MODUL PRAKTIKUM

BASIS DATA

PROGRAM STUDI MATEMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UIN MAULANA MALIK IBRAHIM MALANG

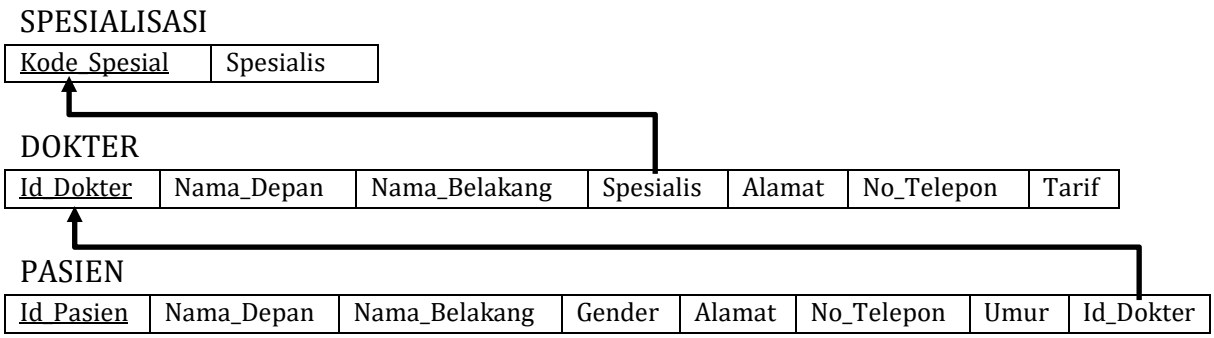
Penyusun
Hisyam Fahmi, M.Kom

2021

Tutorial DDL Database
Basis Data
Semester Genap 2021/2022

I. Membuat Tabel

Untuk latihan membuat tabel, perhatikan contoh berikut.
Misalkan Anda diminta untuk membuat basis data berdasarkan skema relasional berikut:



Dengan keterangan mengenai atribut tiap tabel adalah sebagai berikut:

Tabel SPESIALISASI

Nama *Column*	Tipe Data	Boleh Null	PK	FK
Kode_Spesial	CHAR (5)	-	√	
Spesialis	VARCHAR (25)	-		

Tabel DOKTER

Nama *Column*	Tipe Data	Boleh Null	PK	FK
Id_Dokter	CHAR (5)	-	√	
Nama_Depan	VARCHAR (15)	-		
Nama_Belakang	VARCHAR (15)	√		
Spesialis	CHAR (5)	-		Ke Kode_Spesial di table SPESIALISASI
Alamat	VARCHAR (50)	-		
No_Telepon	VARCHAR (15)	√		
Tarif	NUMERIC (10, 2)	-		

Tabel PASIEN

Nama	Tipe Data	Boleh *Null*	PK	FK
Id_Pasien	CHAR (5)	-	√	
Nama_Depan	VARCHAR (15)	-		
Nama_Belakang	VARCHAR (15)	√		
Gender	CHAR (1)	-		
Alamat	VARCHAR (50)	√		

No_Telepon	VARCHAR (15)	√		
Umur	INT	-		
Id_Dokter	CHAR (5)	√		ke Id_Dokter di tabel DOKTER

Agar tabel-tabel yang Anda buat tertata dengan rapi, maka Anda perlu membuat *schema* yang dikhusukan untuk suatu keperluan tertentu. Untuk tabel-tabel di atas, Anda dapat memasukkan ke dalam *schema* POLIKLINIK.

II. Membuat *Table*

Untuk membuat tabel, format sintaks SQL secara umum adalah sebagai berikut:

```
CREATE TABLE [nama_database.]nama_table(
    nama_atribut1    tipe_atribut1 [NOT NULL],
    nama_atribut2    tipe_atribut2 [NOT NULL],
    :
    PRIMARY KEY (nama_atribut1, . . .)
    FOREIGN KEY (nama_atribut) REFERENCES
      nama_tabel_yang_direfer(nama_atribut_yang_direfer)
      [ON DELETE RESTRICT | CASCADE | SET NULL | SET
DEFAULT][ON UPDATE RESTRICT | CASCADE ]
);
```

Keterangan:

1. Tanda " [ ] " menyatakan pilihan, boleh tidak digunakan
2. Tanda " : " menyatakan baris-baris berikutnya serupa dengan baris sebelumnya
3. Tanda " | " menyatakan beberapa pilihan yang dapat digunakan

Sesuai dengan format tersebut, maka SQL untuk membuat tabel SPESIALISASI adalah sebagai berikut:

```
CREATE TABLE POLIKLINIK.SPESIALISASI (
    Kode_Spesial CHAR(5) NOT NULL,
    Spesialis VARCHAR(25) NOT NULL,
    PRIMARY KEY (Kode_Spesial)
);
```

SQL untuk membuat table DOKTER adalah sebagai berikut:

```
CREATE TABLE POLIKLINIK.DOKTER (
    Id_Dokter CHAR(5) NOT NULL,
    Nama_Depan VARCHAR(15) NOT NULL,
    Nama_Belakang VARCHAR(15),
    Spesialis CHAR(5),
```

```
    Alamat VARCHAR(50) NOT NULL,
    No_Telepon CHAR(15),
    Tarif NUMERIC(10,2) NOT NULL,
    PRIMARY KEY (Id_Dokter),
    FOREIGN KEY (Spesialis) REFERENCES
POLIKLINIK.SPESIALISASI(Kode_Spesial) ON DELETE CASCADE ON
UPDATE CASCADE
);
```

Sedangkan SQL untuk membuat tabel PASIEN adalah sebagai berikut:

```
CREATE TABLE POLIKLINIK.PASIEN (
    Id_Pasien CHAR(5) NOT NULL,
    Nama_Depan VARCHAR(15) NOT NULL,
    Nama_Belakang VARCHAR(15),
    Gender CHAR(1) NOT NULL,
    Alamat VARCHAR(50),
    No_Telepon CHAR(15),
    Umur INT NOT NULL,
    Id_Dokter CHAR(5),
    PRIMARY KEY (Id_Pasien),
    FOREIGN KEY (Id_Dokter) REFERENCES
POLIKLINIK.DOKTER(Id_Dokter) ON DELETE CASCADE ON UPDATE
CASCADE
);
```

III. Mengisi Tabel

Untuk mengisi tabel, format sintaks SQL secara umum adalah sebagai berikut:

```
INSERT INTO nama_database.nama_tabel VALUES
(nilai_atribut_1,
..., nilai_atribut_n);
```

Keterangan:

1. Tanda "[ ]" menyatakan pilihan, boleh tidak digunakan
2. Tanda "..." menyatakan elemen-elemen berikutnya serupa dengan elemen sebelumnya

Sesuai dengan format tersebut, maka contoh SQL untuk mengisi tabel SPESIALISASI adalah sebagai berikut:

```
INSERT INTO POLIKLINIK.SPESIALISASI VALUES  ('SP001',
'Jantung');
```

Contoh SQL untuk mengisi tabel DOKTER adalah sebagai berikut:

```
INSERT INTO POLIKLINIK.DOKTER VALUES  ('DR001', 'Syaiful',
'Anwar', 'SP001', 'Jakarta Pusat', '+6281111222', 150000);
```

Contoh SQL untuk mengisi tabel PASIEN adalah sebagai berikut:

```
INSERT INTO POLIKLINIK.PASIEN VALUES  ('P0001', 'Ubet', '',
'L', 'Bandung', '+6282222123', 21, 'DR011');
```

Lakukan penambahan data sehingga state basis data poliklinik adalah sebagai berikut:
SPESIALISASI

Kode_Spesialis	Spesialis
SP001	Jantung
SP006	Bedah
SP005	Saraf
SP007	Mata
SP008	Anak

DOKTER

Id_Dokter	Nama_Depan	Nama_Belakang	Spesialisasi	Alamat	No_Telepon	Tarif
DR001	Syaiful	Anwar	SP001	Jakarta Pusat	+6281111222	150000
DR003	Edi	Harto	SP006	Bogor	+6221211321	200000
DR004	Andrea	Dian	SP008	Depok	+6288899988	100000
DR011	Dewi		SP006	Bekasi	+6212332111	120000
DR009	Muhammad	Ridwan	SP001	Depok	+625656565	120000
DR012	Agung	Pribadi	SP005	Jakarta Pusat	+624545111	180000
DR007	James	Bon	SP007	Bekasi	+620000007	230000
DR010	Ida	Nurhaida	SP008	Bogor	+621921211	70000

PASIEN

Id_Psien	Nama_Depan	Nama_Belakang	Gender	Alamat	No_Telepon	Umur	Id_Dokter
P0001	Ubet		L	Bandung	+6282222123	21	DR011
P0003	Juju	Jubaidah	P	Cimahi		70	DR012
P0002	Bon	Kurei	L	Bogor		45	DR001
P0005	Arya	Stak	P	Jakarta	+628989898	6	DR004
P0008	Mario	Bolateli	L	Depok	+627117213	21	DR007
P0009	Jamal	Widodo	L	Bekasi	+622167809	55	DR009
P0010	Kiara		P	Bogor		4	DR010
P0011	Bondan	Prakosa	L	Jakarta	+6200101011	21	DR003
P0013	Gatot	Kaca	L	Solo		45	DR003
P0014	Pipit		P		+6233333333	23	DR011

IV. Lain-Lain

Untuk melihat keseluruhan isi tabel tertentu dapat dilakukan dengan perintah:

```
SELECT * FROM [nama_tabel]
```

V. Menghapus Data Tabel

Format umum untuk melakukan operasi DELETE adalah sebagai berikut:

```
DELETE FROM [Nama_DB].NAMA_TABLE [WHERE
CONDITIONAL_STATEMENT];
```

Jika conditional statement tidak disertakan, seluruh data pada tabel akan dihapus. Misalnya, dokter James Bon sudah tidak bekerja pada poliklinik lagi. Untuk menghapus dokter dengan kode nama 'James Bon', SQL statement yang dieksekusi adalah:

```
DELETE FROM POLIKLINIK.DOKTER WHERE Id_Dokter = 'DR007';
```

Satu record akan terhapus. Hal lain yang perlu diperhatikan adalah state dari tabel PASIEN. Record yang merefer dokter 'James Bon' yaitu data pasien 'Mario Bolateli' akan ikut terhapus karena referential constraint ke tabel PASIEN adalah ON DELETE CASCADE.

VI. Mengubah Data Tabel

Format umum untuk melakukan operasi UPDATE adalah sebagai berikut:

```
UPDATE[ Nama_DB].NAMA_TABLE
SET COLUMN1 = VAL1[, COLUMN2 = VAL2, ..., COLUMNN = VALN]
[WHERE CONDITIONAL_STATEMENT];
```

Misalnya, dokter dengan spesialisasi 'jantung' akan dinaiikan tarifnya menjadi 200000. Perintah update untuk merefleksikan hal ini ke dalam state basis data poliklinik adalah:

```
UPDATE POLIKLINIK.DOKTER
SET Tarif = 200000
WHERE Spesialis = 'SP001';
```

VII. Modifikasi Definisi Tabel

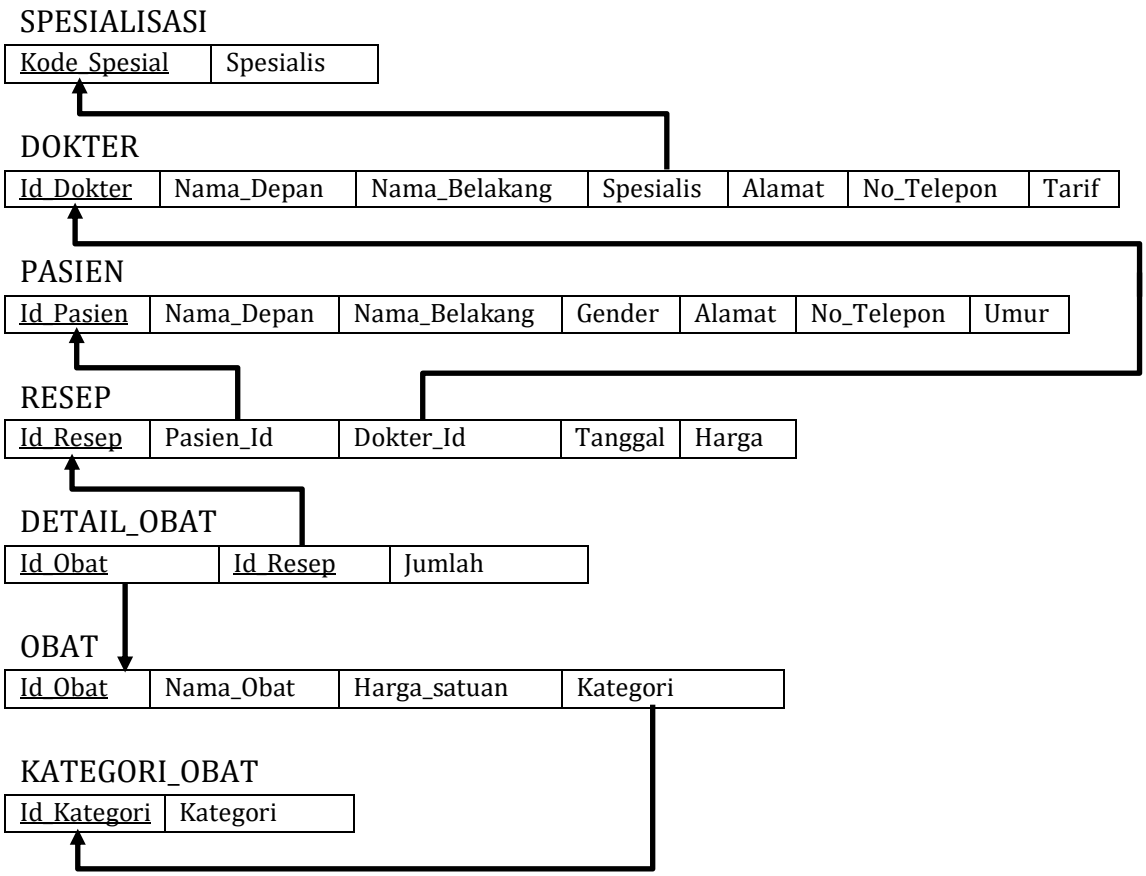
Modifikasi dapat dilakukan pada tabel yang sudah ada. Salah satunya adalah merubah tipe data kolom. Misalnya, ada pasien baru yang nama depannya lebih dari 15 karakter. Untuk menangani masalah ini, tipe data kolom nama_depan pada tabel PASIEN harus diubah menjadi VARCHAR (20). Hal ini dapat dilakukan dengan perintah:

```
ALTER TABLE POLIKLINIK.PASIEN ALTER COLUMN Nama_Depan SET
TYPE VARCHAR(20);
```

Tutorial Basic SQL
Basis Data
Semester Genap 2021/2022

I. Membuat Skema Database

Buatlah database POLIKLINIK sesuai dengan skema relasi berikut:



Dengan keterangan mengenai atribut tiap tabel adalah sebagai berikut:

Tabel SPESIALISASI

Nama *Column*	Tipe Data	Boleh Null	PK	FK
Kode_Spesial	CHAR (5)	-	√	
Spesialis	VARCHAR (25)	-		

Tabel DOKTER

Nama *Column*	Tipe Data	Boleh Null	PK	FK
Id_Dokter	CHAR (5)	-	√	
Nama_Depan	VARCHAR (15)	-		
Nama_Belakang	VARCHAR (15)	√		
Spesialis	CHAR (5)	√		Ke Kode_Spesial di table SPESIALISASI
Alamat	VARCHAR (50)	-		
No_Telepon	VARCHAR (15)	√		
Tarif	NUMERIC (10, 2)	-		

Tabel PASIEN

Nama	Tipe Data	Boleh *Null*	PK	FK
Id_Pasien	CHAR (5)	-	√	
Nama_Depan	VARCHAR (15)	-		
Nama_Belakang	VARCHAR (15)	√		
Gender	CHAR (1)	-		
Alamat	VARCHAR (50)	√		
No_Telepon	VARCHAR (15)	√		
Umur	INT	√		

Tabel RESEP

Nama	Tipe Data	Boleh *Null*	PK	FK
Id_Resep	CHAR (10)	-	√	
Pasien_Id	CHAR (5)	-		Ke Id_Pasien di table PASIEN
Dokter_Id	CHAR (5)	-		Ke Id_Dokter di table DOKTER
Tanggal	DATE	-		
Harga	NUMERIC (10,2)	√		

Tabel DETAIL_OBAT

Nama	Tipe Data	Boleh *Null*	PK	FK
Id_Obat	CHAR (5)	-	√	Ke Id_Obat di table OBAT
Id_Resep	CHAR (10)	-	√	Ke Id_Resep di table RESEP
Jumlah	INT	-		

Tabel OBAT

Nama	Tipe Data	Boleh *Null*	PK	FK
Id_Obat	CHAR (5)	-	√	
Nama_Obat	VARCHAR (25)	-		
Harga_Satuan	NUMERIC (10,2)	-		

Kategori	CHAR (5)	√		Ke Id_Kategori di table KATEGORI_OBAT

Tabel KATEGORI_OBAT

Nama	Tipe Data	Boleh *Null*	PK	FK
Id_Kategori	CHAR (5)	-	√	
Kategori	VARCHAR (20)	-		

Isilah tabel-tabel tersebut dengan data sebagai berikut.

SPESIALISASI

Kode_Spesial	Spesialis
SP001	Jantung
SP006	Bedah
SP005	Saraf
SP007	Mata
SP008	Anak

DOKTER

Id_Dokter	Nama_Depan	Nama_Belakang	Spesialisasi	Alamat	No_Telepon	Tarif
DR001	Syaiful	Anwar	SP001	Jakarta Pusat	+6281111222	150000
DR003	Edi	Harto	SP006	Bogor	+6221211321	200000
DR004	Andrea	Dian	SP008	Depok	+6288899988	100000
DR011	Dewi		SP006	Bekasi	+6212332111	120000
DR009	Muhammad	Ridwan	SP001	Depok	+625656565	120000
DR012	Agung	Pribadi	SP005	Jakarta Pusat	+624545111	180000
DR007	James	Bon	SP007	Bekasi	+620000007	230000
DR010	Ida	Nurhaida	SP008	Bogor	+621921211	70000

PASIEN

Id_Pasien	Nama_Depan	Nama_Belakang	Gender	Alamat	No_Telepon	Umur
P0001	Ubet		L	Bandung	+6282222123	21
P0003	Juju	Jubaidah	P	Cimahi		70
P0002	Bon	Kurei	L	Bogor		45
P0005	Arya	Stak	P	Jakarta	+628989898	6
P0008	Mario	Bolateli	L	Depok	+627117213	21
P0009	Jamal	Widodo	L	Bekasi	+622167809	55
P0010	Kiara		P	Bogor		4
P0011	Bondan	Prakosa	L	Jakarta	+6200101011	21
P0013	Gatot	Kaca	L	Solo		45
P0014	Pipit		P		+6233333333	23

RESEP

Id_Resep	Pasien	Dokter	Tanggal	Harga
R250115001	P0001	DR011	25/1/2015	
R250115002	P0009	DR001	25/1/2015	
R260115001	P0008	DR007	26/1/2015	
R270115001	P0014	DR003	27/1/2015	
R300115001	P0010	DR010	30/1/2015	
R300115002	P0013	DR003	30/1/2015	
R010215001	P0009	DR001	1/2/2015	
R010215002	P0003	DR009	1/2/2015	
R010215003	P0010	DR010	1/2/2015	
R020215001	P0005	DR004	2/2/2015	
R020215002	P0009	DR001	2/2/2015	
R020215003	P0014	DR012	2/2/2015	
R030215001	P0005	DR004	3/2/2015	
R030215002	P0003	DR009	3/2/2015	

DETAIL_OBAT

Id_Obat	Id_Resep	Jumlah
OB013	R250115001	1
OB002	R250115002	5
OB003	R250115002	2
OB012	R270115001	1
OB015	R300115001	1
OB002	R010215001	5
OB001	R010215002	2
OB002	R010215002	4
OB014	R010215003	1
OB016	R020215001	2
OB002	R020215002	3
OB003	R020215002	1
OB007	R020215003	4
OB008	R020215003	6
OB016	R030215001	1
OB003	R030215002	2
OB004	R030215002	4

KATEGORI_OBAT

Id_Kategori	Kategori
OK001	Jantung
OK002	Saraf
OK003	Infus
OK004	Nutrisi
OK005	Mata

OBAT

Id_Obat	Nama_Obat	Harga_Satuan	Kategori
OB001	Akrinor Tablet	65000	OK001
OB002	Cardiject Vial	7800	OK001
OB003	Fargoxin Injeksi	21000	OK001
OB004	Kendaron Ampul	20000	OK001
OB005	Tiaryt Tablet	16000	OK001
OB006	Exelon Capsule 3 Mg	70000	OK002
OB007	Fordesia Tablet	79000	OK002
OB008	Reminyl Tablet 4 Mg	33000	OK002
OB009	Albucid Tetes Mata	15000	OK005

OB010	Cendo Fenicol Salep Mata	20000	OK005
OB011	Interflox Tetes Mata	36000	OK005
OB012	Haemaccel Infus	200000	OK003
OB013	Human Albumin Infus	900000	OK003
OB014	Curfos Syrup	74000	OK004
OB015	Vitacur Syrup	36000	OK004
OB016	Cerebrovit Active	133000	OK004

II. Basic SQL

Untuk mendapatkan data yang ada di dalam Tabel, dapat digunakan SQL query. Perintahnya adalah sebagai berikut:

```
SELECT attribute_list
FROM table_list
[WHERE condition]
[ORDER BY attribute_list];
```

Misalnya, kita ingin mengetahui nama-nama obat pada kategori obat saraf.

```
SELECT O.Nama_Obat
FROM OBAT AS O, KATEGORI_OBAT AS KO
WHERE O.Kategori=KO.Id_Kategori AND
  LOWER(KO.Kategori)='saraf';
```

Keterangan: LOWER merupakan fungsi pada MySQL untuk merubah string pada attribute menjadi huruf kecil semua.

Untuk menampilkan hasil query secara terurut kita bisa menambahkan klausa ORDER BY diikuti kolom yang diurutkan dan metode pengurutannya, ASC untuk mengurutkan dari kecil ke besar (dari A-Z) atau DESC untuk sebaliknya. Misalkan, kita ingin menampilkan nama pasien yang terurut berdasarkan umurnya dari yang paling muda.

```
SELECT CONCAT(Nama_Depan, ' ', Nama_Belakang) AS
  Nama_Pasien, Umur
FROM PASIEN
ORDER BY Umur ASC;
```

Keterangan: fungsi “CONCAT()” merupakan perintah untuk menggabungkan string dalam MySQL.

Untuk melakukan pengecekan terhadap beberapa bagian string saja bisa digunakan operator LIKE. Misalkan, kita ingin mendapatkan nama dokter yang nomor teleponnya berawalan ‘+628’.

```
SELECT CONCAT(Nama_Depan, ' ', Nama_Belakang) AS
  Nama_Dokter, No_Telepon
FROM DOKTER
WHERE No_Telepon LIKE '+628%';
```

Tambahkan keyword DISTINCT untuk menghilangkan duplikasi data hasil query. Misalkan, kita ingin menampilkan nama pasien yang pernah mendapatkan resep dari dokter.

```
SELECT DISTINCT CONCAT(Nama_Depan, ' ', Nama_Belakang) AS
  Nama_Pasien
FROM PASIEN, RESEP
WHERE Id_Pasien=Pasien_Id;
```

Bandingkan dengan hasil query tanpa DISTINCT.

III. Advance SQL

Operasi JOIN

MySQL mendukung operasi INNER JOIN (atau cukup JOIN saja), RIGHT OUTER JOIN, LEFT OUTER JOIN, FULL OUTER JOIN, dan CROSS JOIN. Misalnya, kita ingin mengetahui resep mana saja yang dikeluarkan oleh Dr. Syaiful Anwar.

```
SELECT R.Id_Resep, CONCAT(P.Nama_Depan, ' ',
  P.Nama_Belakang) AS Nama_Pasien, R.Tanggal
FROM RESEP R
INNER JOIN DOKTER D ON R.Dokter_Id=D.Id_Dokter
INNER JOIN PASIEN P ON R.Pasien_Id=P.Id_Pasien
WHERE LOWER(D.Nama_Depan)='syaiful' AND
  LOWER(D.Nama_Belakang)='anwar';
```

Operasi Himpunan

MySQL mendukung operasi UNION, INTERSECT, dan EXCEPT. Misalnya, kita ingin mendapatkan obat jantung yang belum pernah debeli pasien.

```
(SELECT Nama_Obat
FROM OBAT O, KATEGORI_OBAT KO
WHERE KO.Id_Kategori=O.Kategori AND
  LOWER(KO.Kategori)='jantung')
EXCEPT (
  SELECT DISTINCT O.Nama_Obat
```

```
  FROM OBAT O, DETAIL_OBAT DT, KATEGORI _OBAT KO
  WHERE O.Id_Obat=DT.Id_Obat AND
     KO.Id_Kategori=O.Kategori AND
     LOWER(KO.Kategori)='jantung'
);
```

Fungsi Agregat

PostgreSQL menyediakan aggregate function COUNT, SUM, MAX, MIN, dan AVG. Contoh aggregate function lain yang disediakan adalah VARIANCE, STDDEV, dan CORRELATION. Misalnya, kita ingin mengetahui jumlah obat yang terjual berdasarkan kategorinya.

```
SELECT KO.Kategori AS Kategori, SUM(DT.Jumlah) AS
  Jumlah_Terjual
FROM DETAIL_OBAT DT, OBAT O, KATEGORI_OBAT KO
WHERE DT.Id_Obat=O.Id_Obat AND O.Kategori=KO.Id_Kategori
GROUP BY KO.Kategori;
```

IV. Latihan Basic Query

1. Buatlah query untuk mengembalikan data obat yang harga satuannya tidak lebih dari Rp 50.000,00. Tampilkan nama obat, harga, dan nama kategorinya.

```
        nama_obat          | harga_satuan | kategori
---------------------------+--------------+----------
 Cardiject Vial            |      7800.00 | Jantung
 Fargoxin Injeksi          |     21000.00 | Jantung
 Kendaron Ampul            |     20000.00 | Jantung
 Tiaryt Tablet             |     16000.00 | Jantung
 Reminyl Tablet 4 Mg       |     33000.00 | Saraf
 Albucid Tetes Mata        |     15000.00 | Mata
 Cendo Fenicol Salep Mata  |     20000.00 | Mata
 Interflox Tetes Mata      |     36000.00 | Mata
 Vitacur Syrup             |     36000.00 | Nutrisi
(9 rows)
```

2. Buatlah query untuk mengembalikan data pasien yang nama depannya mengandung huruf '**o**'. Tampilkan semua kolomnya kecuali nomor telepon.

```
 id_pasien |  nama_pasien    | gender | alamat   | umur
-----------+-----------------+--------+----------+------
 P0002     | Bon Kurei       | L      | Bogor    |   45
 P0008     | Mario Bolateli  | L      | Depok    |   21
 P0011     | Bondan Prakosa  | L      | Jakarta  |   21
 P0013     | Gatot Kaca      | L      | Solo     |   45
(4 rows)
```

3. Buatlah query untuk mengembalikan data pasien yang berobat pada dokter spesialis jantung. Tampilkan data nama (nama lengkap), umur, dan nama dokter yang menangani.

```
  nama_pasien   | umur |     dokter
----------------+------+-----------------
 Jamal Widodo   |   55 | Syaiful Anwar
 Juju Jubaidah |   70 | Muhammad Ridwan
(2 rows)
```

4. Buatlah query untuk menampilkan resep untuk obat 'Cardiject Vial'. Tampilkan nomor resep, dan tanggalnya.

```
  id_resep   |  tanggal
-------------+------------
 R250115002 | 2015-01-25
 R010215001 | 2015-02-01
 R010215002 | 2015-02-01
 R020215002 | 2015-02-02
(4 rows)
```

5. Buatlah query untuk mengembalikan daftar nama obat yang dalam satu resep diminta dalam jumlah lebih dari 2. Tampilkan id obat dan nama obatnya tanpa duplikasi.

```
 id_obat |      nama_obat
---------+---------------------
 OB007   | Fordesia Tablet
 OB008   | Reminyl Tablet 4 Mg
 OB002   | Cardiject Vial
 OB004   | Kendaron Ampul
(4 rows)
```

Tutorial View, Stored Procedure, dan Trigger
Basis Data
Semester Genap 2021/2022

I. SQL

Operasi JOIN

PostgreSQL mendukung operasi INNER JOIN (atau cukup JOIN saja), RIGHT OUTER JOIN, LEFT OUTER JOIN, FULL OUTER JOIN, NATURAL JOIN, dan CROSS JOIN. Misalnya, kita ingin mengetahui resep mana saja yang dikeluarkan oleh Dr. Syaiful Anwar.

```
SELECT R.Id_Resep, (P.Nama_Depan || ' ' || P.Nama_Belakang) AS
  Nama_Pasien, R.Tanggal
FROM RESEP R
INNER JOIN DOKTER D ON R.Dokter_Id=D.Id_Dokter
INNER JOIN PASIEN P ON R.Pasien_Id=P.Id_Pasien
WHERE LOWER(D.Nama_Depan)='syaiful' AND
  LOWER(D.Nama_Belakang)='anwar';
```

Fungsi Agregat

PostgreSQL menyediakan aggregate function COUNT, SUM, MAX, MIN, dan AVG. Contoh aggregate function lain yang disediakan adalah VARIANCE, STDDEV, dan CORRELATION. Biasanya fungsi agregat digabungkan dengan klausa GROUP BY untuk melakukan agregasi berdasarkan kolom tertentu. Misalnya, kita ingin mengetahui jumlah obat yang terjual berdasarkan kategorinya.

```
SELECT KO.Kategori AS Kategori, SUM(DT.Jumlah) AS Jumlah_Terjual
FROM DETAIL_OBAT DT, OBAT O, KATEGORI_OBAT KO
WHERE DT.Id_Obat = O.Id_Obat AND O.Kategori = KO.Id_Kategori
GROUP BY KO.Kategori;
```

II. VIEW

View merupakan sebuah tabel virtual yang dibuat menggunakan SQL query. Untuk membuat sebuah view, dapat dijalankan sintaks berikut:

```
CREATE [ OR REPLACE ] [ TEMP | TEMPORARY ] VIEW name [ ( column_name
[, ...] ) ]
  AS query
```

Untuk membuat view yang dapat menampilkan daftar obat beserta nama kategorinya, dapat dilakukan dengan perintah berikut:

```
CREATE VIEW daftar_obat AS
    SELECT O.nama_obat, KO.kategori
    FROM OBAT O, KATEGORI_OBAT KO
    WHERE O.kategori = KO.id_kategori;
```

Setelah view berhasil dibuat, kita dapat melakukan query terhadap view tersebut seperti halnya query pada base relation.
Coba lakukan proses insert atau update ke view yang telah dibuat sebelumnya, dan lihatlah apakah perintah-perintah tersebut dapat dijalankan. Misalnya perintah insert ke view daftar_obat seperti berikut:

```
INSERT INTO daftar_obat VALUES ('Tes Obat Baru', 'Saraf');
```

Sekarang coba tambahkan satu baris ke relasi OBAT, kemudian cek apakah data baru tersebut bisa dilihat dengan melakukan query terhadap view daftar_obat.

```
INSERT INTO obat VALUES ('OB100', 'Tes Obat Baru', 100, 'OK002');

SELECT * from DAFTAR_OBAT;
```

Untuk menghapus view yang telah dibuat, dapat dilakukan dengan perintah **DROP VIEW nama_view;**

III. Stored Procedure

Stored Procedure dan fungsi merupakan bagian dari bahasa prosedural di dalam SQL yang biasa disebut PL/SQL (atau PL/pgSQL dalam Postgre).
Untuk membuat stored procedure dengan PostgreSQL, sintaksnya adalah sebagai berikut:

```
CREATE [ OR REPLACE ] FUNCTION <schema_name>.<function_name>
([[{ IN | OUT | INOUT }] <argument_name1> <argument_type1>, ... ])
[ RETURNS <return_type> ] AS
$$
    [ DECLARE
        <variable_name1> <variable_type1>;
        ⋮
    ]
    BEGIN
        <statements>
    END;
$$
LANGUAGE plpgsql;
```

Misalkan kita ingin menghitung harga total pada relasi RESEP. Maka kita dapat membuat stored procedure-nya sebagai berikut:

```
CREATE OR REPLACE FUNCTION hitung_harga (a_resep CHAR(10))
RETURNS NUMERIC(10,2) AS
$$
    DECLARE
        harga_total NUMERIC(10,2);
    BEGIN
        SELECT SUM(jumlah*harga_satuan) INTO harga_total
        FROM DETAIL_OBAT DT, OBAT O
        WHERE DT.id_resep = a_resep AND DT.id_obat = O.id_obat;

        UPDATE resep SET harga = harga_total
        WHERE id_resep = a_resep;

        RETURN harga_total;
    END;
$$
LANGUAGE plpgsql;
```

Untuk menjalankan stored procedure yang sudah dibuat, dapat dilakukan dengan perintah berikut:

```
SELECT <nama_schema>.<nama_fungsi> ([<argument_name1>, … ]);
```

Contoh:

```
SELECT hitung_harga('R250115002');
```

Untuk menghitung semua harga total, kita bisa menggunakan looping pada stored procedure.

```
CREATE OR REPLACE FUNCTION hitung_semua_harga()
RETURNS void AS
$$
    DECLARE
        row RECORD;
    BEGIN
        FOR row IN
            SELECT R.id_resep, SUM(jumlah*harga_satuan) AS
                   harga_total
            FROM RESEP R, DETAIL_OBAT DT, OBAT O
            WHERE R.id_resep = DT.id_resep AND DT.id_obat = O.id_obat
            GROUP BY R.id_resep
        LOOP
            UPDATE resep SET harga = row.harga_total
            WHERE id_resep = row.id_resep;
        END LOOP;
    END;
$$
LANGUAGE plpgsql;
```

IV. Trigger

Trigger merupakan operasi pada sebuah tabel yang otomatis dijalankan ketika ada kejadian tertentu. Format sintaks PL/SQL untuk membuat stored procedure yang akan dipanggil oleh trigger sama dengan yang digunakan untuk membuat stored procedure biasa, namun <return_type> selalu bernilai trigger dan tidak boleh mempunyai argumen. Misalkan kita akan membuat trigger setiap ada kejadian INSERT, UPDATE, atau DELETE terhadap tabel DETAIL_OBAT, secara otomatis akan ter-update attribut "harga" pada tabel RESEP.

```
CREATE OR REPLACE FUNCTION total_harga_resep()
RETURNS trigger AS
$$
    DECLARE
        total NUMERIC(10,2);
    BEGIN
        IF (TG_OP = 'INSERT') THEN
            SELECT NEW.jumlah*O.harga_satuan INTO total
            FROM OBAT O
            WHERE O.id_obat = NEW.id_obat;

            UPDATE resep SET harga = harga+total
            WHERE id_resep = NEW.id_resep;
            RETURN NEW;
        ELSEIF (TG_OP = 'UPDATE') THEN
            SELECT OLD.jumlah*O.harga_satuan INTO total
            FROM OBAT O
            WHERE O.id_obat = OLD.id_obat;

            UPDATE resep SET harga = harga-total
            WHERE id_resep = OLD.id_resep;

            SELECT NEW.jumlah*O.harga_satuan INTO total
            FROM OBAT O
            WHERE O.id_obat = NEW.id_obat;

            UPDATE resep SET harga = harga+total
            WHERE id_resep = NEW.id_resep;
            RETURN NEW;
        ELSEIF (TG_OP = 'DELETE') THEN
            SELECT OLD.jumlah*O.harga_satuan INTO total
            FROM OBAT O
            WHERE O.id_obat = OLD.id_obat;

            UPDATE resep SET harga = harga-total
            WHERE id_resep = OLD.id_resep;
            RETURN OLD;
        END IF;
    END;
$$
LANGUAGE plpgsql;
```

Untuk membuat trigger dengan PostgreSQL, sintaksnya adalah sebagai berikut:

```
CREATE TRIGGER <trigger_name>
{ BEFORE | AFTER } { INSERT | UPDATE | DELETE [ OR ... ] }
ON <schema_name>.<table_name> [ FOR EACH { ROW | STATEMENT } ]
EXECUTE PROCEDURE <function_name> ( [ <argument_name1>, ... ] );
```

Sesuai dengan format tersebut, maka perintah untuk membuat trigger yang memanggil stored procedure "total_harga_resep" adalah sebagai berikut.

```
CREATE TRIGGER jumlah_total_harga
AFTER INSERT OR UPDATE OR DELETE
ON detail_obat FOR EACH ROW
EXECUTE PROCEDURE total_harga_resep();
```

Untuk memanggil trigger yang telah dibuat, dapat dilakukan dengan menjalankan event yang memicu trigger tersebut. Misalnya, trigger "jumlah_total_harga" dipanggil setelah terjadi event INSERT atau UPDATE atau DELETE ke tabel DETAIL_OBAT.

```
INSERT INTO DETAIL_OBAT VALUES ('OB001', 'R030215002', 5);
```

Selanjutnya periksalah perubahan yang terjadi pada kolom harga pada table RESEP.

```
SELECT * FROM RESEP WHERE id_resep = 'R030215002';
```

Silakan cek juga untuk event update dan delete pada tabel DETAIL_OBAT.