Modul Praktikum

STIKOM SURABAYA

Structured Query Language

MAHASISWA MAMPU MENGAMBIL, MEMANIPULASI, DAN MENDEFINISIKAN DATA PADA SUATU DATABASE DENGAN MENGGUNAKAN PERINTAH SQL

ST-RK-1.16-082-007/R-
LABORATORIUM KOMPUTER
132

LABKOM
laboratorium komputer

始まるの話「Forewords」

「自分で受けった心得は消して自分を裏切らない」
“The knowledge you got by yourself will never betray you”

I want 75% of you people to pass this practice, or we can make it more or less in number, that’s my one and only target in this class. So what about you? Will you increase or decrease my target?

I don’t know nor i care whether you like this practice or not, but one thing for sure, if you don’t pass this practice, your step and time to move forward will just be stalled by this practice, so what’s your choice? Learn to pass? Or just leave it be?

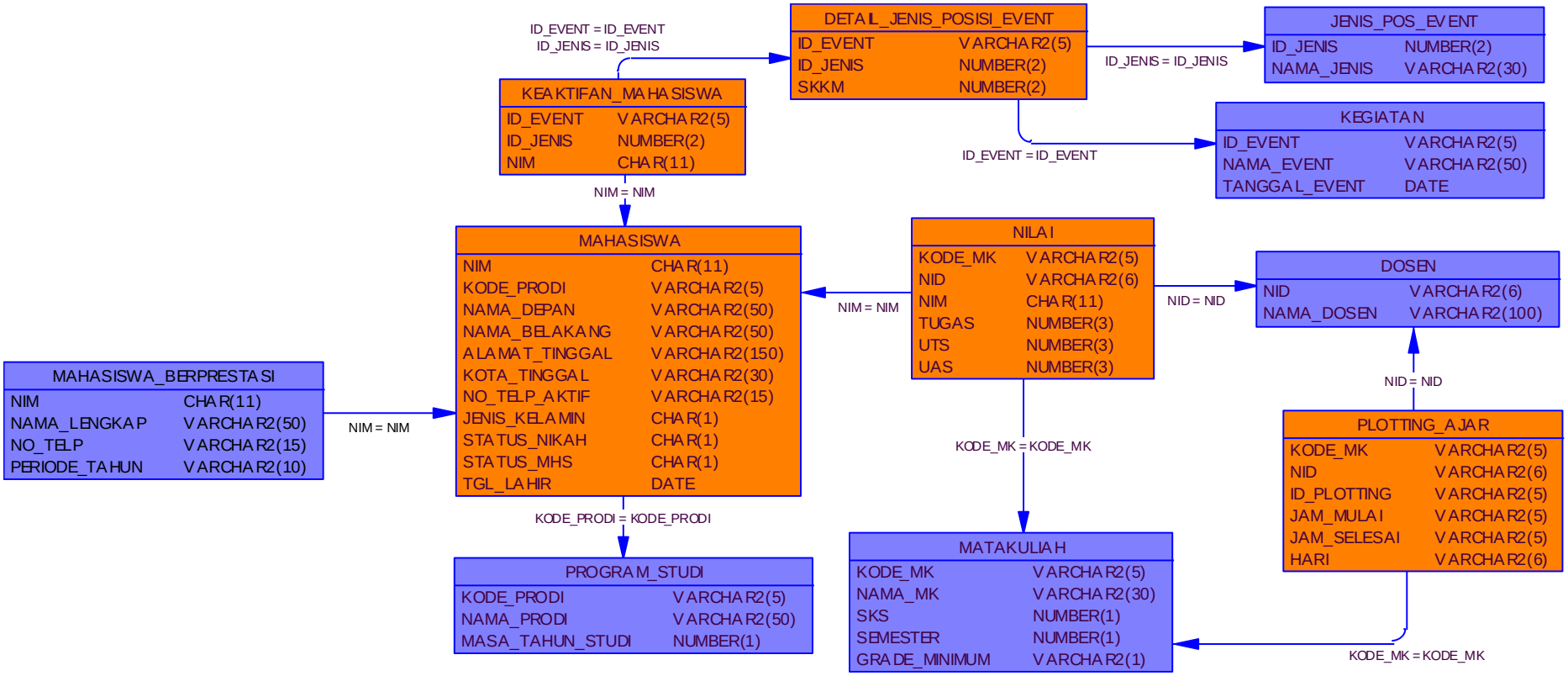
“**The knowledge you got by yourself will never betray you”**, and more to add, **it’ll never harm you**. So can you pass this practice?

頑張るならできる！
“You Can Do It!”

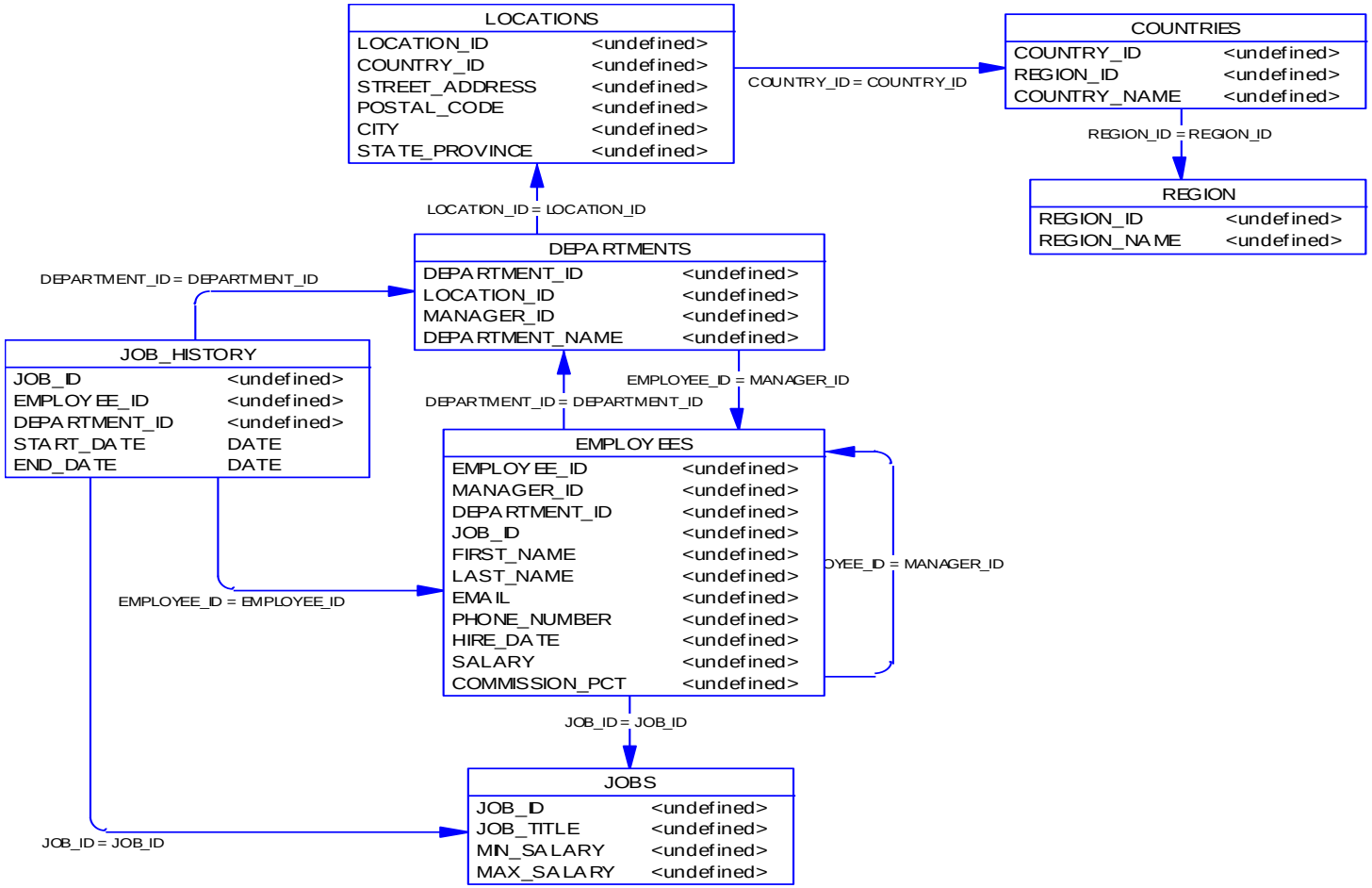
人

Index Of Content

Tabel untuk Latihan



Tabel untuk Tugas



Stage 1

Retrieve, Filter, and Sort

"***A simple step forward starts everything***"

人

Tujuan Hari Ini

Bagaimana melakukan pengambilan,
penyaringan, dan pengurutan data secara sederhana

What We Study

Perintah `SELECT` Sederhana
Membatasi Hasil *query*
Mengurutkan Data

1 Perintah `SELECT` Sederhana

1.1 Perintah `SELECT`

Perintah `SELECT` digunakan untuk menampilkan atau mengambil data dari database. Perintah `SELECT` memiliki 3 (tiga) macam kemampuan:

a. *Projection*: Perintah `SELECT` bisa digunakan untuk memilih kolom apa saja dari tabel yang akan ditampilkan.

b. *Selection*: Perintah `SELECT` bisa digunakan untuk memilih baris data mana saja yang akan ditampilkan.

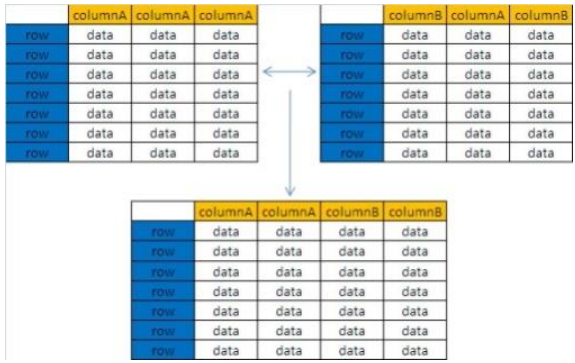
c. *Joining*: Perintah `SELECT` bisa digunakan untuk menampilkan data atau informasi dari 2 atau lebih tabel yang terhubung.

	column	column	column
row	data	data	data
row	data	data	data
row	data	data	data
row	data	data	data
row	data	data	data
row	data	data	data
row	data	data	data

Projection

	column	column	column
row	data	data	data
row	data	data	data
row	data	data	data
row	data	data	data
row	data	data	data
row	data	data	data
row	data	data	data

Selection



*Joining**

*Untuk *Joining*, akan dijelaskan pada *stage* selanjutnya.

Perintah `SELECT` sederhana terdiri atas :

```
SELECT      */[{DISTINCT}      COLUMN/EXPRESSION
{ALIAS}]  FROM [TABLE];
```

Penjelasan:

SELECT: daftar satu atau lebih kolom.

*: memilih semua kolom

DISTINCT: menghilangkan duplikat data.

Column/Expression: memilih kolom yang disebutkan atau ekspresi

Alias: memberikan *header* yang berbeda kepada kolom yang dipilih.

From [table]: menyebutkan tabel dari kolom yang dipilih.

Example:

1. Menampilkan semua kolom dari tabel *Mahasiswa*.

```
SELECT * FROM MAHASISWA;
```

2. Menampilkan hanya nim, nama_depan, dan nama_belakang dari mahasiswa.

```
SELECT NIM, NAMA_DEPAN, NAMA_BELAKANG FROM
MAHASISWA;
```

3. Menampilkan tanggal hari ini.

```
SELECT SYSDATE FROM DUAL;
```

1. 2 *Arithmetic Operations*

Terkadang, anda perlu menggunakan kalkulasi ketika membuat sebuah laporan, atau anda perlu mengubah bagaimana sebuah data perlu ditampilkan.

Penghitungan ini bisa dilakukan di SQL dengan menggunakan operator aritmatik.

Operator	Note
+	Used for addition of number/date data type
-	Used for reduction of number/date data type
*	Used for multiplication of number data type
/	Used for division of number/date data type

Example:

1. Operasi 2x200.

```
SELECT 2*200 FROM DUAL;
```

2. Menampilkan tanggal hari ini dan tanggal 5 hari dari sekarang.

```
SELECT SYSDATE, SYSDATE + 5 FROM DUAL;
```

3. Menampilkan nim, kode_mk, dan nilai akhir dari seorang mahasiswa

```
SELECT NIM, KODE_MK, (TUGAS*0.4) + (UTS*0.3)
+ (UAS*03) FROM NILAI;
```

1.3 *Concate Operators*

operator CONCATE digunakan untuk menggabungkan text, nilai kolom, atau karakter. Anda bisa menghubungkan sebuah kolom dengan kolom lain, ekspresi aritmatik, atau nilai konstan, untuk membuat sebuah ekspresi karakter dengan menggunakan operator CONCATE yang disimbolkan dengan '||'.

Example:

1. Menampilkan nama lengkap dari mahasiswa

```
SELECT 'Nama saya '|| NAMA_DEPAN ||' '||
NAMA_BELAKANG FROM MAHASISWA;
```

2. Menampilkan nama lengkap dan alamat dari mahasiswa.

```
SELECT NAMA_DEPAN ||' '|| NAMA_BELAKANG || '
DENGAN ALAMAT ' || ALAMAT_TINGGAL || ', ' ||
KOTA_TINGGAL FROM MAHASISWA;MAHASISWA;
```

1.4 Baris Data Kembar

SQL *query* akan mengembalikan hasil *query* tanpa menghilangkan baris data kembar. Contohnya, ketika anda akan menampilkan semua *kode_prodi* dari tabel *mahasiswa*. Hasil dari *query* akan menunjukkan beberapa *kode_prodi* ditampilkan lebih dari sekali. Untuk menghilangkan baris data kembar dari hasil *query*, tambahkan kata kunci `DISTINCT` pada klausa `SELECT` segera setelah kata kunci `SELECT`.

Example:

```
SELECT DISTINCT KODE_PRODI FROM MAHASISWA;
```

Sebagai catatan, anda bisa menyebutkan banyak kolom setelah kata kunci DISTINCT. Kata kunci DISTINCT berpengaruh semua kolom yang dipilih, dan menghasilkan semua kombinasi yang berbeda dari kolom-kolom tesebut.

Example:

```
SELECT DISTINCT KODE_PRODI, NIM FROM
MAHASISWA;
```

1.5 *Column Alias*

Ketika menghasilkan hasil sebuah *query*, SQL akan menggunakan nama dari kolom-kolom yang dipilih/ekspresi pada `SELECT` statement sebagai header kolom. Terkadang nama kolom yang muncul sulit dimengerti.atau kurang bisa menjelaskan isi dari kolom. Anda bisa mengganti header dari kolom hasil *query* dengan menggunakan alias.

Tuliskan alias dari sebuah kolom pada klausa `SELECT` dengan menggunakan spasi sebagai pemisah. Jika alias dari sebuah kolom mengandung spasi atau karakter khusus, gunakan tanda petik 2 (“ “)

Example:

1. menampilkan NIM, KODE_MK dan “NILAI AKHIR’.

```
SELECT NIM, KODE_MK, (TUGAS*0.4) + (UTS*0.3) +
(UAS*03) AS "NILAI AKHIR" FROM NILAI;
```

2. menampilkan nama lengkap dari mahasiswa

```
SELECT NAMA_DEPAN || ' ' || NAMA_BELAKANG
"NAMA LENGKAP" FROM MAHASISWA;
```

2. Membatasi Hasil *Query*

Ketika akan mengambil data dari database, anda mungkin perlu untuk membatasi baris data yang ditampilkan, atau menampilkan hasil *query* berdasarkan kriteria tertentu. Anda bisa membatasi baris data yang dikembalikan oleh *query* dengan

menggunakan klausa `WHERE`. Klausa `WHERE` menampung sebuah kondisi yang harus dipenuhi, dan diletakkan setelah klausa `FROM`

Example:

1. menampilkan semua mahasiswa dengan kode_prodi '39010'

```
SELECT * FROM MAHASISWA WHERE KODE_PRODI =
'39020';
```

2. menampilkan semua mahasiswa dengan nama_depan 'Anthony'

```
SELECT * FROM MAHASISWA WHERE NAMA_DEPAN=
'Anthony';
```

Klausa `WHERE` dapat membandingkan nilai dalam kolom, perhitungan aritmatik, fungsi, atau nilai pasti. Klausa ini terdiri atas 3 (tiga) elemen:

- ✓ Nama Kolom
- ✓ Kondisi Pembanding
- ✓ Nama Kolom, nilai konstan, atau kumpulan nilai.

2.1 *Comparison Conditions*

Kondisi pembanding digunakan dalam kondisi yang membandingkan antara 1 ekspresi atau nilai dengan nilai lain atau ekspresi yang lain

Operator	Note
=	Sama dengan
>	Lebih besar dari
>=	Lebih besar sama dengan
<	Kurang dari

Operator	Note
<=	Kurang dari sama dengan
<>. !=, ^=	Tidak sama dengan
BETWEEN … AND …	Diantara 2 nilai
IN (LIST OF VALUES)	Dicocokkan dengan salah satu nilai pada kumpulan nilai
LIKE	Mencocokkan dengan pola karakter.
IS NULL	Adalah nilai *NULL*

Example:

1. Menampilkan nim dan uas dari mahasiswa yang nilai uas nya lebih besar dari 60.

```
SELECT NIM, UAS FROM NILAI WHERE UAS > 60;
```

2. (BETWEEN … AND …) *operator*

   Anda dapat menampilkan baris data berdasarkan sebuah jangkauan nilai dengan menggunakan operator BETWEEN. Jangkauan nilai yang disebutkan memiliki batas atas dan batas bawah. Nilai yang dituliskan dalam kondisi BETWEEN bersifat inklusif, dan anda harus menentukan nilai terendah terlebih dahulu. Anda juga bisa menggunakan kondisi BETWEEN untuk nilai karakter/huruf.

   Example:

   a. Menampilkan hanya nim dan UAS dari mahasiswa yang nilai UAS nya diantara 60 dan 80.

```
SELECT NIM, UAS FROM NILAI WHERE UAS BETWEEN
40 AND 80;
```

b. Menampilkan nama_depan dari mahasiswa, yang nama_depannya diantara ‘Anthony’ dan‘Desi’.

```
SELECT NAMA_DEPAN FROM MAHASISWA WHERE
NAMA_DEPAN BETWEEN 'Anthony' and 'Desi';
```

3. (`IN`) *operator*

Untuk mencoba nilai dalam sebuah kumpulan nilai, gunakan kondisi `IN`. kondisi `IN` dapat digunakan dengan tipe data apapun. Jika tipe data huruf atau tipe data DATE digunakan dalam kumpulan nilai, tipe data tersebut harus dituliskan dengan tanda petik (' ').

Example:

a. Mengambil nama lengkap dari nim 09410100277, 06390100006, dan 06390100003.

```
SELECT NAMA_DEPAN ||' '|| NAMA_BELAKANG FROM
MAHASISWA WHERE NIM IN ('09410250121',
'10390200020', '08410170020');
```

b. Mengambil nim dan kode_mk dari mahasiswa dengan UAS 50,40,30

```
SELECT NIM, KODE_MK FROM NILAI WHERE UAS IN
(50,40,30);
```

4. (`LIKE`) *operator*

   Terkadang, anda tidak mengerti nilai apa yang akan dicari, tetapi anda dapat menampilkan baris data yang cocok dengan pola karakter dengan menggunakan kondisi LIKE. Operasi mencocokkan pola karakter disebut juga dengan pencarian *wildcard*. 2 (dua) simbol dapat digunakan untuk membangun kata kunci pencari:

Symbol	Description
%	mewakilkan 0 atau lebih karakter
_	Mewakilkan 1 karakter

Example:

a. Menampilkan nama depan dari mahasiswa yang nama depannya diawali huruf ‘A’

```
SELECT NAMA_DEPAN FROM MAHASISWA WHERE
NAMA_DEPAN LIKE 'A%';
```

b. Menampilkan nama belakang dari mahasiswa yang huruf kedua dari nama belakangnya adalah ‘i’

```
SELECT NAMA_BELAKANG FROM MAHASISWA WHERE
NAMA_BELAKANG LIKE '_i%';
```

c. Menampilkan nama depan dari mahasiswa yang nama depannya berakhir dengan ‘u’

```
SELECT NAMA_DEPAN FROM MAHASISWA WHERE
NAMA_DEPAN LIKE '%u';
```

d. Menampilkan nama belakang dari mahasiswa yang nama belakangnya mengandung huruf 'd'

```
SELECT NAMA_BELAKANG FROM MAHASISWA WHERE
NAMA_BELAKANG LIKE '%d%';
```

Ketika anda perlu mencari dengan menggunakan karakter % dan _ yang sebenarnya, anda bisa menggunakan opsi `ESCAPE`. Sebagai contoh, ketika kita akan mengambil semua kode_mk dan nama_mk dari matakuliah yang kode_mk-nya dimulai dengan 'M_'

```
SELECT KODE_MK, NAMA_MK FROM MATAKULIAH WHERE
KODE_MK LIKE 'M\_%' ESCAPE '\';
```

Opsi `ESCAPE` menunjukkan garis miring (\) sebagai karakter `ESCAPE`. Pada *query* diatas, karakter ESCAPE mengawali garis bawah (_). Dengan opsi ini, maka garis bawah (_) akan dibaca sebagai karakter, bukan simbol pada perintah SQL.

2.2 *Logical Conditions*

Sebuah kondisi logis menggabungkan hasil dari 2 komponen kondisi untuk menghasilkan sebuah hasil berdasarkan 2 kondisi tersebut. Atau membalikkan hasil dari sebuah kondisi. Sebuah baris data akan dikembalikan jika hasil keseluruhan dari kondisi adalah TRUE. Ada 3 (tiga) operator logis yang ada di SQL:

Operator	Meaning
AND	mengembalikan TRUE jika kedua komponen kondisi mengembalikan TRUE
OR	Mengembalikan TRUE jika salah satu komponen kondisi mengembalikan TRUE
NOT	Mengembalikan TRUE jika komponen kondisi mengembalikan FALSE

1. `AND` *Operator*

   Operator `AND` membutuhkan kedua komponen kondisi untuk bernilaiTRUE. Tabel berikut akan menunjukkan hasil dari menggabungkan 2 kondisi menggunakan operator `AND`:

AND		Kondisi 1		
		TRUE	FALSE	NULL
Kondisi 2	TRUE	TRUE	FALSE	NULL
	FALSE	FALSE	FALSE	FALSE
	NULL	NULL	FALSE	NULL

Example:

a. Mengambil data mahasiswa yang tinggal di kota Sidoarjo dan nama depannya dimulai dengan huruf'A'

```
SELECT * FROM MAHASISWA WHERE KOTA_TINGGAL =
'Sidoarjo' AND NAMA_DEPAN LIKE 'A%';
```

b. Mengambil nim, kode_mk, dan nilai akhir dari mahasiswa yang nilai akhirnya lebih besar dari 70 dan kode_mk nya mengandung angka 1

```
SELECT NIM, KODE_MK, (TUGAS*0.4) + (UTS*0.3) +
(UAS*0.3) AS "FINAL SCORE" FROM NILAI
WHERE (TUGAS*0.4) + (UTS*0.3) + (UAS*0.3)> 70
AND KODE_MK LIKE '%1%';
```

2. `OR` *Operator*

Operator `OR` membutuhkan salah satu dari komponen kondisi untuk bernilai `TRUE`. tabel berikut akan menunjukkan hasil dari menggabungkan 2 kondisi menggunakan operator `OR`:

OR		Kondisi 1		
		TRUE	FALSE	NULL
Kondisi 2	TRUE	TRUE	TRUE	TRUE
	FALSE	TRUE	FALSE	NULL
	NULL	TRUE	NULL	NULL

Example:

Mengambil data matakuliah yang jumlah sks-nya 3 atau semester diadakannya 1

```
SELECT * FROM MATAKULIAH WHERE SKS = 3 OR
SEMESTER = 1;
```

3. `NOT` *Operator*

   Operator `NOT` membutuhkan sekumpulan nilai yang tidak ingin ditampilkan di hasil *query*. Tabel di bawah ini akan menunjukkan hasil penggunaan `NOT` di sebuah kondisi.

NOT	Kondisi		
	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

   Operator NOT bisa juga digunakan dengan operator SQL lain seperti `BETWEEN`, `LIKE` dan `NULL`.

   Example:

   1. Mengambil nim, kode_mk dan nilai akhir dari mahasiswa dengan nilai akhir tidak diantara 60 sampai 100, dan kode_mk-nya mengandung angka 1.

```
SELECT NIM, KODE_MK, (TUGAS*0.4) + (UTS*0.3) +
(UAS*0.3) AS "FINAL SCORE" FROM NILAI
WHERE (TUGAS*0.4) + (UTS*0.3) + (UAS*0.3) NOT
BETWEEN 60 AND 100 AND KODE_MK LIKE '%1%';
```

   2. Mengambil data mahasiswa yang nama depannya tidak mengandung huruf ‘i’ dan tidak dimulai dengan huruf ‘A’

```
SELECT * FROM MAHASISWA WHERE NAMA_DEPAN NOT
LIKE '%I%' OR NAMA DEPAN NOT LIKE 'A%';
```

3. Mengurutkan Data

Urutan dari baris data yang ditampilkan oleh sebuah hasil *query* tidak diketahui. Klausa `ORDER BY` dapat digunakan untuk mengurutkan baris data. Klausa `ORDER BY` diletakkan pada bagian paling akhir pada sebuah *query* SQL. Anda dapat menuliskan sebuah ekspresi, alias, atau posisi kolom sebagai kondisi `SORT`. Secara *default*, SORT mengurutkan secara *ascending* (mengurutkan dari nilai terendah ke nilai tertinggi). Untuk membalik urutan baris data yang ditampilkan, tambahkan keyword `DESC` setelah nama kolom pada klausa `ORDER BY`.

Example:

1. Menampilkan seluruh data mahasiswa yang diurutkan huruf depannya dari 'Z' ke 'A'.

```
SELECT * FROM MAHASISWA ORDER BY NAMA_DEPAN
DESC;
```

2. Menampilkan seluruh data mahasiswa yang diurutkan berdasarkan kota tinggal dan nama depan.

```
SELECT * FROM MAHASISWA ORDER BY KOTA_TINGGAL,
NAMA_DEPAN;
```

3. Menampilkan data mata kuliah diurutkan berdasarkan SKS

```
SELECT * FROM MATAKULIAH ORDER BY SKS;
```

Practice

Build The Command!

1. Mengambil semua data program studi.
2. Mengambil semua data kegiatan yang terjadi pada tanggal yang diadakan tanggal 12 (bulan dan tahun apapun)
3. Mengambil NIM dan nama lengkap dari mahasiswa yang nomor telepon aktifnya dimulai dengan '0888'.
4. Mengambil kode matakuliah dan nama matakuliah yang grade minimumnya bukan 'B' atau 'C'.
5. mengambil nim, nama lengkap, dan alamat_tinggal dari mahasiswa yang status menikahnya B dan tinggal di kota Surabaya.
6. Mengambil data mahasiswa yang bukan merupakan angkatan '07'.
7. Mengambil nim, kode_mk, nid, dan nilai akhir dari mahasiswa hanya yang nilai akhirnya lebih besar 70 dan kode matakuliahnya mengandung 'M_', dan diurutkan berdasarkan nilai akhir terbesar.
8. Menampilkan Data Mahasiswa yang nama belakangnya tidak ada.

Stage 2

Single-Row Functions

"*Strecth Your Hand To The Dream You Hold Dear* "

人

Tujuan Hari Ini

menggunakan *function* untuk memanipulasi tipe data tertentu dan *function* yang bisa digunakan di semua tipe data

What We Study

Character Functions

Number Functions

Date Functions

Data Type Conversion Functions

General Functions

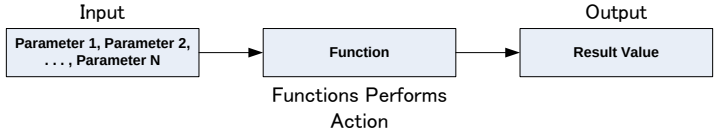
Conditional Expressions

SQL Functions

Functions adalah sebuah fitur yang sangat kuat dari SQL. *Functions* bisa digunakan untuk melakukan hal - hal berikut:

- ✓ Melakukan kalkulasi pada data.
- ✓ Mengubah data individual
- ✓ Mengubah *output* untuk sekumpulan baris data.
- ✓ Membentuk tampilan tanggal dan angka.
- ✓ Mengubah tipe data kolom

SQL Functions terkadang menerima beberapa parameter dan selalu mengembalikan nilai.



SQL *Function*

Single-Row Function digunakan untuk memanipulasi baris data. *Single-row function* menerima 1 (satu) atau lebih *input parameter* / *argument* dan mengembalikan 1 (satu) nilai untuk setiap baris yang dikembalikan oleh *query*.

Sebuah *input parameter* / *argument* dapat terdiri atas / diisi dengan:

- ✓ Nilai konstan yang dimasukkan pengguna
- ✓ Nilai variabel
- ✓ Nama kolom

- ✓ Ekspresi

Fitur-fitur yang terdapat dalam *single-row function*, meliputi:

- ✓ Bekerja pada setiap baris data yang dikembalikan oleh *query*.
- ✓ Mengembalikan 1 hasil untuk setiap barisnya.
- ✓ Memungkinkan untuk mengembalikan sebuah nilai yang berbeda tipe data dengan tipe data yang direferensi.
- ✓ Dapat menerima 1 atau lebih argument.
- ✓ Dapat digunakan pada klausa `SELECT`, `WHERE`, dan `ORDER BY`. Dapat digunakan secara bertingkat (*function* dalam *function*).

```
FUNCTION NAME (ARG 1, ARG 2, …, ARG N)
```

Pada *Syntax* ini:

`FUNCTION_NAME`: nama dari *function*.

`(ARG 1, ARG 2, …, ARG N`: nilai *argument / input parameter* apapun yang akan digunakan oleh *function*.

1. *Character Functions*

Single-row character function menerima data karakter sebagai masukan dan dapat mengembalikan baik nilai karakter dan nilai angka. *Character function* dapat dibagi menjadi 2:

- ✓ *Case-manipulation function*
- ✓ *Character-manipulation function*

1.1 *Case-Manipulation Functions*

Fungsi-fungsi berikut dapat mengubah *case* dari 1 (satu) atau sekumpulan karakter.

Function	Result
LOWER('SQL COURSE')	sql course
UPPER('SQL COURSE')	SQL COURSE
INITCAP('SQL COURSE')	Sql Course

- ✓ LOWER: mengubah karakter *mixed-case* atau *uppercase* menjadi *lowercase*.
- ✓ UPPER: mengubah karakter *mixed-case* atau *lowercase* menjadi *uppercase*.
- ✓ INITCAP: mengubah setiap huruf depan dari kata menjadi *uppercase* dan membiarkan huruf yang lain tetap *lowercase*.

1.2 *Character-Manipulation Functions*

FUNCTION	RESULT
CONCAT('HELLO','WORLD')	HELLOWORLD
SUBSTR('HELLO WORLD',1,5)	HELLO
LENGTH('HELLOWORLD')	10
INSTR('HELLOWORLD',W)	6
LPAD(5000,10,'*')	******5000
RPAD('MASKED',8,'+')	MASKED++
REPLACE('ROAD','R','BR')	BROAD
TRIM('D' FROM 'BROAD')	BROA

- ✓ `CONCAT`: menggabungkan 2(dua) nilai menjadi 1 (satu). `CONCAT` hanya dapat menggabungkan 2 *parameter* / *argument*.
- ✓ `SUBSTR`: mengambil karakter dari posisi karakter dan panjang yang disebutkan.
- ✓ `LENGTH`: mengambil panjang sebuah *string* karakter menjadi nilai angka.
- ✓ `INSTR`: mengambil posisi (dalam bentuk angka) dari karakter yang disebutkan.
- ✓ `LPAD`: menambahkan karakter di bagian kiri *string* karakter
- ✓ `RPAD`: menambahkan karakter di bagian kanan *string* karakter.
- ✓ `REPLACE`: menggantikan karakter yang disebut pada *string* karakter dengan karakter / *string* karakter lain.
- ✓ `TRIM`: menghapus karakter yang disebutkan dari huruf paling awal atau paling akhir (atau keduanya) dari *string* karakter. Dapat juga digunakan untuk menghapus spasi.

Example:

1. Menampilkan 3 huruf terdepan nama depan mahasiswa sebagai inisial dan ditampilkan *uppercase*.

```
SELECT UPPER(SUBSTR(NAMA_DEPAN,1,3))
"INISIAL" FROM MAHASISWA;
```

2. Mengambil panjang nama lengkap (tanpa spasi) dan hanya menampilkan yang panjangnya lebih dari 10 karakter.

```
SELECT NIM,
LENGTH(CONCAT(NAMA_DEPAN,NAMA_BELAKANG)
) "PANJANG NAMA" FROM MAHASISWA
WHERE
LENGTH(CONCAT(NAMA_DEPAN,NAMA_BELAKANG)
) > 10;
```

2. *Number Functions*

Number functions menerima masukan bertipe angka dan mengembalikan nilai angka sebagai hasil. Beberapa *number functions* yang ada antara lain:

Function	Result
ROUND(45.926,2)	45.93
TRUNC(45.956,2)	45.92
MOD(1600,300)	100

- ✓ `ROUND(COLUMN/EXPRESSION,N)`

  Membulatkan nilai pada kolom; ekspresi; atau nilai ke posisi desimal N, atau jika nilai N dihilangkan/tidak diisi, maka nilai akan dibulatkan tanpa posisi desimal. (jika nilai N negatif, maka angka-angka yang ada di sebelah kiri koma (*decimal point*) akan dibulatkan. (dibulatkan ke atas)

- ✓ `TRUNC(COLUMN/EXPRESSION,N)`

  Memotong nilai pada kolom; ekspresi; atau nilai ke posisi decimal N, atau jika nilai N dihilangkan/tidak diisi, maka nilai *default* N adalah 0. (dibulatkan ke bawah)

✓ `MOD(M,N)`

Mengembalikan nilai sisa dari hasil M dibagi dengan N. (sisa bagi)

Example:

1. Membulatkan ke atas (tanpa desimal) nilai akhir dari mahasiswa yang nimnya mengandung angka '39020'

```
SELECT NIM, KODE_MK,
ROUND((TUGAS*0.4) + (UTS*0.3) +
(UAS*0.3)) "NILAI AKHIR"
FROM NILAI WHERE NIM LIKE '%39020%'
ORDER BY 1;
```

2. Menampilkan data mahasiswa yang nimnya genap saja.

```
SELECT * FROM MAHASISWA WHERE
MOD(NIM,2) = 0;
```

3. *Date Functions*

Oracle Database menyimpan tanggal dalam sebuah format angka internal: abad, tahun, bulan, hari, jam, menit, dan detik.

Tampilan *default* untuk tanggal adalah DD-MON-RR.

✓ DD: 2 (dua) digit tanggal

- ✓ MON: 3 (dua) digit terdepan nama bulan
- ✓ RR : 2 (dua) digit terbelakang tahun.

Meskipun model tampilan tanggal hanya menampilkan 2 (dua) digit tahun, tanggal tidak disimpan seperti model tampilan ini di *database*. Semua komponen dari tanggal dan waktu juga ikut disimpan dalam *database*.

Untuk melihat tanggal hari ini, anda bisa menggunakan `SYSDATE` *function*. `SYSDATE` *Function* adalah sebuah *date function* yang mengembalikan tanggal dan waktu dari *server database* sekarang ini. SYSDATE bisa digunakan sebagaimana anda menggunakan nama kolom.

```
SELECT SYSDATE FROM DUAL
```

3.1 Operasi Aritmatik pada Tanggal

Karena database menyimpan tanggal sebagai angka, anda dapat melakukan perhitungan dengan operator aritmatik, seperti penambahan dan pengurangan.

Operation	Result	Description
date + number	date	menambahkan sejumlah hari ke tanggal
date - number	date	mengurangi sejumlah hari dari tanggal
date - date	number of days	mengurangi tanggal yang satu dengan yang lain
date + number/24	date	menambahkan sejumlah jam ke tanggal

3.2 *The Functions*

Function	Result
MONTHS_BETWEEN	jumlah bulan dari 2 (dua) buah tanggal.
ADD_MONTHS	menambahkan bulan kalender pada tanggal
NEXT_DAY	menampilkan hari berikutnya dari tanggal berdasarkan karakter string
LAST_DAY	menampilkan hari terakhir dari bulan berdasarkan tanggal
ROUND	membulatkan tanggal ke atas berdasarkan format
TRUNC	membulatkan tanggal ke bawah berdasarkan format.

- ✓ `MONTH_BETWEEN(DATE1, DATE2)`

  Mengembalikan jumlah bulan antara *date1* dan *date2*, dan dapat mengembalikan nilai negatif.
- ✓ `ADD_MONTHS(DATE,N)`

  Menambahkan bulan sejumlah `N` ke `DATE`. Nilai dari N harus angka dan bisa negative.
- ✓ `NEXT_DAY(DATE,'CHAR')`

  Menemukan tanggal dari hari berikutnya dalam minggu (`'CHAR'`), setelah `DATE`. Nilai (`'CHAR'`) dapat berisi angka yang mewakilkan hari atau *string* karakter.
- ✓ `LAST_DAY(DATE)`

  Menemukan tanggal dari hari terakhir pada bulan yang menampung `DATE`.
- ✓ `ROUND(DATE[,'FMT'])`

  Mengembalikan tanggal yang dibulatkan ke atas berdasarkan `'FMT'`. Jika `'FMT'` tidak disebutkan, maka tanggal akan dibulatkan ke hari yang terdekat.

- ✓ `TRUNC(DATE,'FMT')`

  Mengembalikan tanggal yang dibulatkan ke bawah berdasarkan `FMT'`. Jika `'FMT'` tidak disebutkan, maka tanggal akan dibulatkan ke hari yang terdekat.

Example:

1. Menampilkan nim, nama lengkap dan umur setiap mahasiswa yang umurnya sudah lebih dari 24 tahun dan kode_prodi-nya = '41015'.

```
SELECT NIM, NAMA_DEPAN ||' '||
NAMA_BELAKANG "NAMA LENGKAP", ROUND
(MONTHS_BETWEEN(SYSDATE,TGL_LAHIR)/12)
"UMUR" FROM MAHASISWA WHERE
MONTHS_BETWEEN(SYSDATE,TGL_LAHIR)/12 >
24 AND KODE_PRODI = '41015';
```

2. Menampilkan tanggal untuk hari pertama minggu depan

```
SELECT NEXT_DAY(SYSDATE, 'MONDAY')
FROM DUAL;
```

4. *Data Type Conversions*

Pada kondisi tertentu, server ORACLE menggunakan data dari salah satu tipe data ketika membutuhkan data dari tipe data yang berbeda. Ketika kondisi ini muncul, server ORACLE dapat mengubah data ke tipe data yang dibutuhkan secara otomatis. Perubahan tipe data ini dapat dilakukan secara implisit oleh server ORACLE atau secara eksplisit oleh pengguna.

Secara implisit, server ORACLE dapat mengubah secara otomatis tipe data berikut:

From	To
Varchar2 or Char	Number
Varchar2 or Char	Date
Number	Varchar2
Date	Varchar2

Sedangkan secara eksplisit, SQL pada server ORACLE memiliki 3 (tiga) fungsi untuk mengubah sebuah nilai ke tipe data lain:

- ✓ `TO_CHAR(NUMBER|DATE,[fmt], [nlsparams])`

  Mengubah sebuah nilai `NUMBER` atau `DATE` menjadi sebuah karakter *string* `VARCHAR2` dengan format `fmt`.

  Number Conversion: Parameter `nlsparams` menyebutkan karakter-karakter yang akan dikembalikan oleh elemen format number. Jika `nlsparams` atau parameter lain tidak diisi, maka fungsi akan menggunakan nilai parameter *default.*

  Date Conversion: Parameter `nlsparams` menyebutkan dalam bahasa tanggal apa bulan, nama hari hari, serta singkatan akan dikembalikan. Jika parameter `nlsparams` ini tidak diisi, maka fungsi akan menggunakan bahasa tanggal *default*.

- ✓ `TO_NUMBER(CHAR,[fmt], [nlsparams])`

  Mengubah sebuah karakter string yang memuat angka menjadi tipe data `NUMBER` dengan format sesuai pada `fmt`.

Parameter `[nlsparams]` di fungsi ini memiliki kesamaan dengan parameter `[nlsparams]` pada *function* `TO_CHAR` untuk *number conversion*.

- ✓ `TO_DATE(CHAR,[fmt], [nlsparams])`

  Mengubah sebuah karakter string yang mewakili tanggal ke nilai dengan tipe data `DATE` berdasarkan `fmt` yang disebutkan. Jika `fmt` tidak diisi, maka format yang dikembalikan adalah `DD-MON-YY` (akan dijelaskan nanti).

  Parameter `[nlsparams]` di fungsi ini memiliki kesamaan dengan parameter `[nlsparams]` pada *function* `TO_CHAR` untuk *date conversion.*

Telah dijelaskan sebelumnya, bahwa tanggal pada Oracle ditampilkan dengan format DD-MON-YY, tetapi anda dapat menggunakan fungsi `TO_CHAR` untuk mengubah format tampilan dari sebuah tanggal ke format yang diinginkan. Elemen dari format tampilan tanggal adalah:

Element	Result
YYYY	Tahun dalam Angka
Year	Tahun dalam bentuk huruf
MM	2(dua)-digit angka bulan
MONTH	nama penuh dari bulan
MON	Singkatan 3 (tiga) huruf dari bulan
DY	Singkatan 3 (tiga) huruf dari hari dalam seminggu
DAY	Nama Lengkap dari hari dalam seminggu
DD	angka hari dalam sebulan
ddspth	Menuliskan tanggal dalam huruf

Example:

1. Menampilkan nim, nama depan mahasiswa, dan tanggal lahir dengan format [tanggal] [nama bulan penuh] [tahun angka penuh]

```
SELECT NIM, NAMA_DEPAN,
TO_CHAR(TGL_LAHIR, 'DD MONTH YYYY')
FROM MAHASISWA;
```

2. Menampilkan nim, nama lengkap, alamat, dan kota yang kode_prodinya '41017' dan tanggal lahirnya adalah 15 February 1987.

```
SELECT NIM, NAMA_DEPAN || ' ' ||
NAMA_BELAKANG AS NAMA_LENGKAP,
ALAMAT_TINGGAL, KOTA_TINGGAL FROM
MAHASISWA
WHERE KODE_PRODI = '41017' AND
TO_CHAR(TGL_LAHIR, 'DD MONTH YYYY') =
'14 FEBRUARY  1988';
```

5. *General Functions*

Fungsi-fungsi berikut ini dapat digunakan dengan tipe data apapun dan berhubungan dengan penggunaan nilai NULL:

- ✓ `NVL (expr1, expr2)`

  Mengubah sebuah nilai NULL menjadi nilai yang tertentu.

- ✓ `NVL2 (expr1, expr2, expr3)`

Jika `expr1` bernilai `NULL`, maka fungsi akan mengembalikan nilai pada `expr3`; sedangkan jika `expr1` tidak bernilai null, maka fungsi akan mengembalikan nilai pada `expr2`.

- ✓ `NULLIF (expr1, expr2)`

  Membandingkan 2 (dua) ekspresi dan mengembalikan `NULL` jika ekspresi tersebut sama. Jika kedua ekspresi tidak sama, maka fungsi akan mengembalikan `expr1`.

- ✓ `COALESCE (expr1, expr2, . . ., exprN)`

  Mengembalikan ekspresi non-`NULL` pertama dari daftar ekspresi.

Example

1. Menampilkan nama belakang mahasiswa yang tidak memiliki nama belakang sebagai "unknown"

```
SELECT nama_depan, nvl(nama_belakang,
'unknown') from mahasiswa;
```

2. Menampilkan daftar nim mahasiswa dan keterangan apakah memiliki nama belakang atau tidak.

```
SELECT nim, nvl2(nama_belakang, 'ada
nama belakang', 'tidak ada nama
belakang') "keterangan" from mahasiswa;
```

6. *Conditional Expression*

Conditional expression digunakan untuk fungsi percabangan (`IF-THEN-ELSE` LOGIC) dalam perintah SQL. *Conditional expression* dapat dilakukan dengan 2 (dua) metode: ekspresi `CASE` dan ekspresi `DECODE`.

✓ `CASE` *Expression*

```
CASE expr WHEN comparison_expr1 THEN
return_expr1
         [WHEN comparison_expr2 THEN
return_expr2
          WHEN comparison_exprN THEN
return_exprN
          ELSE else_expr]
END
```

✓ `DECODE` *Expression*

```
DECODE (expr, search1, result1,
              [search2, result2,
               searchN, resultN,
               default])
```

Example:

1. Menampilkan nim, nama_depan, status_nikah mahasiswa sebagai menikah atau belum menikah

```
SELECT NIM, NAMA_DEPAN, CASE STATUS_NIKAH WHEN 'B' THEN 'BELUM MENIKAH'
                                          WHEN 'M' THEN 'MENIKAH'
                        END "STATUS NIKAH"
FROM MAHASISWA
```

2. Menampilkan nim, nama_depan, status mahasiswa (sebagai aktif dan tidak aktif), dan Jenis kelamin (sebagai pria dan wanita)

```
SELECT NIM, NAMA_DEPAN, CASE STATUS_MHS WHEN 'A' THEN 'AKTIF'
                                        WHEN 'T' THEN 'TIDAK AKTIF'
                        END "STATUS MAHASISWA"
                      , DECODE(JENIS_KELAMIN, 'P', 'PRIA',
                                              'W', 'WANITA') "JENIS KELAMIN"
FROM MAHASISWA
```

3. Menampilkan keterangan ‘work over 15 years’ bagi karyawan yang sudah bekerja lebih dari 15 tahun, dan keterangan ‘work in progress’ untuk karyawan yang belum bekerja lebih dari 15 tahun.

```
select employee_id, last_name,
case when (sysdate-hire_date)/365 > 15 then 'Work Over 15 Years'
                                       else 'Work in Progress'
                                       end "Terms of Office"
from employees
```

Practice

Build The Command!

1. Menampilkan Nim, Kode_MK, NID, Nilai Akhir, Grade, dan Status Lulus, dengan aturan jika Grade A+, A, B+, B maka dianggap lulus, selain itu tidak lulus.

   Penghitungan Grade:

   Nilai <= 69 = D

   70 - 74 = C

   75 - 79 = B

   80 - 84 = B +

   85 - 89 = A

   90- 100 = A+
2. Menampilkan Password dari dosen, yaitu dengan menggabungkan 3 (tiga) karakter awal (dan dijadikan uppercase) dari nama dosen dengan id_dosen.
3. Menampilkan Data Mahasiswa yang lahir pada quartal kedua, dan bulan genap
4. Menampilkan NIM, nama lengkap, dan tanggal lahir mahasiswa dengan format (hari, tanggal dalam huruf "of" nama bulan penuh, tahun lengkap) ex: (monday, the twenty-eighth of January, 2012)

Stage 3

Group Functions / Aggregate Functions

"*You're Weak When You Say You're Weak*"

人

Tujuan Hari Ini

menggunakan *function* untuk menghasilkan informasi berdasarkan pengelompokan

What We Study

Group Function and its Types

Condition in Group Function Result

1. *Group Function and Its Types*

1.1 *Types Of Group Function*

Tidak seperti pada *single-row function*, *Group Function* atau *Aggregate Function* bekerja pada sekumpulan baris data untuk mengembalikan 1 buah hasil untuk setiap groupnya. Sekumpulan baris data yang dimaksud bisa melingkupi 1 (satu) tabel, atau sebuah tabel yang dibagi menjadi beberapa group.

Tipe-Tipe dari *Group Function* adalah sebagai berikut:

Function	Description
`AVG([DISTINCT|ALL] n)`	mengembalikan rata-rata nilai dari `n` dengan mengabaikan nilai NULL.
`COUNT({*|[DISTINCT|ALL] expr})`	mengembalikan jumlah dari baris data. `Expr` akan mengembalikan nilai yang tidak NULL saja. (`COUNT ALL` dengan menggunakan * akan mengembalikan semua data kembar dan baris data dengan nilai NULL)
`MAX ([DISTINCT|ALL] expr)`	mengembalikan nilai maksimum dari `expr`, dengan mengabaikan nilai NULL.
`MIN ([DISTINCT|ALL] expr)`	mengembalikan nilai minimum dari `expr`, dengan mengabaikan nilai

Function	Description
	NULL.
STDDEV([DISTINCT|ALL] n)	mengembalikan standar deviasi dari n, dengan mengabaikan nilai NULL
SUM([DISTINCT|ALL] n)	mengembalikan jumlah nilai dari n, dengan mengabaikan nilai NULL.
VARIANCE([DISTINCT|ALL] x)	mengembalikan nilai varian dari x, dengan mengabaikan nilai NULL.

Syntax sederhana yang menggunakan *group function*:

```
 SELECT [COLUMN,] GROUP_FUNCTION(COLUMN),. . .
 FROM TABLE
[WHERE CONDITION]
[GROUP BY COLUMN]
[ORDER BY COLUMN];
```

Beberapa pedoman dalam menggunakan *group function*:

- ✓ Keyword DISTINCT akan membuat function membaca nilai-nilai yang tidak kembar saja. ALL akan membuat function membaca semua nilai, termasuk nilai kembar. *Default* dari sebuah function adalah ALL.
- ✓ Tipe data yang dapat digunakan pada function dengan parameter expr antara lain *CHAR*, *VARCHAR2*, *NUMBER*, atau *DATE*.

- ✓ Semua group function mengabaikan nilai NULL. Untuk menggantikan nilai NULL dengan nilai lain, gunakan function NVL, NVL2,atau COALESCE.

Anda dapat menggunakan AVG, SUM, MIN, dan MAX *Function* pada kolom yang menyimpan data bertipe angka. Anda juga dapat menggunakan MIN dan MAX *Function* pada kolom yang menyimpan data bertipe angka, karakter (huruf), dan tanggal. Sebagai catatan:

- ✓ VARIANCE dan STDDEV *Function* hanya dapat digunakan pada tipe data angka.
- ✓ MAX dan MIN *Function* tidak bisa digunakan pada tipe data LOB atau tipe data LONG.

Example:

1. Menampilkan tanggal lahir mahasiswa dengan umur paling tua.

```
SELECT MIN(TGL_LAHIR)FROM MAHASISWA;
```

2. Menampilkan rata-rata nilai akhir dari matakuliah dengan kode_mk ‘P_010’

```
SELECT AVG((TUGAS*0.4) + (UTS*0.3) +
(UAS*0.3)) "RATA-RATA NILAI AKHIR"
FROM NILAI WHERE KODE_MK = 'P_010';
```

3. Menampilkan Total SKS matakuliah pada semester 3 dan hanya yang matakuliahnya wajib ‘B’

```
SELECT SUM(SKS) FROM MATAKULIAH WHERE
SEMESTER = 3 AND GRADE_MINIMUM = 'B';
```

4. Menampilkan jumlah mahasiswa yang berasal dari 'Surabaya' dan nama depannya diawali oleh huruf 'A'

```
SELECT COUNT(NIM) FROM MAHASISWA
WHERE KOTA_TINGGAL = INITCAP('SURABAYA')
AND NAMA_DEPAN LIKE 'A%';
```

5. Menampilkan nilai uas tertinggi pada matakuliah dengan kode_mk 'P_011'

```
SELECT MAX(UAS) FROM NILAI
WHERE KODE_MK = 'P_011';
```

1.2 *Grouping Data*

Pada waktu tertentu, anda harus membagi sebuah tabel menjadi group-group yang lebih kecil. Hal ini dapat dilakukan dengan menggunakan klausa `GROUP BY`.

Anda dapat menggunakan klausa `GROUP BY` untuk membagi baris data pada sebuah tabel ke dalam group-group, kemudian anda dapat menggunakan *Group Function* untuk mengembalikan ringkasan informasi yang dibutuhkan untuk setiap group.

```
 SELECT [COLUMN,] GROUP_FUNCTION(COLUMN),. . .
 FROM TABLE
[WHERE CONDITION]
[GROUP BY Group_by_expression(s)]
[ORDER BY COLUMN];
```

Pada *Syntax* ini `Group_by_expression` diisi dengan kolom yang nilainya dijadikan dasar dalam mengelompokkan baris data.

Panduan dalam menggunakan `GROUP BY`:

- ✓ Jika anda memasukkan kolom lain selain *group function* pada klausa `SELECT`, maka semua kolom lain tersebut harus dimasukkan ke dalam klausa `GROUP BY`.
- ✓ Dengan menggunakan klausa `WHERE`, anda dapat menyaring baris data sebelum digroupkan.
- ✓ anda tidak dapat menggunakan *Column Alias* (*Stage* 1) pada klausa `GROUP BY`.
- ✓ Jika klausa `GROUP BY` diisi dengan lebih dari 1 (satu) kolom, maka pengelompokan akan dimulai dari kolom paling pertama yang disebutkan, lalu dilanjutkan dengan kolom berikutnya (dalam pengelompokan sebelumnya).
- ✓ Anda dapat menggunakan klausa `GROUP BY` tanpa menggunakan *Group Function.*

Example:

1. Menampilkan jumlah mahasiswa per prodi dan per kota.

```
SELECT KODE_PRODI, KOTA_TINGGAL,
COUNT(NIM) "Jumlah Mahasiswa" FROM
MAHASISWA
GROUP BY KODE_PRODI, KOTA_TINGGAL;
```

2. Menampilkan rata-rata umur mahasiswa yang sudah menikah dan yang belum menikah untuk kode prodi '39020'.

```
SELECT status_nikah,
round(avg(months_between(sysdate,tgl_lahi
r)/12)) "Rata-rata Umur"
from mahasiswa where kode_prodi = '39020'
group by status_nikah;
```

3. Menampilkan nilai akhir tertinggi untuk setiap kode_mk yang mengandung angka 2.

```
SELECT KODE_MK, MAX((TUGAS*0.4) +
(UTS*0.3) + (UAS*0.3)) "NA TERTINGGI" FROM
NILAI
WHERE KODE_MK LIKE '%2%' GROUP BY KODE_MK;
```

4. Menampilkan rata-rata nilai tertinggi dari semua matakuliah

```
SELECT
MAX(AVG((TUGAS*0.4)+(UAS*0.3)+(UTS*0.3)))
FROM NILAI GROUP BY KODE_MK
```

2. *Condition in Group Function Result*

Sama seperti klausa `WHERE` yang digunakan untuk menyaring baris data yang anda `SELECT`, anda juga dapat menyaring hasil dari *Group Function* dengan menggunakan klausa `HAVING`. Anda dapat menggunakan klausa `HAVING` untuk menyaring group mana yang akan ditampilkan berdasarkan hasil *group function.*

```
 SELECT [COLUMN,] GROUP_FUNCTION(COLUMN),. . .
 FROM TABLE
[WHERE CONDITION]
[GROUP BY Group_by_expression(s)]
[HAVING Group_Condition(s)]
[ORDER BY COLUMN];
```

Pada *Syntax* ini *`Group_Condition(s)`* diisi dengan kondisi untuk menyaring group mana yang akan ditampilkan berdasarkan hasil *Group Function*.

Example:

1. Menampilkan semester dan jumlah sks pada semester tersebut, dan hanya menampilkan yang jumlah sks nya di atas 10.

```
SELECT SEMESTER, SUM(SKS) "JUMLAH SKS" FROM
MATAKULIAH GROUP BY SEMESTER HAVING
SUM(SKS)>10;
```

2. Menampilkan jumlah mahasiswa per kota tinggal mahasiswa dan tampilkan yang jumlah mahasiswanya lebih dari 8.

```
SELECT KOTA_TINGGAL, COUNT(NIM) "JUMLAH
MHS" FROM MAHASISWA GROUP BY
KOTA_TINGGAL HAVING COUNT(NIM) > 8;
```

3. Menampilkan umur paling tua untuk setiap kode_prodi dan menampilkan hanya yang umurnya diantara 25 sampai 28 tahun dan statusnya adalah aktif (‘A’).

```
SELECT kode_prodi,
max(round(months_between(sysdate,tgl_lahir
)/12)) "Umur Paling Tua" from mahasiswa
where Status_mhs = 'A' group by kode_prodi
having
max(months_between(sysdate,tgl_lahir)/12)
between 25 and 28;
```

4. Menampilkan jumlah mahasiswa per kota tinggal dan per angkatan dari angkatan 2006 sampai 2008.

```
1 SELECT KOTA_TINGGAL, SUM(DECODE(SUBSTR(NIM, 1, 2), '06', 1, 0)) "ANGKATAN 2006",
2                      SUM(DECODE(SUBSTR(NIM, 1, 2), '07', 1, 0)) "ANGKATAN 2007",
3                      SUM(DECODE(SUBSTR(NIM, 1, 2), '08', 1, 0)) "ANGKATAN 2010"
4 FROM MAHASISWA
5 GROUP BY KOTA_TINGGAL;
```

Practice

***Build The Command*!**

1. Menampilkan jumlah mahasiswa yang mengambil matakuliah dengan kode MK 'P_010'
2. Menampilkan kode matakuliah dan jumlah mahasiswa per angkatan untuk angkatan 2006 sampai 2008 beserta total mahasiswa dari angkatan 2006 sampai 2008 yang mengambil mata kuliah tersebut.
3. Menampilkan nilai akhir tertinggi, nilai akhir terendah, rata-rata nilai akhir, dan selisih antara nilai akhir tertinggi dan nilai akhir terendah untuk setiap kode matakuliah.
4. Menampilkan jumlah plotting harian dikelompokkan berdasarkan hari dan kode mk.

Stage 4

Obtain Data from Multiple Table

"***Nothing Impossible If You Have Determination***"

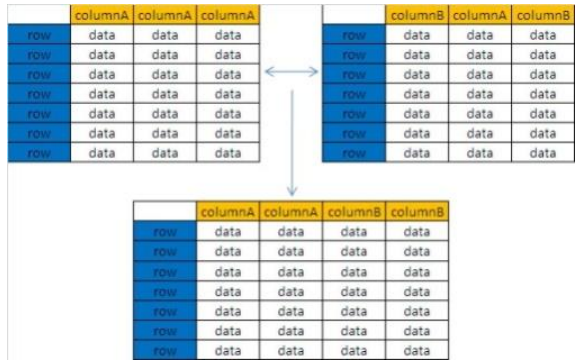
人

Tujuan Hari Ini

Melakukan pengambilan data dari 2 (dua) atau lebih tabel yang berbeda

What We Study

EQUIJOIN
OUTER JOIN
SELF JOIN



Terkadang anda perlu menggunakan data dari 2 (dua) atau lebih tabel. Untuk menggunakan data yang tersebar pada banyak tabel, anda harus menghubungkan tabel-tabel tersebut antara satu dengan yang lain. sebuah `JOIN` dapat digunakan untuk menampilkan informasi dari banyak tabel, sehingga anda bisa menggabungkan tabel-tabel secara bersamaan untuk menampilkan informasi yang dibutuhkan.

1. *EQUIJOIN*

Equijoin memerlukan kehadiran Primary Key dan Foreign Key pada tabel-tabel yang akan digabungkan. *Equijoin* juga biasa disebut sebagai *simple* `JOIN` atau *inner* `JOIN`. penulisan *equijoin* dapat dilakukan dengan menggunakan `USING`, `ON`, dan `WHERE`.

1.1 Menggunakan `USING`

Klausa `USING` dapat digunakan untuk menyebutkan kolom apa saja yang dijadikan referensi `JOIN` ketika ada lebih dari 1 (satu) kolom yang

cocok antara tabel-tabel yang digabungkan. Kolom-kolom yang digunakan pada klausa `USING` tidak boleh ditambahkan dengan nama tabel atau alias pada bagian lain dari query.

Example:

1. Menampilkan nama lengkap mahasiswa dan nama program studi dari seluruh mahasiswa

```
SELECT MAHASISWA.NAMA_DEPAN ||' '||
MAHASISWA.NAMA_BELAKANG "NAMA
LENGKAP", PROGRAM_STUDI.NAMA_PRODI
FROM MAHASISWA JOIN PROGRAM_STUDI
USING (KODE_PRODI);
```

2. Menampilkan nid, nama_dosen, kode_mk, jam mulai dan jam selesai untuk plotting pada hari rabu.

```
SELECT NID, DOSEN.NAMA_DOSEN,
PLOTTING_AJAR.KODE_MK,
PLOTTING_AJAR.JAM_MULAI,
PLOTTING_AJAR.JAM_SELESAI
FROM PLOTTING_AJAR JOIN DOSEN
USING (NID)
WHERE PLOTTING_AJAR.HARI = 'Rabu';
```

1.2 Menggunakan ON

Klausa `ON` dapat digunakan untuk menuliskan kondisi acak atau untuk menuliskan kolom yang digunakan sebagai kondisi `JOIN`. pada klausa

ON, kondisi JOIN dipisahkan dari kondisi pencarian lain yang dituliskan pada klausa WHERE.

Example:

1. Menampilkan nama lengkap mahasiswa dan nama program studi dari seluruh mahasiswa

```
SELECT MAHASISWA.NAMA_DEPAN ||' '||
MAHASISWA.NAMA_BELAKANG "NAMA
LENGKAP", PROGRAM_STUDI.NAMA_PRODI
FROM MAHASISWA  JOIN PROGRAM_STUDI
ON MAHASISWA.KODE_PRODI =
PROGRAM_STUDI.KODE_PRODI;
```

2. Menampilkan nid, nama_dosen, kode_mk, jam mulai dan jam selesai untuk plotting pada hari rabu.

```
SELECT DOSEN.NID, DOSEN.NAMA_DOSEN,
PLOTTING_AJAR.KODE_MK,
PLOTTING_AJAR.JAM_MULAI,
PLOTTING_AJAR.JAM_SELESAI
FROM PLOTTING_AJAR JOIN DOSEN
ON PLOTTING_AJAR.NID = DOSEN.NID
WHERE PLOTTING_AJAR.HARI = 'Rabu';
```

1.3 Menggunakan WHERE

JOIN dapat dilakukan juga dengan menggunakan klausa WHERE. jika kondisi JOIN diletakkan pada klausa WHERE, maka kata kunci JOIN dapat dihilangkan dari klausa FROM.

Example:

1. Menampilkan nama lengkap mahasiswa dan nama program studi dari seluruh mahasiswa

```
SELECT MAHASISWA.NAMA_DEPAN ||' '||
MAHASISWA.NAMA_BELAKANG "NAMA
LENGKAP", PROGRAM_STUDI.NAMA_PRODI
FROM MAHASISWA, PROGRAM_STUDI
WHERE MAHASISWA.KODE_PRODI =
PROGRAM_STUDI.KODE_PRODI;
```

2. Menampilkan nid, nama_dosen, kode_mk, jam mulai dan jam selesai untuk plotting pada hari rabu.

```
SELECT DOSEN.NID, DOSEN.NAMA_DOSEN,
PLOTTING_AJAR.KODE_MK,
PLOTTING_AJAR.JAM_MULAI,
PLOTTING_AJAR.JAM_SELESAI
FROM PLOTTING_AJAR, DOSEN
WHERE PLOTTING_AJAR.NID = DOSEN.NID
AND PLOTTING_AJAR.HARI = 'Rabu';
```

1.4 *Table Alias*

Pada 3 (tiga) cara di atas, nama kolom dituliskan bersamaan dengan nama tabelnya untuk menghindari nama kolom yang ambigu/kembar. Dan mempercepat performa query. Menuliskan nama kolom dengan nama tabel dapat memakan waktu, jika nama tabel yang dituliskan

panjang. Untuk menyelesaikan masalah ini, anda dapat menggunakan *table aliases* untuk menggantikan nama tabel.

Jika sebuah *table aliases* digunakan pada sebuah nama tabel di klausa `FROM`, maka *table aliases* tersebut harus digunakan untuk menggantikan nama tabel pada query tersebut.

Example:

1. Menampilkan nama lengkap mahasiswa dan nama program studi dari seluruh mahasiswa

```
SELECT M.NAMA_DEPAN ||' '||
M.NAMA_BELAKANG "NAMA LENGKAP",
P.NAMA_PRODI
FROM MAHASISWA M, PROGRAM_STUDI P
WHERE M.KODE_PRODI = P.KODE_PRODI;
```

2. Menampilkan nid, nama_dosen, kode_mk, jam mulai dan jam selesai untuk plotting pada hari rabu.

```
SELECT DSN.NID, DSN.NAMA_DOSEN,
PLOT.KODE_MK, PLOT.JAM_MULAI,
PLOT.JAM_SELESAI
FROM PLOTTING_AJAR PLOT JOIN DOSEN DSN
ON PLOT.NID = DSN.NID
WHERE PLOT.HARI = 'Rabu';
```

3. Menampilkan kode_mk, nama_mk, dan uas tertinggi dari setiap mata kuliah.

```
select mk.kode_mk, mk.nama_mk, max(n.uas)
"Nilai UAS Tertinggi" from matakuliah mk
join nilai n
on mk.kode_mk = n.kode_mk group by
mk.kode_mk, mk.nama_mk
```

4. Menampilkan Kode_MK, Nama MK, NID, Nama dosen, Jam Mulai, Jam Selesai, dan hari hanya untuk matakuliah 3 sks.

```
SELECT PLOT.KODE_MK, MK.NAMA_MK,
PLOT.NID,D.NAMA_DOSEN, PLOT.JAM_MULAI,
PLOT.JAM_SELESAI, PLOT.HARI
FROM DOSEN D JOIN PLOTTING_AJAR PLOT
ON D.NID = PLOT.NID
JOIN MATAKULIAH MK
ON PLOT.KODE_MK = MK.KODE_MK
WHERE MK.SKS = 3;
```

5. menampilkan jumlah skkm yang sudah didapat per mahasiswa

```
SELECT M.NIM,SUM(DET.SKKM)
FROM MAHASISWA M, KEAKTIFAN_MAHASISWA K,
DETAIL_JENIS_POSISI_EVENT DET
WHERE M.NIM = K.NIM AND K.ID_EVENT =
DET.ID_EVENT AND K.ID_JENIS =
DET.ID_JENIS
GROUP BY M.NIM;
```

2. *OUTER JOIN*

Pada *equijoin*, jika sebuah baris data tidak memenuhi kondisi `JOIN`, maka baris tersebut tidak akan ditampilkan pada hasil query. Untuk menampilkan baris-baris data yang tidak memenuhi kondisi `JOIN` tersebut, dapat digunakan *outer join*. *outer join* terdiri atas 3 (tiga) macam, yaitu: `LEFT OUTER JOIN`, `RIGHT OUTER JOIN`, dan `FULL OUTER JOIN`.

2.1 `LEFT OUTER JOIN`

`LEFT OUTER JOIN` akan menampilkan semua baris data dari tabel sebelah kiri (dari kata kunci `LEFT OUTER JOIN`) meskipun ada baris data yang tidak memenuhi kondisi `JOIN` dengan tabel sebelah kanan.

Example:

1. Menampilkan semua kode mk dan nama mk beserta nilai uas tertinggi dari mata kuliah tersebut beserta matakuliah yang belum memiliki nilai uas.

```
SELECT MK.KODE_MK, MK.NAMA_MK,
nvl(MAX(N.UAS),0)
FROM MATAKULIAH MK LEFT OUTER JOIN NILAI
N ON MK.KODE_MK = N.KODE_MK
GROUP BY MK.KODE_MK, MK.NAMA_MK;
```

2. Menampilkan semua nid dosen, nama dosen, dan plotting yang sudah dilakukan, beserta dosen-dosen yang belum melakukan plotting.

```
SELECT P.ID_PLOTTING, D.NID, D.NAMA_DOSEN,
P.KODE_MK, P.HARI, P.JAM_MULAI, P.JAM_SELESAI
FROM DOSEN D LEFT OUTER JOIN PLOTTING_AJAR P
ON D.NID = P.NID;
```

2.2 RIGHT OUTER JOIN

`RIGHT OUTER JOIN` akan menampilkan semua baris data dari tabel sebelah kanan (dari kata kunci `RIGHT OUTER JOIN`) meskipun ada baris data yang tidak memenuhi kondisi `JOIN` dengan tabel sebelah kanan.

Example:

1. Menampilkan semua kode mk dan nama mk beserta nilai uas tertinggi dari mata kuliah tersebut beserta matakuliah yang belum memiliki nilai uas.

```
SELECT MK.KODE_MK, MK.NAMA_MK,
nvl(MAX(N.UAS),0)
FROM NILAI N RIGHT OUTER JOIN MATAKULIAH
MK ON MK.KODE_MK = N.KODE_MK
GROUP BY MK.KODE_MK, MK.NAMA_MK;
```

2. Menampilkan semua nid dosen, nama dosen, dan plotting yang sudah dilakukan, beserta dosen-dosen yang belum melakukan plotting.

```
SELECT P.ID_PLOTTING, D.NID, D.NAMA_DOSEN,
P.KODE_MK, P.HARI, P.JAM_MULAI, P.JAM_SELESAI
FROM PLOTTING_AJAR P RIGHT OUTER JOIN DOSEN D
ON D.NID = P.NID;
```

2.3 FULL OUTER JOIN

`FULL OUTER JOIN` akan menampilkan semua baris data dari tabel sebelah kanan dan kiri meskipun ada baris data pada kedua tabel yang tidak memenuhi kondisi `JOIN` antar tabel.

Example:

Menampilkan semua nid dosen, nama dosen, dan plotting yang sudah dilakukan, beserta dosen-dosen yang belum melakukan plotting. Tampilkan juga mata kuliah yang belum di-*plotting* sama sekali.

```
SELECT P.ID_PLOTTING, D.NID, D.NAMA_DOSEN,
MK.KODE_MK, MK.NAMA_MK, P.HARI, P.JAM_MULAI,
P.JAM_SELESAI
FROM PLOTTING_AJAR P FULL OUTER JOIN DOSEN D
ON D.NID = P.NID
FULL OUTER JOIN MATAKULIAH MK ON P.KODE_MK =
MK.KODE_MK;
```

3. *SELF JOIN*

Self join adalah kondisi ketika anda melakukan `JOIN` pada 1 tabel yang sama. Untuk membedakan antara tabel yang satu dengan yang lain,

gunakan *table aliases*. Kondisi yang digunakan pada *self join* tidak harus berupa kondisi *equijoin*.

Example:

1. Menampilkan nim, nama belakang, dan kota_tinggal dari mahasiswa yang tinggal di kota yang sama dengan mahasiswa bernama depan 'Entin'

```
SELECT M1.NIM, M1.NAMA_BELAKANG,
M1.KOTA_TINGGAL
FROM MAHASISWA M1 JOIN MAHASISWA M2
ON M2. NAMA_DEPAN = 'Entin'
WHERE M1.KOTA_TINGGAL = M2.KOTA_TINGGAL;
```

2. Menampilkan seluruh daftar matakuliah beserta matakuliah yang menjadi syarat mata kuliah yang ditampilkan. Tampilkan juga matakuliah yang tidak memiliki mata kuliah syarat

```
select mk.kode_mk, mk.nama_mk, mk.sks,
mk.mk_syarat, mk_s.nama_mk "nama Mk
Syarat", mk_s.sks
from matakuliah mk left outer join
matakuliah mk_s on mk.mk_syarat =
mk_s.kode_mk
```

Practice

***Build The Command*!**

1. Tampilkan jumlah SKS yang sudah didapat oleh mahasiswa bernama 'Anthony'.
2. Tampilkan NIM, nama lengkap mahasiswa, kode_mk, nama_mk, dan nilai akhir untuk mahasiswa angkatan '07' dan tampilkan hanya nilai akhirnya di atas 60 saja.
3. Tampilkan menampilkan jumlah mahasiswa berdasarkan nilai gradenya dan mata kuliah yang ditempuh.

   Penghitungan Grade:

   Nilai <= 69 = D

   70 - 74 = C

   75 - 79 = B

   80 - 84 = B +

   85 - 89 = A

   90- 100 = A+
4. Tampilkan jumlah plotting per dosen untuk hari senin.
5. Tampilkan nid, nama dosen dan jumlah mahasiswa yang pernah diajar.

Stage 5

Joining Multiple Query

"*Uncut Diamond only need enough Pressure to knock of its rough edges and Shine* "

人

Tujuan Hari Ini

Bagaimana menggabungkan 2 (dua) *query* atau lebih

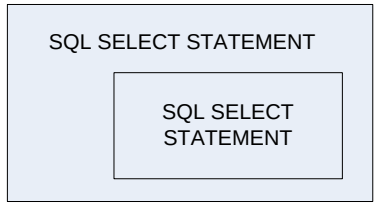
Menggunakan *subquery dan Set Operator*

What We Study

Subquery
Set Operator

1. *SUBQUERY*

Pada *stage* 1 sampai 4, anda dihadapkan dengan pencarian data yang kondisinya masih jelas pada tabel, tetapi tidak menutup kemungkinan bahwa akan muncul kebutuhan data yang kondisinya tidak jelas. Hal ini bisa diselesaikan dengan menggunakan penggabungan 2 (dua) *query* atau lebih. (memasukkan *query* yang 1 (satu) ke *query* yang lain)



Query yang ada di bagian dalam mengembalikan sebuah nilai (atau kumpulan nilai) yang digunakan oleh *query* bagian luar.

Sebuah *subquery* adalah sebuah *query* `SELECT` yang ditancapkan pada sebuah klausa di *query* `SELECT` yang lain. anda dapat meletakkan *subquery* pada beberapa klausa pada SQL *query*, meliputi:

- ✓ Klausa `WHERE`
- ✓ Klausa `HAVING`
- ✓ Klausa `FROM`

Syntax sederhana untuk menuliskan *subquery* adalah sebagai berikut:

```
SELECT SELECT_LIST
FROM TABLE
WHERE EXPR OPERATOR
(SELECT SELECT_LIST FROM TABLE);
```

OPERATOR yang dimaksud pada *Syntax* ini bisa meliputi *single-row operator* (>, =, >=, <, <=, <>) dan *multiple-row operator* (IN, ANY, ALL). *Single-row operator* digunakan pada *subquery* yang mengembalikan 1 baris data (*single-row subquery*), sedangkan *multiple-row operator* digunakan pada *subquery* yang mengembalikan banyak baris data (*multiple-row subquery*). Terdapat juga *subquery* yang mengembalikan banyak baris dan banyak kolom.

Example For *Single-Row Subquery*:

1. Menampilkan nim, nama depan, dan kota_tinggal dari mahasiswa yang tinggal di kota yang sama dengan mahasiswa bernama depan 'Bambang'

```
SELECT NIM, NAMA_DEPAN, KOTA_TINGGAL
FROM MAHASISWA WHERE KOTA_TINGGAL =
(SELECT KOTA_TINGGAL FROM MAHASISWA
WHERE NAMA_DEPAN = 'Bambang');
```

2. menampilkan kode_mk, nama_mk dan nilai uas dari matakuliah yang nilai uas terendahnya lebih tinggi dari nilai uas terendah matakuliah 'Pemrograman Visual I'

```
SELECT MK.KODE_MK, MK.NAMA_MK,
MIN(N.UAS)
FROM NILAI N JOIN MATAKULIAH MK ON
N.KODE_MK = MK.KODE_MK
GROUP BY MK.KODE_MK, MK.NAMA_MK
HAVING MIN(N.UAS) > (SELECT
MIN(NL.UAS)
FROM NILAI NL JOIN MATAKULIAH MK ON
NL.KODE_MK = MK.KODE_MK
WHERE
MK.NAMA_MK = 'Pemrograman Visual II');
```

Example For *Multiple-Row Subquery*:

1. Menampilkan nim, nama lengkap, kode mk, nama mk, dan nilai uas yang dari mahasiswa yang mengambil mata kuliah yang diadakan pada semester 4.

```
SELECT MHS.NIM, MHS.NAMA_DEPAN ||' '||
MHS.NAMA_BELAKANG "NAMA LENGKAP",
N.KODE_MK, MK.NAMA_MK, N.UAS
FROM MAHASISWA MHS JOIN NILAI N
ON MHS.NIM = N.NIM
JOIN MATAKULIAH MK
ON MK.KODE_MK = N.KODE_MK
WHERE N.KODE_MK IN (SELECT KODE_MK FROM
MATAKULIAH WHERE SEMESTER = 4);
```

2. Menampilkan semua nim, nama belakang, dan tanggal lahir mahasiswa yang bulan lahirnya tidak sama dengan bulan lahir mahasiswa prodi SI Desain Komunikasi dan Visual

```
SELECT NIM, NAMA_BELAKANG, TGL_LAHIR
FROM MAHASISWA
WHERE TO_CHAR(TGL_LAHIR, 'MM') <> ALL
(SELECT TO_CHAR(M.TGL_LAHIR, 'MM')
FROM MAHASISWA M JOIN PROGRAM_STUDI P
ON M.KODE_PRODI = P.KODE_PRODI
WHERE P.NAMA_PRODI = 'SI Desain
Komunikasi dan Visual');
```

3. Menampilkan nilai uas tertinggi dari matakuliah yang diadakan pada hari rabu.

```
SELECT mk.kode_mk, mk.nama_mk, max(N.UAS)
FROM matakuliah mk JOIN NILAI N ON
Mk.kode_mk = n.kode_mk
WHERE mk.kode_mk = ANY (SELECT kode_mk FROM
PLOTTING_AJAR WHERE HARI = 'Rabu')
GROUP BY mk.kode_mk, mk.nama_mk;
```

Example For a sub*query* on `FROM` clause:

Menampilkan kode mk dan nama mk yang memiliki rata-rata nilai akhir paling tinggi dari semua matakuliah.

```
SELECT KODE_MK, NAMA_MK, RATA2_NILAI
FROM (SELECT MK.KODE_MK, MK.NAMA_MK,
AVG(((N.TUGAS*0.4)+(N.UAS*0.3)+(N.UTS*0
.3))) AS RATA2_NILAI
FROM NILAI N JOIN MATAKULIAH MK ON
N.KODE_MK = MK.KODE_MK
GROUP BY MK.KODE_MK, MK.NAMA_MK)
where rata2_nilai = (select
max(rata2_nilai) from (SELECT
MK.KODE_MK, MK.NAMA_MK,
AVG(((N.TUGAS*0.4)+(N.UAS*0.3)+(N.UTS*0
.3))) AS RATA2_NILAI
FROM NILAI N JOIN MATAKULIAH MK ON
N.KODE_MK = MK.KODE_MK
GROUP BY MK.KODE_MK, MK.NAMA_MK))
```

2. *Set Operator*

Set operator menggabungkan 2 (dua) atau lebih *query* menjadi 1 (satu) hasil. Semua *set operator* memiliki tingkatan yang sama, jika sebuah *query* memiliki banyak *set operator*, maka Oracle *Server* akan membaca *query* dari kiri (atau atas) ke kanan (bawah), jika tidak ada tanda kurung ( ) yang digunakan untuk mengurutkan.

Operator	Returns
UNION	semua baris berbeda di-SELECT oleh kedua *query*
UNION ALL	semua baris data di-SELECT oleh setiap *query* beserta data duplikat
INTERSECT	semua baris data berbeda yang di-SELECT oleh

Operator	Returns
	kedua query
MINUS	semua baris berbeda yang di-SELECT oleh *query* SELECT yang pertama dan tidak di-SELECT oleh *query* SELECT kedua

Khusus untuk operator `INTERSECT`, anda harus menggunakan tanda kurung ( ) untuk mengurutkan urutan membaca *query* Jika operator `INTERSECT` digunakan bersamaan dengan set operator lain.

2.1 `UNION` *Operator*

Operator `UNION` mengembalikan semua baris data yang di-`SELECT` oleh kedua *query* dan menghilangkan semua baris data kembar.

Example:

Tampilkan id dan nama dari mahasiswa dan dosen yang namanya mengandung kata 'ria'

```
select nim "ID", nama_depan||'
'||nama_belakang "Full Name"
from mahasiswa where
nama_depan||' '||nama_belakang
like '%ria%'
union
select nid, nama_dosen from dosen
where nama_dosen like '%ria%';
```

2.2 `UNION ALL` *Operator*

Operator `UNION ALL` mengembalikan semua baris data yang di-`SELECT` oleh kedua *query* termasuk semua baris data kembar. Sebagai catatan, keyword `DISTINCT` tidak dapat digunakan dengan operator `UNION ALL`..

Example:

Menampilkan semua nid, nama dosen, dan hari dari semua dosen yang sudah melakukan plotting ajar untuk hari senin dan rabu

```
select d.nid, d.nama_dosen, p.hari
from plotting_ajar p join dosen d on
p.nid = d.nid where p.hari = 'Senin'
union all
select d.nid, d.nama_dosen, p.hari
from plotting_ajar p join dosen d on
p.nid = d.nid where p.hari = 'Rabu'
```

2.3 `INTERSECT` *Operator*

Operator `INTERSECT` hanya mengembalikan semua baris data yang kembar saja dari kedua *query*.

Example:

Menampilkan nama dosen yang sudah melakukan plotting ajar

```
SELECT D.NID, D. NAMA_DOSEN
FROM DOSEN D WHERE D.NID IN
(SELECT DS.NID FROM DOSEN DS
INTERSECT
SELECT NID FROM PLOTTING_AJAR);
```

2.4 `MINUS` *Operator*

Operator `MINUS` mengembalikan baris data dari *query* pertama, yang tidak ada di *query* kedua. (hasil *query* pertama – hasil *query* kedua). Jika anda menggunakan klausa `WHERE` pada Operasi `MINUS`, maka semua kolom yang ada di klausa `WHERE` harus dimasukkan ke Klausa `SELECT`.

Example

Menampilkan nid dan nama_dosen yang belum pernah melakukan plotting ajar

```
select nid, nama_dosen from dosen
where nid in
(select nid from dosen
minus
select nid from plotting_ajar);
```

Practice

***Build The Command*!**

1. Menampilkan nim, nama lengkap, dan alamat dari mahasiswa yang lahir pada tahun yang sama dengan mahasiswa dengan nama depan 'Abe'
2. Menampilkan nama kegiatan dan jumlah skkm total berdasarkan posisi yang didapatkan pada sebuah kegiatan, dan tampilkan hanya yang jumlah skkm nya terbesar dari total kegiatan yang lain.
3. Menampilkan Jumlah SKS kurang milik mahasiswa bernama depan 'Desi'. SKS kurang didapat dari jumlah total sks semua matakuliah dikurangi jumlah SKS yang sudah diambil.
4. Menampilkan kode MK dan Nama MK yang belum memiliki nilai.
5. Tampilkan nim, nama mahasiswa, dan jumlah sks/skkm yang jumlah sks yang sudah ditempuh dan jumlah skkm yang dimilikinya sama.

Stage 6
Data Manipulation Language (DML)
and Transactions

"***Fight for what you believe, not what you're told!***"

人

Tujuan Hari Ini

Menggunakan DML untuk memanipulasi data

Membatalkan atau menyimpan *transaction* yang sudah dilakukan

What We Study

Data Manipulation Language

Transaction Controls

1. *Data Manipulation Language* (DML)

Data Manipulation Language (DML) adalah sebuah bagian inti dari SQL. Ketika anda ingin menambahkan, mengubah, atau menghapus data dari *database*, anda menjalankan sebuah *query* DML. Sekumpulan *query* DML yang membentuk sebuah unit logis dari sebuah pekerjaan disebut sebuah *transaction.*

1.1 Menambahkan baris data baru ke dalam tabel

Untuk menambahkan baris data baru ke dalam tabel, anda menggunakan DML `INSERT`. *Syntax* dasar `INSERT`:

```
INSERT INTO TABLE [(COLUMN[, COLUMN. . .])]
VALUES (VALUE[, VALUE. . .]);
```

Keterangan:

- ✓ `TABLE` adalah nama dari tabel yang akan diisi baris data baru.
- ✓ `COLUMN` adalah nama kolom pada tabel yang akan diisi. Kolom tidak harus dituliskan jika anda akan mengisi semua kolom yang ada pada tabel tersebut.
- ✓ `VALUE` adalah nilai yang sesuai dengan kolom. Penulisan nilai harus sesuai dengan urutan kolom pada tabel.
- ✓ *Syntax* ini hanya memasukkan satu baris data saja ke dalam tabel.

Example:

1. Memasukkan data mahasiswa baru.

```
INSERT INTO MAHASISWA
VALUES('06390200123','39020', 'Adr', 'Wjy',
'Jl. S 1/4, S', 'Sidoarjo', '0874451234',
'P', 'B', 'A', '16-JAN-1988');
```

2. Memasukkan data kegiatan baru.

```
INSERT INTO KEGIATAN (ID_EVENT, NAMA_EVENT,
TANGGAL_EVENT)
VALUES ('EC020','CARDFIGHT!! VANGUARD
NATIONAL TOURNAMENT','10-JAN-2013');
```

3. Memasukkan data mahasiswa dengan nama belakang dibiarkan kosong

```
INSERT INTO MAHASISWA (NIM,
KODE_PRODI, NAMA_DEPAN,
NAMA_BELAKANG, JENIS_KELAMIN,
STATUS_NIKAH, STATUS_MHS)
VALUES ('08410150023', '41015',
'Rendy', NULL, 'P', 'B', 'A');
```

4. memasukkan jenis posisi kegiatan berdasarkan inputan dari user

.

```
INSERT INTO JENIS_POS_EVENT
VALUES (&ID_POSISI, &NAMA_POSISI);
```

Anda dapat menggunakan `INSERT` untuk menambahkan baris data ke dalam sebuah tabel yang nilai pada baris datanya diambil dari tabel lain. untuk melakukan ini, bagian `VALUES` digantikan dengan *subquery*.

Cat:

- ✓ Jumlah kolom pada klausa INSERT harus sesuai dengan pada sub*query*, jika kolom tidak disebutkan pada klausa INSERT, maka jumlah kolom pada sub*query* harus sesuai dengan kolom pada tabel.
- ✓ Dengan cara ini, anda dapat memasukkan banyak baris data ke dalam sebuah tabel jika hasil sub*query* yang dikembalikan lebih dari 1 baris data.

Example:

Memasukkan data mahasiswa yang nama depannya 'Anthony' ke dalam mahasiswa berprestasi.

```
INSERT INTO MAHASISWA_BERPRESTASI
SELECT NIM, NAMA_DEPAN || NAMA_BELAKANG,
NO_TELP_AKTIF, TO_CHAR(SYSDATE, 'YYYY')
FROM MAHASISWA
WHERE NAMA_DEPAN = 'Anthony';
```

Anda juga dapat menggunakan *subquery* untuk menggantikan nama tabel dalam *query* `INSERT`. Jika anda menggunakan cara ini, maka `SELECT` pada *subquery* harus memiliki kolom yang sesuai dengan kolom yang diisi pada klausa `VALUES`.

Example:

Memasukkan data matakuliah baru.

```
INSERT INTO (SELECT KODE_MK, NAMA_MK,
SKS, SEMESTER, GRADE_MINIMUM FROM
MATAKULIAH)
VALUES ('M-080', 'CARD CREATION', 2,
1, 'B');
```

1.2 Mengubah baris data yang sudah ada

Untuk mengubah baris data yang sudah ada di tabel, anda menggunakan DML `UPDATE`. *Syntax* dasar `UPDATE`:

```
UPDATE TABLE
SET COLUMN = VALUE[, COLUMN = VALUE,. . .]
[WHERE CONDITION];
```

Keterangan:

- ✓ `TABLE` adalah nama dari tabel yang akan diubah datanya.
- ✓ `COLUMN` adalah nama kolom yang akan dirubah nilainya.
- ✓ `VALUE` adalah nilai atau sub*query* yang sesuai dengan kolom
- ✓ `CONDITION` adalah kondisi yang menyebutkan baris data mana yang akan dirubah. Umumnya, primary key digunakan sebagai dasar untuk merubah 1 (satu) kolom, karena jika tidak, maka ada kemungkinan banyak baris data yang akan berubah baris datanya. Jika `CONDITION` tidak dituliskan, maka *query* `UPDATE` akan mengubah semua baris data pada tabel.

Example:

1. Mengubah nama belakang dan kota tinggal dari mahasiswa dengan nama ‘Henny' menjadi ‘Wijaya’ dan ‘Sidoarjo’

```
UPDATE MAHASISWA
SET NAMA_BELAKANG = 'Wijaya',
KOTA_TINGGAL = 'Sidoarjo'
WHERE NAMA_DEPAN = 'Henny';
```

2. Mengubah status semua mahasiswa menjadi ‘T’ (‘Tidak Aktif’)

```
UPDATE MAHASISWA
SET STATUS_MHS = 'T';
```

3. Mengubah data plotting dosen yang id_plottingnya ‘P1005’. Ganti NID nya dengan NID dari ‘David’ dan Kode MK nya dari kode mk matkul ‘Statistika’

```
UPDATE PLOTTING_AJAR
SET NID = (SELECT NID FROM DOSEN
           WHERE NAMA_DOSEN = 'David'),
KODE_MK = (SELECT KODE_MK FROM MATAKULIAH
           WHERE NAMA_MK = 'Statistika')
WHERE ID_PLOTTING = 'P1005';
```

1.3 Menghapus baris data

Untuk menghapus baris data yang sudah ada pada tabel, anda menggunakan DML `DELETE`. *Syntax* Dasar `DELETE`:

```
DELETE [FROM] TABLE
[WHERE CONDITION];
```

Keterangan:

- ✓ TABLE adalah nama dari tabel yang akan dihapus datanya.
- ✓ CONDITION adalah kondisi yang menyebutkan baris data mana yang akan dirubah. Jika tidak ada kondisi yang dituliskan, maka *query* DELETE akan menghapus semua baris data pada tabel.

Example:

1. Menghapus data nilai mahasiswa bagi mahasiswa dengan nim '06390200123' dan kode mk 'M-040'

```
DELETE FROM NILAI
WHERE NIM = '06390200123' AND
KODE_MK = 'M_040';
```

2. Menghapus semua data mahasiswa berprestasi.

```
DELETE MAHASISWA_BERPRESTASI;
```

3. Menghapus data nilai mahasiswa yang nilai UAS-nya paling rendah diantara semuanya

```
Delete from nilai
Where uas =
(select min(uas) from nilai);
```

2. *Transaction Control*

Oracle Server menjamin konsistensi data berdasarkan *transactions*. Transaction dapat terdiri dari:

Tipe	Keterangan
Data Manipulation Language (DML)	terdiri atas sejumlah *query* DML yang dianggap *Oracle Server* sebagai sebuah entitas tunggal atau sebuah unit kerja logis
Data Definition Language (DDL)	terdiri atas 1 (satu) *query* DDL
Data Control Language (DCL)	terdiri atas 1 (satu) *query* DCL

Transaction dimulai ketika sebuah *query* DML dijalankan, dan selesai ketika:

- ✓ *Query* `COMMIT` atau `ROLLBACK` dijalankan.
- ✓ Sebuah *query* DDL atau DCL dijalankan.
- ✓ Sebuah *query* DCL dijalankan.
- ✓ Terjadi *crash* pada mesin.

Anda dapa mengendalikan Transaction dengan menggunakan `COMMIT`, `SAVEPOINT`, `ROLLBACK`.

Tipe	Keterangan
COMMIT	menghentikan Transaction sekarang, dan membuat semua perubahan data sementara menjadi permanen
SAVEPOINT *Name*	menandai sebuah *savepoint* dalam *Transaction* sekarang
ROLLBACK	menghentikan *transaction* sekarang dengan menghilangkan semua perubahan data sementara
ROLLBACK TO SAVEPOINT *Name*	ROLLBACK TO SAVEPOINT *Name* mengembalikan posisi *Transaction* sekarang ke SAVEPOINT tertentu, sehingga menghilangkan semua perubahan data sementara dan/atau SAVEPOINT yang dibuat setelah SAVEPOINT yang dijadikan dasar ROLLBACK. Jika klausa TO SAVEPOINT dihilangkan, maka ROLLBACK akan menghilangkan semua perubahan data sementara.

Kondisi data sebelum COMMIT atau ROLLBACK:

- ✓ Kondisi data sebelumnya dapat dikembalikan
- ✓ *User* sekarang mampu melihat hasil DML dengan menggunakan *query* SELECT
- ✓ *User* lain tidak dapat melihat hasil DML yang dijalankan *user* sekarang.
- ✓ Baris data yang dipengaruhi DML "dikunci", sehingga *user* lain tidak dapat mengganti nilai pada baris data tersebut.

Kondisi data setelah `COMMIT`:

- ✓ Perubahan data dibuat menjadi permanen dalam *database*.
- ✓ Kondisi data yang lama hilang.
- ✓ Semua *user* bisa melihat hasil DML.
- ✓ Kunci dari baris data akan dilepas, sehingga dapat dimanipulasi oleh *user* lain
- ✓ Semua SAVEPOINT dihapus.

Example:

1. Menambahkan data baru ke dalam tabel matakuliah dan menyimpan tambahan data tersebut.

```
INSERT INTO MATAKULIAH VALUES ('M-090',
'BAHASA JEPANG DASAR', '2','2','-');

COMMIT;
```

2. Tambahkan Data Baru ke dalam tabel Prodi lalu lakukan `ROLLBACK`. Lakukan `SELECT` untuk membuktikan bahwa data program studi tidak berubah sama sekali.

```
INSERT INTO PROGRAM_STUDI
VALUES ('41030','Card Research', 4);

ROLLBACK;

SELECT * FROM PROGRAM_STUDI;
```

3. Tambahkan 2 data baru ke dalam tabel matakuliah lalu buat sebuah SAVEPOINT dengan nama 'one', lalu ganti salah satu nama matakuliah yang dimasukkan dengan nama 'Struktur Data'. lalu lakukan ROLLBACK ke SAVEPOINT yang dibuat. Lakukan SELECT untuk membuktikan bahwa perubahan sudah dibatalkan.

```
INSERT INTO MATAKULIAH VALUES
('M-023', 'JARKOM', 3,4, 'B');
INSERT INTO MATAKULIAH VALUES
('M-034', 'SISPAK', 3, 4, '-');

SAVEPOINT ONE;

UPDATE MATAKULIAH SET NAMA_MK =
'STRUKTUR DATA' WHERE KODE_MK =
'M-034';
ROLLBACK TO SAVEPOINT ONE;
SELECT * FROM MATAKULIAH;
```

Practice

***Build The Command*!**

1. Masukkan data kegiatan baru, dengan ketentuan tanggal kegiatan adalah hari ini.
2. Masukkan Data Berikut ke dalam tabel Plotting Ajar:
   ID_PLOTTING = P0001
   NID = DN-001
   KODE MK = M-100
   JAM MULAI = 08.00
   JAM SELESAI = 10.00
   HARI = SELASA
3. Ganti NID pada data plotting yang dimasukkan dengan NID yang dimiliki oleh Budi.
4. Masukkan data mahasiswa ke dalam mahasiswa berprestasi, hanya yang mahasiswa yang memiliki nilai akhir pada matakuliah diatas 75.
5. Hapus seluruh data mahasiswa yang ada di tabel mahasiswa berprestasi.
6. lakukan ROLLBACK dari semua transaksi yang dilakukan.

Stage 7

Data Definition Language (DDL), *Table, and View*

"*Whether to Rise again after every fall or stay mourning in the darkness, the choice is yours.*"

人

Tujuan Hari Ini

Menggunakan DDL untuk membuat, mengubah, dan menghapus Table dan View

What We Study

Table

View

1. *Table*

1.1 *Simple Table Creation*

Sebuah Oracle *Database* dapat menampung banyak struktur data. setiap struktur harus diuraikan dalam desain *database*, sehingga dapat dibangun pada saat tahap pembangunan dari pengembangan *database*. salah satu dari struktur tersebut adalah tabel yang digunakan untuk menyimpan data.

Pada saat membuat tabel, terdapat peraturan penamaan yang harus dipatuhi dalam membuat nama tabel atau nama kolom.

- ✓ Nama tabel atau kolom harus dimulai dengan huruf dan panjang nama kolom antara 1 sampai 30 karakter.
- ✓ Nama tabel atau kolom hanya mengandung A-Z, a-z, 0-9, _, $, dan #.
- ✓ Nama tabel atau kolom tidak boleh kembar dengan nama dari obyek lain yang dimiliki oleh *user* yang sama.
- ✓ Nama tabel atau kolom tidak boleh menggunakan kata-kata yang sudah digunakan oleh Oracle *Server*.

Untuk membuat tabel, digunakan query `CREATE TABLE`. Selain itu seorang user harus memiliki `CREATE TABLE` *privilege* dan sebuah area penyimpanan untuk menyimpan obyek yang dibuat.

Syntax dasar membuat tabel:

```
CREATE TABLE [SCHEMA.]TABLE
(COLUMN DATATYPE [DEFAULT EXPR] [, . . .]);
```

Keterangan:

- ✓ SCHEMA: nama dari SCHEMA database.
- ✓ *TABLE*: nama dari tabel.
- ✓ DEFAULT *EXPR*: sebuah nilai *default* jika nilai dari kolom dihilangkan dalam query INSERT.
- ✓ *COLUMN*: nama dari kolom.
- ✓ *DATATYPE*: tipe data dan panjang dari kolom.

Ketika anda membuat sebuah tabel, anda bisa menentukan agar sebuah kolom diberikan sebuah nilai *default* dengan menggunakan opsi DEFAULT. Opsi ini mencegah agar kolom ini tidak diisi dengan nilai NULL jika sebuah baris dimasukkan tanpa nilai untuk kolom tersebut.

Cat:

- ✓ Yang bisa digunakan sebagai DEFAULT adalah nilai pasti, ekspresi, atau SQL *Function*. Kolom dari tabel lain atau sebuah *pseudocolumn* tidak bisa digunakan.
- ✓ Tipe data dari nilai pada DEFAULT harus sesuai dengan tipe data kolom.

Example:

1. Membuat tiruan tabel Plotting_ajar dengan nama Plot_Mengajar.

```
CREATE TABLE PLOT_MENGAJAR
(ID VARCHAR2(5), KODE_MK VARCHAR2(5),
NID VARCHAR2(6),JAM_MULAI VARCHAR2(5),
JAM_SELESAI VARCHAR2(5), HARI VARCHAR(6));
```

2. Membuat tiruan tabel Matakuliah dengan nama MK, dengan memberikan nilai DEFAULT '3' pada kolom SKS, dan nilai DEFAULT '-' pada kolom Min_Grade.

```
CREATE TABLE MK
(ID_MK VARCHAR2(5), NAMA_MK VARCHAR2(30),
SKS NUMBER(1) DEFAULT 3,
SEMESTER NUMBER(1),
MIN_GRADE VARCHAR2(1) DEFAULT '-');
```

1.2 CONSTRAINT

Anda dapat menggunakan `CONSTRAINT` untuk melakukan hal-hal berikut:

- ✓ Memasukkan aturan pada data dalam sebuah tabel ketika sebuah baris ditambahkan, diubah, atau dihapus dari tabel tersebut. Aturan `CONSTRAINT` harus dipenuhi agar *query* bisa berhasil dijalankan.
- ✓ Mencegah penghapusan data dari sebuah tabel jika terdapat *dependency*

Tipe –tipe `CONSTRAINT`:

Constraint	Description
`NOT NULL`	Menentukan sebuah kolom tidak boleh diisi nilai NULL
`UNIQUE`	Menentukan 1 (satu) atau banyak kolom tidak boleh bernilai kembar antara 1 (baris) dengan yang lain

Constraint	Description
PRIMARY KEY	Menentukan agar sebuah kolom menjadi dasar identifikasi yang tidak kembar dari setiap baris dari tabel (NOT NULL + UNIQUE)
FOREIGN KEY	Menghubungkan sebuah kolom dengan kolom dari tabel lain yang direferensi
CHECK	Menentukan kondisi yang harus dipenuhi

Keterangan:

- ✓ Anda dapat memberi nama `CONSTRAINT`, atau *Oracle Server* akan mengeluarkan sebuah nama dengan menggunakan `SYS_Cn` *format*. `CONSTRAINT` tidak boleh memiliki nama yang sama dengan `CONSTRAINT` yang sudah ada pada database.
- ✓ `CONSTRAINT` dapat dibuat pada saat bersamaan dengan pembuatan tabel atau setelah tabel dibuat.
- ✓ `CONSTRAINT` dapat dibuat pada level kolom atau level tabel. `CONSTRAINT` yang dibuat pada level kolom dituliskan pada saat kolom dibuat. `CONSTRAINT` yang dibuat pada level tabel dituliskan pada bagian akhir dari definisi tabel dan harus mereferensi ke kolom
- ✓ `CONSTRAINT NOT NULL` harus dituliskan pada level kolom

Syntax membuat `CONSTRAINT`:

```
CREATE TABLE [SCHEMA.]TABLE
(COLUMN DATATYPE [DEFAULT EXPR]
[COLUMN_CONSTRAINT],
 . . .
[TABLE_CONSTRAINT] [, . . .]);
```

Example:

1. Membuat tabel tiruan tabel Matakuliah dengan nama MK dan membuat agar kolom ID dan nama MK tidak boleh kosong.

```
CREATE TABLE MK_2
(ID_MK VARCHAR2(5) NOT NULL,
NAMA_MK VARCHAR2(30) NOT NULL,
SKS NUMBER(1) DEFAULT 3,
SEMESTER NUMBER(1),
MIN_GRADE VARCHAR2(1) DEFAULT '-');
```

2. Membuat tabel mahasiswa yang kolom e-mail-nya tidak boleh kembar, tetapi boleh kosong.

Level Kolom:

```
CREATE TABLE MHS (NIM CHAR(11),
NAMA_LENGKAP   VARCHAR2(50)   NOT   NULL,
EMAIL VARCHAR2(25)
CONSTRAINT MHS_MAIL_UK UNIQUE,
NO_TELP VARCHAR2(15) NOT NULL);
```

Level Tabel:

```
CREATE TABLE MHS_T (NIM CHAR(11),
NAMA_LENGKAP VARCHAR2(50) NOT NULL,
EMAIL VARCHAR2(25),
NO_TELP VARCHAR2(15) NOT NULL,
CONSTRAINT MHS_MAIL_UK_T UNIQUE(EMAIL));
```

3. Membuat tabel plotting ajar dengan ID_Plot sebagai Primary Key.

Level Kolom:

```
CREATE TABLE PLOT_MENGAJAR_K
(ID VARCHAR2(5)
CONSTRAINT PK_PLOT PRIMARY KEY,
KODE_MK VARCHAR2(5) NOT NULL,
NID VARCHAR2(6) NOT NULL,
JAM_MULAI VARCHAR2(5),
JAM_SELESAI VARCHAR2(5),
HARI VARCHAR(6) NOT NULL);
```

Level Table:

```
CREATE TABLE PLOT_MENGAJAR_T
(ID VARCHAR2(5),
KODE_MK VARCHAR2(5) NOT NULL,
NID VARCHAR2(6) NOT NULL,
JAM_MULAI VARCHAR2(5),
JAM_SELESAI VARCHAR2(5),
HARI VARCHAR(6) NOT NULL,
CONSTRAINT PK_PLOT_T PRIMARY
KEY(ID));
```

4. Membuat tiruan tabel dari tabel nilai dengan nim, kode_mk, dan nid sebagai primary key

```
CREATE TABLE NILAI_MAHASISWA
(NIM CHAR(11), KODE_MK VARCHAR2(5),
NID VARCHAR2(6), N_TUGAS NUMBER(3),
N_UTS NUMBER(3), N_UAS NUMBER(3),
CONSTRAINT PK_NL PRIMARY KEY(NIM,
NID, KODE_MK));
```

5. membuat tiruan tabel Plot mengajar dengan foreign key ke tabel mahasiswa, dosen, dan matakuliah.

Level Kolom:

```
CREATE TABLE PLOT_MENGAJAR_2
(ID_PLOT VARCHAR2(8) CONSTRAINT PK_PLOT_2
PRIMARY KEY,
KODE_MK VARCHAR2(5)
CONSTRAINT FK_MK REFERENCES
MATAKULIAH(KODE_MK) NOT NULL,
NID VARCHAR2(6)
CONSTRAINT FK_DSN REFERENCES DOSEN(NID) NOT
NULL,
N_TUGAS NUMBER(3,1),
N_UTS NUMBER(3,1),N_UAS NUMBER(3,1));
```

Level Table:

```
CREATE TABLE PLOT_MENGAJAR_3
(ID_PLOT VARCHAR2(8),
KODE_MK VARCHAR2(5) NOT NULL,
NID VARCHAR2(6) NOT NULL,
N_TUGAS NUMBER(3,1), N_UTS NUMBER(3,1),
N_UAS NUMBER(3,1),
CONSTRAINT PK_PLOT_3 PRIMARY KEY(ID_PLOT),
CONSTRAINT FK_MK_3 FOREIGN KEY(KODE_MK)
REFERENCES MATAKULIAH(KODE_MK),
CONSTRAINT FK_DSN_3 FOREIGN KEY(NID)
REFERENCES DOSEN(NID));
```

Kolom dengan Foreign Key Constraint hanya bisa diisi dengan nilai yang ada pada kolom yang direferensi atau NULL.

6. membuat tiruan tabel nilai yang yang kolom tugas, uts, dan uas nya hanya bisa diisi antara angka 0 sampai 100

```
CREATE TABLE NILAI_MAHASISWA_2
(NIM CHAR(11)
CONSTRAINT FK_MHS REFERENCES MAHASISWA(NIM),
KODE_MK VARCHAR2(5)
CONSTRAINT FK_MK_2 REFERENCES
MATAKULIAH(KODE_MK),
NID VARCHAR2(6)
CONSTRAINT FK_DSN_2 REFERENCES DOSEN(NID),
N_TUGAS NUMBER(3)
CONSTRAINT CK_TGS CHECK(N_TUGAS BETWEEN 0 AND
100),
N_UTS NUMBER(3)
CONSTRAINT CK_UTS CHECK(N_UTS BETWEEN 0 AND
100),
N_UAS NUMBER(3)
CONSTRAINT CK_UAS CHECK(N_UAS BETWEEN 0 AND
100),
CONSTRAINT PK_NL_2 PRIMARY KEY(NIM, NID,
KODE_MK));
```

Selain semua cara di atas, anda juga dapat membuat tabel dengan memanfaatkan *subquery*.

Example:

Membuat tiruan tabel mahasiswa dengan ketentuan hanya mengambil kolom nim, nama lengkap (nama depan + nama belakang), dan alamat tinggal hanya untuk mahasiswa yang tinggal di kota 'Surabaya'.

Cara 1 (Satu):

```
CREATE TABLE COPY_MHS AS
SELECT NIM, NAMA_DEPAN || ' ' ||
NAMA_BELAKANG AS NAMA_LENGKAP,
ALAMAT_TINGGAL
FROM MAHASISWA WHERE KOTA_TINGGAL =
'Surabaya';
```

Cara 2 (Dua):

```
CREATE TABLE COPY_MHS_2
(NIM CONSTRAINT PK_CP PRIMARY KEY,
NAMA_LENGKAP NOT NULL,
ALAMAT NOT NULL) AS
SELECT NIM, NAMA_DEPAN || ' ' ||
NAMA_BELAKANG, ALAMAT_TINGGAL
FROM MAHASISWA WHERE KOTA_TINGGAL =
'Surabaya';
```

1.3 ALTER TABLE & DROP TABLE

Setelah anda membuat sebuah tabel, mungkin anda perlu struktur dari tabel yang sudah anda buat. Untuk merubah struktur dari tabel yang sudah ada, anda menggunakan query ALTER TABLE. Query ALTER TABLE dapat digunakan untuk:

a. Menambahkan kolom atau constraint baru
b. Mengubah kolom atau constraint yang sudah ada
c. Mendefinisikan sebuah nilai default untuk kolom
d. Menghapus sebuah kolom.

Syntax dasar untuk `ALTER TABLE`:

✓ Untuk menambahkan kolom baru

```
ALTER TABLE TABLE
ADD (COLUMN DATATYPE [DEFAULT EXPR]
[, COLUMN DATATYPE]...);
```

✓ Untuk mengubah kolom yang sudah ada

```
ALTER TABLE TABLE
MODIFY (COLUMN DATATYPE [DEFAULT EXPR]
[, COLUMN DATATYPE]...);
```

✓ Untuk menghapus kolom

```
ALTER TABLE TABLE
DROP (COLUMN);
```

✓ Untuk menambahkan `CONSTRAINT` baru.

```
ALTER TABLE <TABLE_NAME>
ADD [CONSTRAINT <CONSTRAINT_NAME>] TYPE
(<COLUMN NAME>);
```

✓ Untuk menghapus CONSTRAINT yang sudah ada

```
ALTER TABLE TABLE
DROP PRIMARY KEY |UNIQUE (COLUMN)|
CONSTRAINT CONSTRAINT [CASCADE];
```

Example:

1. Menambahkan kolom q_pinjam ke tabel mahasiswa yang sudah dengan nilai default 5.

```
ALTER TABLE MAHASISWA
ADD (Q_PINJAM NUMBER(2) DEFAULT 5 NOT NULL);
```

2. Mengubah panjang kolom kode_mk dari tabel matakuliah menjadi 7 digit.

```
ALTER TABLE MATAKULIAH
MODIFY (KODE_MK VARCHAR2(7));
```

3. Menghapus kolom quota perpus yang sudah ditambahkan ke tabel mahasiswa.

```
ALTER TABLE MAHASISWA
DROP (Q_PINJAM);
```

4. Menambahkan `CONSTRAINT CHECK` ke kolom STATUS_MHS. Pada tabel mahasiswa

```
ALTER TABLE MAHASISWA
ADD CONSTRAINT CK_STSM CHECK
(STATUS_MHS IN ('A','T'));
```

5. Menghapus Constraint check yang ditambahkan.ke tabel mahasiswa

```
ALTER TABLE MAHASISWA
DROP CONSTRAINT CK_STSM;
```

Untuk menghapus tabel yang sudah dibuat, gunakan query `DROP TABLE`. Pada saat query `DROP TABLE` dijalankan, maka:

- ✓ Semua data dan struktur dari tabel dihapus
- ✓ Semua *transaction* akan langsung `COMMIT`
- ✓ Semua `INDEX` dihapus
- ✓ Semua `CONSTRAINT` dihapus
- ✓ Anda tidak bisa melakukan `ROLLBACK` pada query `DROP TABLE`.

Syntax untuk melakukan `DROP TABLE`:

```
DROP TABLE TABLE
```

Example:

Menghapus tabel mahasiswa berprestasi

```
DROP TABLE MAHASISWA_BERPRESTASI;
```

2. VIEW

Anda dapat menampilkan kombinasi data atau sebagian data dengan membuat VIEW dari tabel. sebuah VIEW adalah sebuah tabel logis berdasarkan sebuah tabel(atau banyak tabel) atau VIEW lain. sebuah VIEW tidak menampung data miliknya sendiri tetapi lebih seperti sebuah jendela untuk merubah atau melihat data dari tabel-tabel lain. tabel yang dijadikan dasar oleh sebuah VIEW disebut sebagai *base table*.

Keuntungan dari VIEW:

- ✓ VIEW membatasi akses ke data karena VIEW dapat menampilkan kolom yang dipilih dari tabel.
- ✓ VIEW dapat digunakan untuk membuat query sederhana untuk mengambil hasil dari query yang rumit
- ✓ VIEW menyediakan kebebasan data untuk *ad hoc user* dan program aplikasi.
- ✓ VIEW menyediakan kelompok *user* akses ke data berdasarkan kriteria tertentu masing-masing.

Terdapat 2 (dua) klasifikasi untuk VIEW: *simple* dan *complex*. Perbedaan yang paling mendasar dari 2 (dua) klasifikasi ini berhubungan dengan DML.

Sifat	*Simple Views*	*Complex Views*
Jumlah Tabel	1 (satu)	1 (satu) atau lebih
Menampung Function	tidak	ya
Menampung Data Grup	tidak	ya
Operasi DML pada VIEW	ya	tidak selalu

Syntax untuk membuat VIEW:

```
CREATE [OR REPLACE] [FORCE|NOFORCE] VIEW
view [(alias[, alias]...)]
AS subquery
[WITH CHECK OPTION [CONSTRAINT constraint]]
[WITH READ ONLY [CONSTRAINT constraint]];
```

Keterangan:

- ✓ `OR REPLACE`: membuat ulang `VIEW` jika sudah ada.
- ✓ `FORCE`: membuat `VIEW` meskipun *base table*-nya tidak ada.
- ✓ `NOFORCE`: membuat `VIEW` hanya jika *base table*-nya ada. Ini adalah default pada saat membuat `VIEW`.
- ✓ *`view`*: adalah nama dari `VIEW`.
- ✓ *`alias`*: nama *alias* yang akan digunakan untuk setiap kolom pada *query*.
- ✓ Subquery: *query* `SELECT` yang menentukan isi dari `VIEW`.
- ✓ `WITH CHECK OPTION`: `CONSTRAINT` untuk menentukan agar hanya baris data yang bisa diakses oleh VIEW saja, yang bisa dimasukkan atau diubah.
- ✓ *`Constraint`*: nama yang diberikan pada constraint `CHECK OPTION`
- ✓ `WITH READ ONLY`: memastikan agar tidak ada operasi DML bisa dilakukan pada `VIEW`.

Example:

1. Membuat VIEW untuk menampilkan NIM, nama lengkap, dan status mahasiswa dari mahasiswa yang nama prodi-nya 'D3 Game Design' .

```
CREATE OR REPLACE VIEW VIEW_MHS_D3GM
(NIM, NAMA_LENGKAP, STS_KULIAH)
AS SELECT M.NIM, M.NAMA_DEPAN || ' ' ||
M.NAMA_BELAKANG, M.STATUS_MHS FROM MAHASISWA
M, PROGRAM_STUDI P WHERE M.KODE_PRODI =
P.KODE_PRODI AND
P.NAMA_PRODI = 'D3 Game Design';
```

2. Membuat VIEW untuk menampilkan semua nama dosen, nama matakuliah, dan waktu mengajar untuk matakuliah semester 3 .

```
CREATE OR REPLACE VIEW VIEW_AJAR_SMT3
AS SELECT D.NAMA_DOSEN, MK.NAMA_MK,
P.HARI, P.JAM_MULAI, P.JAM_SELESAI
FROM MATAKULIAH MK JOIN PLOTTING_AJAR P
ON MK.KODE_MK = P.KODE_MK
JOIN DOSEN D
ON P.NID = D.NID
WHERE MK.SEMESTER = 3;
```

3. Membuat VIEW untuk menampilkan kode_mk, nama_mk, dan jumlah dosen per mk.

```
CREATE OR REPLACE VIEW VIEW_MK_DOSEN
(KODE_MK, NAMA_MK, JUMLAH_DOSEN)
AS
SELECT MK.KODE_MK, MK.NAMA_MK, COUNT(N.NID)
FROM MATAKULIAH MK, NILAI N
WHERE MK.KODE_MK = N.KODE_MK
GROUP BY MK.KODE_MK, MK.NAMA_MK;
```

Untuk mengubah VIEW yang sudah dibuat, dapat dengan menggunakan *keyword* OR REPLACE untuk menindih VIEW yang sudah dibuat dengan VIEW yang baru.

Anda dapat melakukan operasi DML pada *simple* VIEW, sedangkan anda tidak bisa melakukan operasi DML jika sebuah View memiliki:

- ✓ Group function
- ✓ Klausa GROUP BY
- ✓ Keyword DISTINCT
- ✓ Keyword ROWNUM
- ✓ Kolom yang didefinisikan dari ekspresi.
- ✓ Kolom NOT NULL pada base table, yang tidak di-SELECT pada VIEW

Untuk menghapus sebuah VIEW, gunakan query DROP VIEW.

```
DROP VIEW VIEW;
```

Example:

Menghapus VIEW_AJAR_SMT3

```
DROP VIEW VIEW_AJAR_SMT3;
```

Practice

***Build The Command*!**

1. Buatlah sebuah tabel dengan keterangan sebagai berikut:

Nama Tabel: Jenis_Koleksi		
Nama Kolom	**Tipe Data**	**Constraint**
id_jenis	number(1)	primary key
nama_jenis	varchar2(30)	not null

Nama Tabel: Koleksi_Perpustakaan		
Nama Kolom	**Tipe Data**	**Constraint**
id_koleksi	varchar2(8)	primary key
Judul_Koleksi	varchar2(50)	not null
id_jenis	number(1)	Foreign Key (Jenis_Koleksi)
Tahun_Terbit	number(4)	not null
Pengarang	varchar2(50)	not null
Bahasa	varchar2(20)	-
Sts_Koleksi	char(1)	Check (A/K)

2. Tambahkan Constraint Check pada kolom Tahun_Terbit di tabel koleksi_perpustakaan agar panjang nilai yang dimasukkan harus 4.
3. Buatlah view untuk menampilkan nama event, kapan diadakan, dan jumlah mahasiswa yang ikut pada event tersebut.

4. Buatlah View untuk menampilkan data semua mahasiswa dan jumlah SKKM yang sudah didapat dari mengikuti kegiatan. Tampilkan juga yang belum mendapatkan SKKM.

Stage 8

A Moment Before Final

"*Maybe it's not about the distance, it's about how we walk the road.*"

人

Tujuan Hari Ini

Mereview semua stage yang sudah dilewati

(Pre Ujian Praktikum)

Practice

***Build The Command*!**

1. Buat tabel dengan nama Role, User, dan Status yang diikuti dengan nim pendek anda. Struktur setiap tabel adalah sebagai berikut:

Tabel: Role

Nama Kolom	Tipe Data	Constraint
Role_ID	Number (2)	Primary Key
Role_Name	Varchar2 (30)	Not Null

Tabel: Status

Nama Kolom	Tipe Data	Constraint
Sts	Char (1)	Primary Key
Keterangan	Varchar2 (50)	Not Null

Tabel: User

Nama Kolom	Tipe Data	Constraint
UserID	Varchar2 (10)	Primary Key
UserName	Varchar2 (50)	Not Null
Email	Varchar2 (50)	Not Null + Unique + Contains (‘@’, ‘.’)
Role_ID	Number (2)	Foreign Key dari tabel role
Sts	Char (1)	Foreign Key dari tabel status

2. Masukkan data berikut ke tabel role dan tabel status

Tabel: Role

Role_ID	Role_Name
10	Admin
20	Administrative
30	Guest

Tabel: Status

Sts	Keterangan
A	Active User
B	Banned User

3. Masukkan data Ke dalam tabel user dengan mengopi data dari tabel-tabel pada database hr ,dengan ketentuan sebagai berikut:
   - ✓ UserID = Employee_ID + 2 Digit Nama Belakang dari Kiri
   - ✓ UserName = Nama Lengkap Karyawan (First name + ' ' + Last name)
   - ✓ Email = 2 Digit Nama Depan dari kiri + 1 Digit Nama Belakang dari Kanan + Country ID + '@mail.com'
   - ✓ Jika Department_id = 90, maka Role_ID = 10; jika Department_ID = 80, maka Role_ID = 20, selain itu, Role_ID = 30
   - ✓ Sts= A

   Data Karyawan yang dikopikan adalah karyawan dengan masa bakti di atas 15 tahun dan gajinya diatas rata-rata gaji seluruh karyawan.

4. Tampilkan buat sebuah view bernama V[NimPendek] yang menampilkan jumlah karyawan yang masuk pada setiap departemen yang ada, sebelum tahun 1995 dan sesudah tahun 1995.

5. Tampilkan department dengan jumlah karyawan terbanyak. (output hanya 1 baris data saja)

BIBLIOGRAPHY

Oracle. 2004. *Oracle Database 10g: SQL Fundamental I Volume I*. Redwood *Shores*, Amerika Serikat.: Oracle *Corporation*

Oracle. 2004. *Oracle Database 10g: SQL Fundamental I Volume II*. Redwood *Shores*, Amerika Serikat.: Oracle *Corporation*

Oracle. 2009. *Oracle Database 11g*: *SQL Fundamental* II. Redwood *Shores*, Amerika Serikat.: Oracle *Corporation*