang)

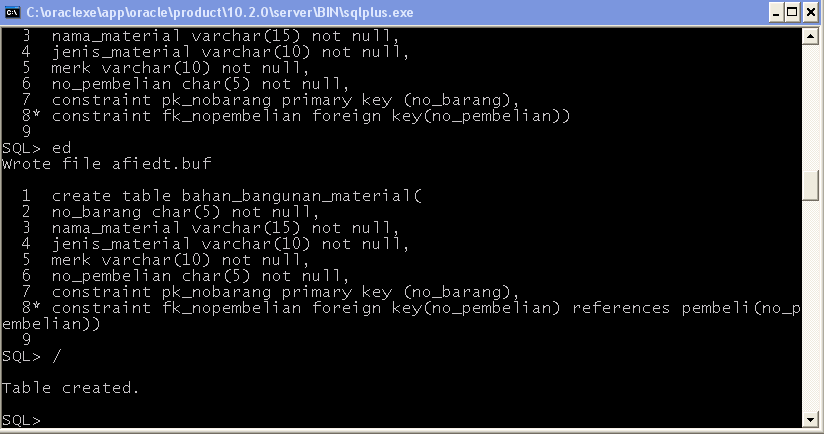
-dibeli :(@no\_pembelian,@ no\_barang)

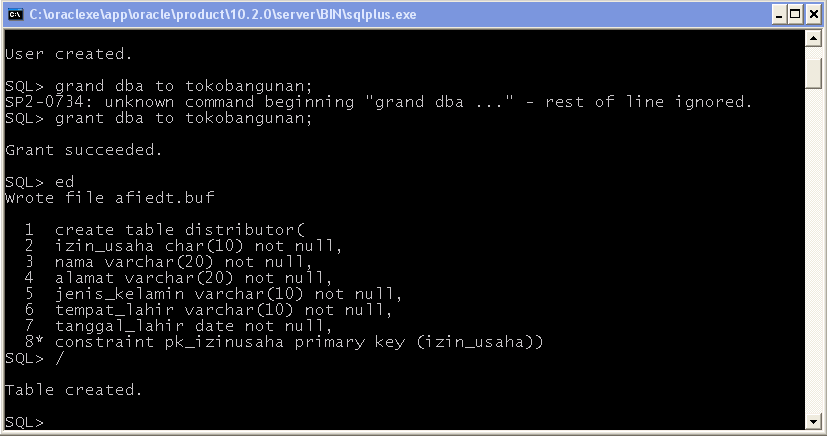
# Diagram Relationship

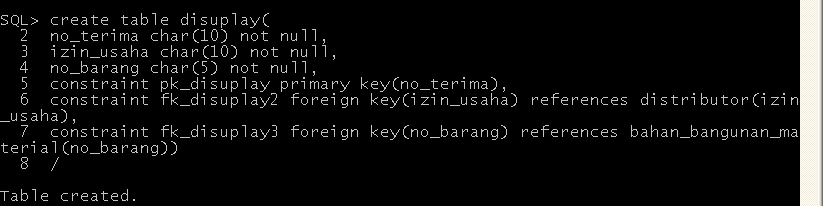


# Sintaks Create Table (semua tabel)

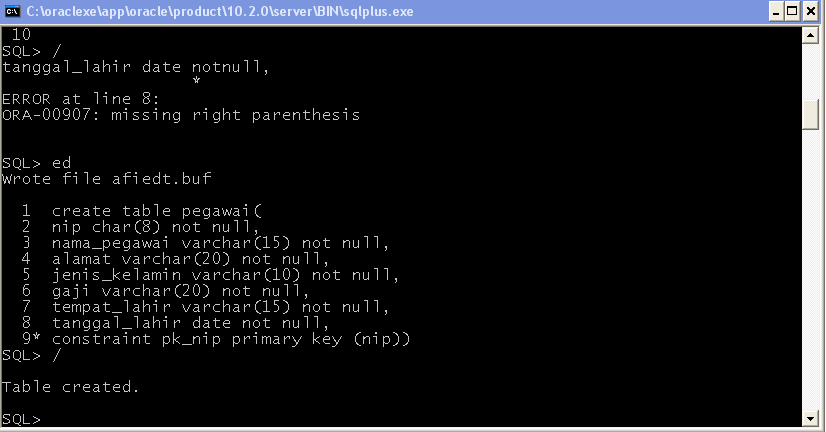
* **Table bahan\_bangunan**



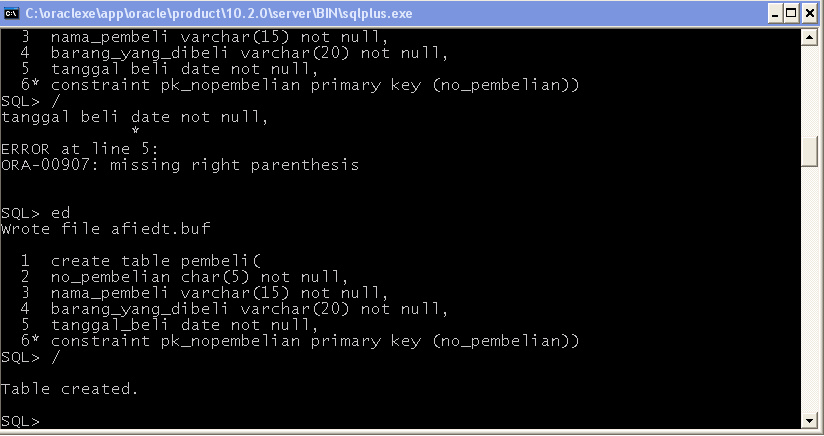
* **Table distributor**
* **Table disuplay**



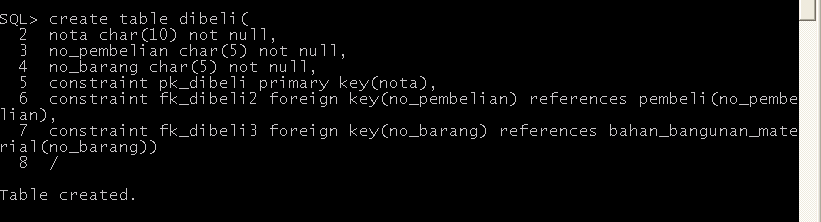
* **Table Pegawai**



* **Table Pembeli**

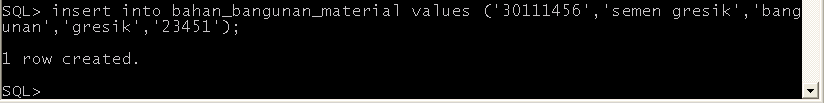


* **Table dibeli**

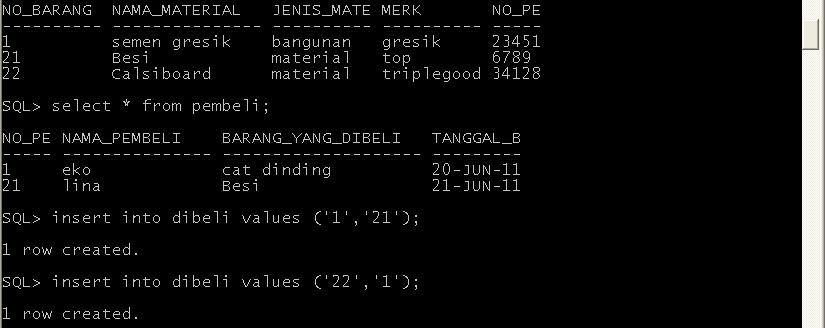


# Contoh Sintaks Insert Data

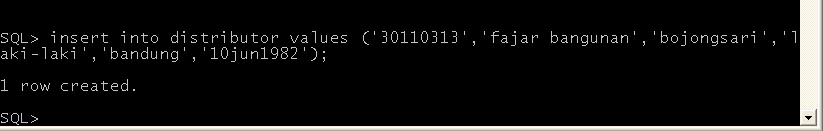
* **Insert bahan\_bangunan**



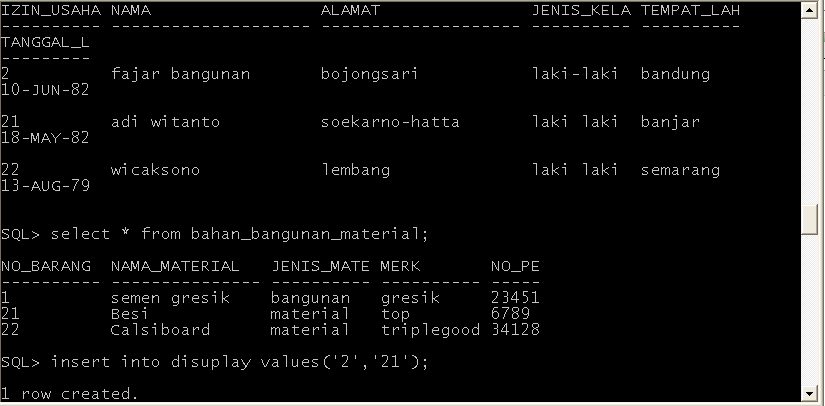
* **Insert dibeli**



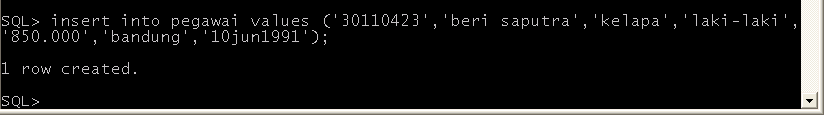
* **Insert distributor**



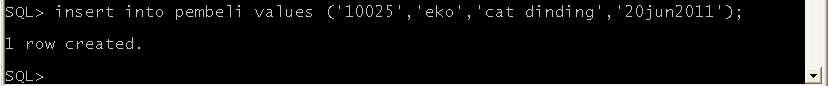
* **Insert disuplay**



* **Insert pegawai**

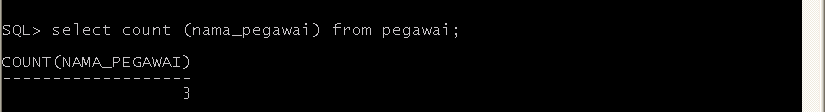


* **Insert pembeli**



# Contoh Query

## Query 1 (tanpa join dengan agregasi)



## Query 2 (dengan join 2 tabel)

## Query 2 .PNG

## Query 3 (dengan join 3 tabel atau lebih)

## Query 3 .PNG

## Query 4 (dengan klausa Group By, boleh join dan boleh tidak)

## query 4 tabel distributor baru.PNG

## Query 5 (dengan klausa Group By dan Having)

