Tutorial 2 PostgreSQL
Basis Data
CSGE602070
Semester Gasal 2022/2023

Note!

Baca dengan teliti agar tidak ada yang terlewat. Pastikan setiap langkah yang Anda lakukan di masukkan ke dalam laporan. **Kerjakan semua latihan Soal** dengan menampilkan SQL yang Anda buat beserta capture output dari SQL yang anda buat.

Riwayat Versi

Versi	*Timestamp*	Halaman	Perubahan
1	31-10-2022 / 19:00	Semua	Rilis Pertama
2	5-11-2022 / 00:00	18	Perubahan latihan 2 Bagian B

Note : Setiap revisi ditandai dengan highlights warna kuning Revisi

Tutorial ini menggunakan skema basis data **SIWANAP** hasil kelanjutan dari tutorial 1. Jangan lupa **SET SEARCH PATH** terlebih dahulu.

I. Basic SQL

Pada Tutorial I kita telah membuat database **SIWANAP**, untuk mendapatkan data yang ada di dalam sebuah tabel tertentu, maka dapat menggunakan SQL *Query* dengan syntax sebagai berikut:

```
SELECT <attribute list>
FROM <table list>
[WHERE <condition>]
[ORDER BY <attribute list>];
```

Jika kondisi **WHERE** tidak dicantumkan, maka akan menampilkan seluruh hasil kombinasi yang ada karena dianggap **WHERE TRUE**. Kondisi **WHERE** sering digunakan untuk melakukan penyaringan data.

Contoh 1:

Misalkan, kita ingin mengetahui nama **DOKTER** dengan id 'DO04'. Maka digunakan perintah SQL sebagai berikut:

```
SELECT nama
FROM DOKTER
```

```
WHERE id_dokter = 'DO04';
```

Keterangan :
1. nama merupakan salah satu nama atribut yang terdapat di dalam tabel dokter.

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.

Contoh 2:

Misalkan, kita ingin menampilkan `nama`, `jenis`, dan `kapasitas` dari tabel **KAMAR** terurut berdasarkan harga. Kita bisa gunakan perintah SQL sebagaimana berikut.

```
SELECT nama, jenis, kapasitas
FROM KAMAR
ORDER BY harga;
```

Kita juga bisa menggunakan **operasi aritmatika (+, -, /, *)** dan **logika (AND, OR , etc)** dalam *query* SQL. Operasi logika hanya bisa digunakan pada klausa **WHERE**.

Contoh 3:

Misalkan, kita ingin menampilkan dari **KAMAR** yang memiliki kapasitas di antara 2 sampai 4 (inklusif). Kita bisa gunakan perintah sebagai berikut.

```
SELECT *
FROM KAMAR
WHERE (kapasitas >= 2) AND (kapasitas <= 4);
```

Kita juga bisa menggunakan operasi **BETWEEN** sebagai berikut:

```
SELECT *
FROM KAMAR
WHERE (kapasitas BETWEEN 2 AND 4);
```

Untuk melakukan pengecekan terhadap beberapa bagian string saja bisa digunakan operator **LIKE** dan tanda '_' (*underscore*) atau '%' (lebih dari satu karakter) sebagai *pattern matching.*

Contoh 4:

Misalkan, kita ingin melihat seluruh informasi dari **PERAWAT** yang memiliki string 'Karen' di awal `nama`-nya. Kita bisa gunakan perintah SQL sebagaimana berikut.

```
SELECT nama, email
FROM PERAWAT
WHERE nama LIKE 'Karen%';
```

Kita juga bisa mengoperasikan hasil dua atau lebih *query* yang berbeda dengan menggunakan operasi **UNION**, **INTERSECT** dan **EXCEPT** .

Contoh 5:

Misalkan kita ingin menampilkan informasi dokter yang spesialisasinya adalah '*anak*' **atau** spesialisasinya 'bedah' . Kita bisa menggunakan operator **UNION**.

```
(SELECT * FROM DOKTER
WHERE spesialisasi = 'anak')
UNION
(SELECT * FROM DOKTER
WHERE spesialisasi = 'bedah');
```

Contoh 6:

Selanjutnya, apabila kita ingin menampilkan informasi **DOKTER** yang nama diawali dengan huruf C **dan** memiliki spesialisasi 'bedah'. Kita bisa menggunakan operator **INTERSECT**.

```
(SELECT * FROM DOKTER
WHERE nama LIKE 'C%')
INTERSECT
(SELECT * FROM DOKTER
WHERE spesialisasi = 'bedah');
```

Contoh 7:

Untuk **EXCEPT**, dipakai jika kita ingin menampilkan informasi **DOKTER** yang nama diawali dengan huruf C namun **tidak memiliki** spesialisasi 'bedah'.

```
(SELECT * FROM DOKTER
WHERE nama LIKE 'C%')
EXCEPT
(SELECT * FROM DOKTER
WHERE spesialisasi = 'bedah');
```

Ketika berhadapan dengan tipe data timestamp yang mempunyai beberapa sub nilai, kita dapat menggunakan **EXTRACT**. Fungsi **EXTRACT** dapat mengembalikan sub nilai seperti hari, bulan, tahun, dan lainnya.

Contoh 8:

Misalkan kita ingin menampilkan daftar nama **PASIEN** yang lahir pada tahun 1990. Ini dapat dilihat dapat menggunakan **EXTRACT**.

```
SELECT nama
FROM PASIEN
WHERE EXTRACT(DECADE FROM tgl_lahir) = '199';
```

Perhatikan bahwa ketika ingin mengambil DECADE, kita harus membagi tahun lahir dengan 10 terlebih dahulu.

II. Operasi Join

Operasi join ada bermacam-macam. PostgreSQL mendukung operasi INNER/THETA JOIN (atau cukup JOIN saja), RIGHT OUTER JOIN, LEFT OUTER JOIN, FULL OUTER JOIN, NATURAL JOIN, dan CROSS JOIN.

1. CROSS JOIN

Merupakan jenis JOIN yang digunakan untuk mendapatkan data kombinasi dari tua tabel atau lebih.

Contoh 9:

Misalkan kita ingin menampilkan nama dari pasien yang mana sedang menggunakan kamar 'Merak 1'. Karena tabel **PASIEN** saja tidak cukup, maka kita juga membutuhkan tabel **KAMAR** dan **RAWAT INAP**. Sehingga kita dapat menggunakan CROSS JOIN sebagai berikut:

```
SELECT P.nama
FROM PASIEN P, KAMAR K, RAWAT_INAP R
WHERE R.id_pasien=P.id_pasien AND
      R.id_kamar=K.id_kamar AND
      K.nama='Merak 1';
```

Contoh 10:

Misalkan, n = jumlah baris data pada tabel di sebelah kiri, dan m = jumlah baris data pada tabel di sebelah kanan. Maka hasil jumlah baris dari CROSS JOIN adalah n X m baris data. Operasi ini sama dengan menggunakan tanda ',' (koma) untuk menggabungkan dua atau lebih tabel.

```
SELECT *
FROM PERAWAT CROSS JOIN SHIFT_PERAWAT;
```

2. JOIN / INNER JOIN

Merupakan jenis JOIN yang digunakan untuk mendapatkan data dari dua tabel atau lebih yang persis saling berelasi (sesuai dengan foreign key).

Contoh 11:

Misalkan kita ingin melihat seluruh informasi dari **PERAWAT** yang memiliki **SHIFT_PERAWAT** yang sesuai. Sehingga kita dapat menggunakan INNER JOIN sebagai berikut:

```
SELECT *
FROM PERAWAT P JOIN SHIFT_PERAWAT S
ON P.id_perawat = S.id_perawat;
```

3. LEFT JOIN / LEFT OUTER JOIN

Merupakan jenis JOIN yang digunakan untuk mendapatkan data dari dua tabel atau lebih dimana data di tabel sebelah kiri ditampilkan semua baik yang berelasi dengan data di tabel sebelah kanan maupun tidak, sebab berpotensi menghasilkan nilai null pada tabel sebelah kanan.

Contoh 12:

Misalkan kita ingin menampilkan daftar semua **PERAWAT** dengan **SHIFT_PERAWAT** yang sesuai, tanpa memedulikan apakah **PERAWAT** tersebut memiliki shift atau tidak. Oleh karena itu, kita dapat menggunakan LEFT OUTER JOIN sebagai berikut:

```
SELECT *
FROM PERAWAT P LEFT OUTER JOIN SHIFT_PERAWAT S
ON P.id_perawat = S.id_perawat;
```

4. RIGHT JOIN / RIGHT OUTER JOIN

Merupakan jenis JOIN yang digunakan untuk mendapatkan data dari dua tabel atau lebih dimana data di tabel sebelah kanan ditampilkan semua baik yang berelasi dengan data di tabel sebelah kiri maupun tidak, sebab berpotensi menghasilkan nilai null pada tabel sebelah kiri.

Contoh 13:

```
SELECT *
FROM SHIFT_PERAWAT S RIGHT OUTER JOIN PERAWAT P
ON P.id_perawat = S.id_perawat;
```

5. FULL JOIN / FULL OUTER JOIN

Merupakan jenis JOIN yang digunakan untuk mendapatkan data dari dua tabel atau lebih dimana data di tabel sebelah kanan dan kiri ditampilkan semua baik yang saling berelasi ataupun tidak, sebab berpotensi menghasilkan nilai null pada tabel sisi kanan dan kiri.

Contoh 14:

Misalkan kita ingin menampilkan daftar semua **OBAT** dengan **PEMBERIAN_OBAT** yang sesuai, tanpa memedulikan apakah **OBAT** tersebut pernah diberikan kepada pasien atau resep dibuat tanpa obat . Oleh karena itu, kita dapat menggunakan LEFT OUTER JOIN sebagai berikut:

```
SELECT *
FROM OBAT O FULL OUTER JOIN PEMBERIAN_OBAT PO
ON O.id_obat = PO.id_obat;
```

6. NATURAL JOIN

Merupakan jenis JOIN yang mendapatkan data dari dua tabel atau lebih berdasarkan nama column yang sama pada seluruh tabel yang kita JOIN.

Contoh 15:

Sebagai contoh, kita dapat mendapatkan hasil yang sama dari contoh 11 melalui perintah SQL berikut.

```
SELECT *
FROM PERAWAT NATURAL JOIN SHIFT_PERAWAT;
```

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

III. Advanced SQL

Seperti yang sudah dipelajari sebelumnya, advanced SQL merupakan tahap lanjut dari basic SQL. Dengan advanced SQL, banyak perintah kompleks yang bisa dilakukan untuk menampilkan data yang diinginkan dari database.
Untuk melakukan pengecekan apakah suatu nilai atribut bernilai **NULL** atau tidak kita bisa gunakan **IS / IS NOT NULL.**

Contoh 16:

Misalkan kita ingin mendapatkan daftar id_pasien dari pasien yang telah keluar dari rumah sakit. Oleh sebab itu, tanggal keluar tidak boleh NULL. Sehingga kita dapat melakukan sebagai berikut: Perhatikan bahwa hasil tersebut mungkin tidak unik.

```
SELECT id_pasien
FROM RAWAT_INAP
WHERE tgl_keluar IS NOT NULL;
```

Perhatikan bahwa data mungkin tidak unik

Contoh 17:

Dengan menggunakan ide yang sama, misalkan kita ingin menampilkan daftar id_pasien yang belum keluar dari rumah sakit dapat diperoleh sebagai berikut.:

```
SELECT id_pasien
FROM RAWAT_INAP
WHERE tgl_keluar IS NULL;
```

Pada beberapa kasus, dibutuhkan nilai pada database yang akan digunakan sebagai perbandingan. Untuk menyelesaikan kasus seperti ini, akan digunakan *nested query*. Sebuah *query* disebut *nested query* apabila terdapat *query* lagi di dalamnya. Biasanya terletak pada kondisi **WHERE**.
Dalam hal ini, kita bisa menggunakan operator **IN/NOT IN**. Operator IN/NOT IN akan membandingkan nilai v dengan sekumpulan nilai yang ada di himpunan V. IN akan menghasilkan nilai TRUE apabila v adalah salah satu elemen dari V sedangkan NOT IN sebaliknya.

Contoh 18:

Misalkan kita ingin menampilkan daftar nama obat yang diresepkan oleh 'Mitchell Greger'. Query yang kita dapat lakukan adalah sebagai berikut.

```
SELECT O.nama
FROM OBAT O
WHERE O.id_obat IN (
     SELECT PO.id_obat
  FROM PEMBERIAN_OBAT PO, SHIFT_PERAWAT SP, PERAWAT
P
  WHERE PO.id_shift_perawat = SP.id_shift_perawat
AND
        SP.id_perawat = P.id_perawat AND
        P.nama = 'Mitchell Greger'
);
```

Selain itu, kita juga bisa menggunakan operator **EXISTS / NOT EXISTS.** Berbeda dari IN/NOT IN, operator ini harus digunakan pada *query* bersifat *correlated query* yaitu kondisi WHERE-clause pada *nested query* berelasi dengan atribut pada relasi yang dideklarasikan pada outer *query*.

Contoh 19:

Misalnya kita ingin menampilkan nama dan id perawat yang tidak pernah mengobati pasien. Sehingga kita dapat melakukan sebagai berikut:

```
SELECT P.id_perawat, P.nama
FROM PERAWAT P
WHERE NOT EXISTS (
  SELECT *
  FROM PEMBERIAN_OBAT PO, SHIFT_PERAWAT SP
  WHERE PO.id_shift_perawat=SP.id_shift_perawat AND
      SP.id_perawat=P.id_perawat
);
```

IV. Fungsi Aggregate, Grouping and Having Clause

Kita bisa menggunakan fungsi aggregate dalam *query* SQL antara lain COUNT, SUM, MAX, MIN, dan AVG.

Contoh 20:

Misalkan kita ingin menampilkan semua kamar yang memiliki **harga terendah**. Sehingga dapat menggunakan **MIN** sebagai berikut:

```
SELECT min(harga)
FROM KAMAR;
```

Contoh 21:

Misalkan kita ingin menampilkan semua kamar yang memiliki **harga terbesar**. Sehingga dapat menggunakan **MAX** sebagai berikut:

```
SELECT max(harga)
FROM KAMAR;
```

Contoh 22:

Misalkan kita ingin menampilkan **jumlah** kamar yang tercatat dalam *database*. Sehingga dapat menggunakan **COUNT** sebagai berikut:

```
SELECT COUNT(*)
FROM KAMAR;
```

Contoh 23:

Misalkan kita ingin menampilkan **total jumlah** kapasitas dari seluruh kamar yang tercatat dalam database. Sehingga dapat menggunakan **SUM** sebagai berikut:

```
SELECT sum(kapasitas)
FROM KAMAR;
```

Contoh 24:

Misalkan, kita ingin menampilkan **rata-rata** harga dari kamar yang tercatat di database. Sehingga dapat menggunakan **AVG** sebagai berikut:

```
SELECT AVG(harga)
FROM KAMAR;
```

Kita bisa menggunakan fungsi **GROUP BY** untuk mengelompokkan dengan atribut yang muncul pada SELECT-clause. Semua atribut yang muncul di select harus diletakkan juga di GROUP BY atau diaplikasikan pada fungsi aggregate yang lain.

Contoh 25:

Misalkan kita ingin menampilkan jumlah kapasitas pada setiap jenis kamar.

```
SELECT jenis, SUM(kapasitas)
FROM KAMAR
GROUP BY jenis;
```

Jika kita membutuhkan *query* untuk menampilkan hanya hasil yang memenuhi kondisi tertentu, kita dapat menggunakan klausa HAVING.

Contoh lainnya, misalkan kita ingin mengetahui banyaknya pasien yang lahir pada tiap tahunnya. Kita bisa menjalankan perintah berikut;

```
SELECT EXTRACT(YEAR FROM P.tgl_lahir) as TAHUN,
COUNT(*)
FROM PASIEN P
GROUP BY EXTRACT (YEAR FROM P.tgl_lahir);
```

Contoh 26:

Misalkan kita ingin menampilkan beberapa kali kamar telah digunakan berdasarkan tipenya hanya jika kamar tersebut telah digunakan minimal 5 kali.

```
SELECT K.jenis, COUNT(K.id_kamar)
FROM KAMAR K, RAWAT_INAP RI
WHERE RI.id_kamar = K.id_kamar
GROUP BY K.jenis
HAVING COUNT(K.id_kamar) >= 5;
```

Tutorial 2 PostgreSQL
Basis Data
CSGE602070
Semester Gasal 2022/2023

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

Latihan Soal 1

Pada latihan ini, soal dengan tanda [SQL] harus dijalankan pada database masing-masing dan sertakan screenshoot-nya (berupa SQL dan hasilnya) pada laporan. Sedangkan soal dengan tanda [TRIVIA], tuliskan langsung jawaban Anda pada laporan sesuai dengan apa yang diminta pada masing-masing soal.

1. **[SQL]** Jalankan SQL Query pada Contoh 1 hingga Contoh 26 di atas dan cantumkan hasilnya pada laporan.
2. **[SQL]** Tampilkan nama obat yang telah diresepkan oleh perawat wanita dengan menggunakan keyword **IN**.
3. **[SQL]** Tampilkan nama UNIK obat yang **setidaknya** telah diberikan pada pasien.
4. **[SQL]** Tampilkan daftar dokter yang tidak pernah merawat pasien rawat inap.
5. **[SQL]** Tampilkan nama dokter dengan total jumlah pasien rawat inap yang telah ditugaskan kepada dokter tersebut diurutkan berdasarkan ascending alphabetical order (A-Z) dari namanya, tanpa peduli jika dokter tersebut memiliki pasien rawat inap atau tidak.
6. **[SQL]** Tampilkan jumlah pasien yang lahir pada setiap bulan. Anda disarankan menggunakan EXTRACT
7. **[SQL]** Tampilkan **jenis kamar** dan **harga rata-rata** kamar untuk setiap jenis kamar.
8. **[SQL]** Tampilkan **nama pasien, id** kamar, dan **nama dokter yang bertugas** untuk setiap pasien yang masih dirawat di rumah sakit.
9. **[SQL]** Tampilkan **tanggal masuk** dan **tanggal keluar** setiap pasien yang namanya tidak mengandung huruf E (tidak case sensitive).
10. **[SQL]** Tampilkan **nama** dan **nomor telepon** pasien yang tidak pernah menjadi pasien rawat inap.
11. **[SQL]** Tampilkan daftar pasien yang pernah menjadi pasien rawat inap di kamar jenis VIP atau VVIP. Anda harus menggunakan keyword **UNION.**
12. **[SQL]** Tampilkan **nama** dan **jenis kelamin** perawat yang merawat semua pasien wanita.
13. **[Trivia]** Apakah kita mungkin mendapatkan data tertentu dari operasi inner join menggunakan keyword **IN**?

V. View

VIEW merupakan sebuah tabel virtual yang dibuat menggunakan SQL *query*. *View* tidak disimpan dalam *database* seperti halnya *base relation*. Berikut ini sintaks untuk dapat membuat sebuah *view*.

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

Contoh 27:

Untuk membuat *view* yang dapat menampilkan daftar dokter beserta nama dan IDnya, kita bisa menjalankan perintah berikut.

```
CREATE VIEW daftar_dokter AS
SELECT id_dokter, nama
FROM dokter;
```

Setelah *view* berhasil dibuat, kita dapat melakukan *query* menggunakan *view* tersebut seperti halnya *query* pada *base relation*.

Contoh 28:

Misalnya, perintah *select* ke *view* daftar_dokter sebagai berikut.

```
SELECT * FROM daftar_dokter;
```

Contoh 29:

View biasanya hanya untuk penyimpanan sementara. Untuk menghapus view daftar dokter yang telah dibuat sebelumnya, kita bisa menggunakan perintah berikut.

```
DROP VIEW daftar_dokter;
```

UNIVERSITAS INDONESIA
Veritas, Probitas, Iustitia
FACULTY OF COMPUTER SCIENCE

VI. Indexing

Dalam sebuah basis data, *index* bertujuan untuk melakukan pencarian, terutama pada tabel dengan jumlah data yang sangat banyak sehingga saat kita mencari sebuah data berdasarkan kolom tertentu, data tersebut akan lebih mudah untuk ditemukan. Secara default, *primary key* merupakan sebuah *index* dalam basis data yang diinisiasi sebagai kolom kunci dalam sebuah tabel.

Pembuatan *index* dapat dilakukan dengan format sintaks berikut.

```
CREATE [ UNIQUE ] INDEX [ CONCURRENTLY ] [ name ] ON
table [ USING method ] ( { column | ( expression ) }
[ COLLATE collation ] [ opclass ] [ ASC | DESC ]
[ NULLS { FIRST | LAST } ] [, ...] )
[ WITH ( storage_parameter = value [, ... ] ) ]
[ TABLESPACE tablespace ] [ WHERE predicate ];
```

NOTES:

Tanda "[]" menyatakan pilihan, boleh tidak digunakan.
Tanda "|" menyatakan pilihan yang dapat digunakan.
Keterangan lebih lanjut ada pada tautan berikut.

Contoh 30:

Kita bisa mencoba membuat *index* untuk tabel **KAMAR** berdasarkan nama kamar. Jika tidak memberi spesifikasi apapun, index yang dibuat akan menggunakan *method* btree dan diurutkan secara *ascending*.

```
CREATE INDEX index_jenis_kamar ON kamar(jenis);
```

Contoh 31:

Mari kita buat *index* untuk tabel **PASIEN**. Pada contoh ini, kita menggunakan *method* hash.

```
CREATE INDEX index_nama_pasien
ON pasien USING HASH (nama);
```

Index komposit adalah *index* dengan dua atau lebih kolom dalam suatu tabel.

Contoh 32:

Kali ini, kita buat index komposit untuk tabel **PERAWAT** berdasarkan id_perawat dan nama yang diurutkan secara *descending*.

```
CREATE INDEX index_perawat
ON perawat (id_perawat, nama DESC);
```

Untuk satu tabel, kita dapat memiliki lebih dari satu *index*.

Contoh 33:

Sebagai contoh, berikut ini dua *index* yang berbeda untuk tabel **DOKTER**.

```
CREATE INDEX index_nama_dokter
ON dokter (nama);

CREATE INDEX index_spesialisasi_dokter
ON dokter (spesialisasi);
```

Untuk melihat *index* yang ada pada suatu tabel, kita dapat menjalankan perintah berikut.

```
\d table_name;
```

Untuk menghapus *index* yang telah dibuat sebelumnya, kita dapat menjalankan perintah berikut.

```
DROP INDEX [ CONCURRENTLY ] [ IF EXISTS ] name [, ...]
[ CASCADE | RESTRICT ]
```

Contoh 34:

Sebagai contoh, perintah berikut akan menghapus index nama dokter yang telah dibuat sebelumnya dari tabel **DOKTER**.

```
DROP INDEX index_nama_dokter;
```

VII. Explain

EXPLAIN adalah perintah yang dapat digunakan **untuk menampilkan estimasi eksekusi dari sebuah query**. Berikut ini cara menggunakannya.

```
EXPLAIN [ ANALYZE ] query;
```

Bagian terpenting dari data yang ditampilkan adalah ***execution cost*** yang merupakan **estimasi waktu yang diperlukan untuk menjalankan query** tersebut. Adanya opsi **ANALYZE** membuat query dapat dijalankan. Hal ini berguna untuk **membandingkan waktu estimasi dengan waktu sebenarnya**.

Contoh 35:

Sebagai contoh, jalankan perintah berikut.

```
EXPLAIN ANALYZE
SELECT *
FROM PASIEN
WHERE jenis_kelamin = 'L';
```

Di bawah ini merupakan hasil eksekusi *query* di atas menggunakan db.cs.ui.ac.id. *Execution time* menunjukkan waktu yang dibutuhkan untuk mengeksekusi *query* tersebut.

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Seq Scan on pasien  (cost=0.00..1.63 rows=1 width=464) (actual time=0.013..0.020 rows=25 loops=1)
   Filter: (jenis_kelamin = 'L'::bpchar)
   Rows Removed by Filter: 25
 Planning Time: 0.077 ms
 Execution Time: 0.033 ms
(5 rows)
```

Latihan Soal 2

Pada latihan ini, soal dengan tanda [SQL] harus dijalankan pada database masing-masing dan sertakan screenshoot-nya (berupa SQL dan hasilnya) pada laporan. Sedangkan soal dengan tanda [TRIVIA], tuliskan langsung jawaban Anda pada laporan sesuai dengan apa yang diminta pada masing-masing soal.

1. **[SQL]** Jalankan SQL Query pada Contoh 27 hingga Contoh 35 di atas dan cantumkan hasilnya pada laporan.
2. **View**
   a. **[Trivia]** Apa yang akan terjadi jika kita membuat View menggunakan nama yang sama dengan nama tabel yang ada pada database? Jelaskan!
   b. **[Trivia]** Apa fungsi TEMP atau TEMPORARY di View?
   c. **[SQL]** Buatlah View yang menyimpan **nama** beserta **durasi bekerja** yang dilakukan oleh perawat.
   d. **[SQL]** Buatlah View yang menyimpan **nama-nama perawat** yang merawat pasien dikelompokkan berdasarkan id pasien dan **nama pasien** yang belum keluar dari rumah sakit (HINT: string_agg)
3. **Indexing and Analyze**

   Diberikan query berikut.

```
SELECT * FROM kamar ORDER BY harga DESC;
```

```
SELECT * FROM rawat_inap WHERE tgl_keluar IS
NULL;
```

```
SELECT * FROM perawat WHERE nama LIKE 'T%';
```

```
SELECT * FROM pasien ORDER BY alamat LIMIT 10;
```

a. **[SQL]** Jalankan perintah EXPLAIN ANALYZE untuk setiap *query* di atas. *Screenshot* eksekusinya dan tulis hasilnya pada tabel di bawah, sertakan dalam laporan submisi Anda.

b. **[SQL]** Buat *index* berikut (*method* nya terserah Anda):
i. index_harga_kamar pada tabel **KAMAR** kolom **harga**.
ii. index_tgl_keluar_rawat_inap pada tabel **RAWAT_INAP** kolom **tgl_keluar**.
iii. index_nama_perawat pada tabel **PERAWAT** kolom **nama**.
iv. index_alamat_pasien pada tabel **PASIEN** kolom **alamat**.

Tampilkan *query* dan *index* untuk setiap tabel di dalam laporan submisi Anda.

c. **[SQL]** Jalankan kembali **setiap *query* SELECT** di atas dari pertanyaan nomor 3 menggunakan perintah EXPLAIN ANALYZE. *Screenshot* eksekusinya dan tulis hasilnya pada tabel di bawah, sertakan dalam laporan submisi Anda.

d. **[Trivia]** Bandingkan *planning time* dan *execution time* (menggunakan tabel di bawah) dari *query* saat tanpa index dan setelah menggunakan *index*. Mana yang lebih baik? Berikan penjelasan!

Query	***Planning Time***		***Execution Time***	
	TANPA INDEX	**DENGAN INDEX**	**TANPA INDEX**	**DENGAN INDEX**
1				
2				
3				

4				