CSGE602070 Basis Data
Semester Genap 2023/2024
Tutorial 2
Advanced SQL & Indexing

Deadline: 20 April 2024 23:55 WIB (Waktu SCeLE)

I. Basic SQL

Pada tutorial I kita telah membuat database **SIMART**, untuk mendapatkan data yang ada di dalam sebuah tabel tertentu, kita 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 kombinasi yang ada (*cartesian product*) karena kondisi tiap hasil dianggap **TRUE**. Kondisi **WHERE** sering digunakan untuk melakukan penyaringan data.

Contoh 1:
Misalkan kita ingin menampilkan nama PRODUK dengan id_produk 'PRD014'. Maka gunakan perintah SQL sebagai berikut:

```
SELECT NAMA
FROM PRODUK
WHERE ID_PRODUK = 'PRD014';
```

Untuk menampilkan hasil *query* secara terurut, kita dapat menambahkan klausa **ORDER BY** diikuti oleh kolom yang menjadi basis pengurutan dan metode pengurutannya, **ASC** (*Ascending*) untuk mengurutkan **dari kecil ke besar (1, 2, 3)** atau **DESC** (*Descending*) untuk **sebaliknya (3, 2 ,1)**.

Contoh 2:
Misalkan, kita ingin menampilkan nama, stok, dan harga dari tabel **PRODUK** terurut berdasarkan jumlah stok tersedia yang paling banyak. Kita bisa gunakan perintah SQL sebagai berikut:

```
SELECT nama, stok, harga
```

```
FROM produk
ORDER BY stok DESC;
```

Kita juga bisa menggunakan **operasi aritmatika (+, -, /, *)** dan **logika (AND, OR, NOT)** dalam *query* SQL. Operasi logika hanya bisa digunakan pada klause **WHERE**.

Contoh 3:

Misalkan kita ingin menampilkan nama produk yang memiliki harga di antara 1,000,000 dan 2,000,000 (inklusif). Kita bisa gunakan perintah sebagai berikut.

```
SELECT nama
FROM produk
WHERE harga >= 1000000 and harga <= 2000000;
```

Kita juga bisa menggunakan operasi **BETWEEN** dan mendapatkan hasil yang sama. Berikut contohnya.

```
SELECT nama
FROM produk
WHERE (harga BETWEEN 1000000 AND 2000000);
```

Untuk melakukan pengecekan terhadap pola string sederhana kita bisa menggunakan operator **LIKE** dan tanda '_' (*Wildcard* 1 karakter) atau tanda ('%') (*Wildcard* 1 atau lebih karakter).

Contoh 4:
Misalkan, kita ingin melihat produk yang 3 huruf terakhirnya diawali V2. Kita bisa menggunakan perintah SQL seperti berikut.

```
SELECT nama
FROM produk
WHERE nama LIKE '%V2_';
```

Kita juga bisa mengoperasikan hasil dua atau lebih *query* yang berbeda dengan menggunakan operasi **UNION**, **INTERSECT**, dan **EXCEPT** (perlu diperhatikan jumlah kolom antar *query* harus sama).

Contoh 5:
Misalkan kita ingin melihat seluruh kurir yang mengurus transaksi yang sedang diproses atau transaksi yang sedang dikirim. Berikut kode yang bisa digunakan.

```
(SELECT id_kurir
FROM TRANSAKSI
WHERE status = 'Diproses')
UNION
(SELECT id_kurir
FROM TRANSAKSI
WHERE status = 'Dikirim');
```

Contoh 6:
Misalkan kita ingin melihat id pembeli yang pernah melakukan pembayaran menggunakan transfer bank dan kartu kredit. Berikut kode yang bisa digunakan.

```
(SELECT id_pembeli
FROM PEMBAYARAN
WHERE metode_pembayaran = 'Transfer Bank')
INTERSECT
(SELECT id_pembeli
FROM PEMBAYARAN
WHERE metode_pembayaran = 'Kartu Kredit');
```

Contoh 7:
Misalkan kita ingin melihat id pembayaran dari pembayaran dengan nominal 500,000 tapi tidak menggunakan kartu kredit. Berikut kode yang bisa digunakan.

```
(SELECT id_pembayaran
FROM pembayaran
```

```
WHERE nominal >= 500000)
EXCEPT
(SELECT id_pembayaran
FROM pembayaran
WHERE metode_pembayaran = 'Kartu Kredit');
```

Contoh 8:

Berikut adalah contoh apabila kita melakukan Union tapi operand memiliki kolom yang berbeda, sehingga akan menghasilkan **error**.

```
(SELECT id_pembayaran, id_pembeli
FROM pembayaran
WHERE nominal >= 500000)
EXCEPT
(SELECT id_pembeli
FROM pembayaran
WHERE metode_pembayaran = 'Kartu Kredit');
```

Ketika berhadapan dengan tipe data timestamp yang mempunya beberapa *sub-value*, kita dapat menggunakan **EXTRACT**. Fungsi EXTRACT dapat mengembalikan *sub-value* seperti hari, bulan, tahun, dan lainnya. Secara lengkap bisa lihat di dokumentasi.

Contoh 9:

Misalkan kita ingin menampilkan id_pembayaran dari pembayaran yang dilakukan setelah quartal 2. Berikut kode yang bisa digunakan.

```
SELECT id_pembayaran
FROM PEMBAYARAN
WHERE EXTRACT(MONTH FROM tanggal_pembayaran) > 6;
```

FAKULTAS ILMU KOMPUTER

CSGE602070 Basis Data
Semester Genap 2023/2024
Tutorial 2
Advanced SQL & Indexing

II. Operasi Join

JOIN adalah salah satu operasi yang sangat berguna. Pada PostgreSQL, kita bisa melakukan **INNER/THETA JOIN, RIGHT OUTER JOIN, LEFT OUTER JOIN, FULL OUTER JOIN, NATURAL JOIN,** dan **CROSS JOIN**

1. **CROSS JOIN**
   Merupakan jenis JOIN yang digunakan untuk mendapatkan *cartesian product* dari dua tabel atau lebih

   Contoh 10:
   Misalkan **n** adalah **jumlah baris data** pada tabel di sebelah **kiri** dan **m** adalah **jumlah baris data** pada tabel di sebelah **kanan**. Maka hasil jumlah baris dari CROSS JOIN adalah $n \times m$ baris data.

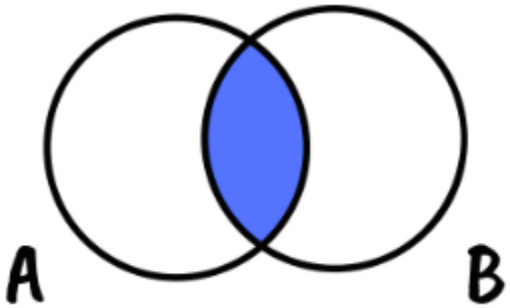
```
SELECT COUNT(*)
FROM PRODUK;

SELECT COUNT(*)
FROM BRAND;

SELECT COUNT(*)
FROM PRODUK, BRAND;
```

   Contoh 11:
   Misalkan kita ingin menampilkan nama dari pembeli yang pernah melakukan suatu pembayaran dengan nominal di atas 5,000,000. Perlu diperhatikan, tabel PEMBELI atau PEMBAYARAN saja tidak cukup, kita perlu menggabungkan keduanya. Sehingga kita bisa menggunakan CROSS JOIN sebagai berikut:

```
SELECT PEMBELI.NAMA, NOMINAL
FROM PEMBELI, PEMBAYARAN
WHERE PEMBELI.ID_PEMBELI = PEMBAYARAN.ID_PEMBELI
AND NOMINAL > 5000000;
```

CSGE602070 Basis Data
Semester Genap 2023/2024
Tutorial 2
Advanced SQL & Indexing

2. JOIN / INNER JOIN



Gambar 1: Visualisasi Inner JOIN

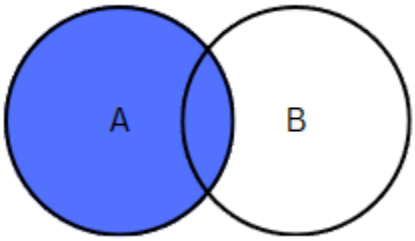
Merupakan jenis **JOIN** yang digunakan untuk mendapatkan data dari dua tabel atau lebih yang persis saling berelasi (sesuai dengan kondisi join).

Contoh 12:

Misalkan kita ingin melihat seluruh informasi dari transaksi suatu produk termasuk informasi mengenai produk dan transaksinya. Kita dapat menggunakan INNER JOIN seperti berikut.

```
SELECT *
FROM PRODUK P JOIN TRANSAKSI_PRODUK TP ON P.ID_PRODUK = TP.ID_PRODUK
JOIN TRANSAKSI T ON TP.ID_TRANSAKSI = T.ID_TRANSAKSI;
```

3. LEFT JOIN / LEFT OUTER JOIN



Gambar 2: Visualisasi LEFT JOIN

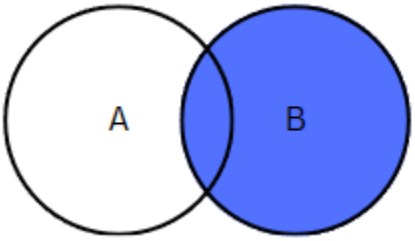
Merupakan jenis **JOIN** yang digunakan untuk mendapatkan data dari dua tabel atau lebih dimana data di tabel sebelah kiri ditampilkan semua baik yang berelasi dengan data di tabel sebelah kanan maupun tidak.

Contoh 13:

Misalkan kita ingin melihat status transaksi yang ada dan id_kurir yang ditugaskan pada transaksi itu. Kita dapat menggunakan LEFT JOIN seperti berikut:

```
SELECT STATUS, K.id_kurir
FROM TRANSAKSI T LEFT JOIN KURIR K ON T.ID_KURIR = K.ID_KURIR;
```

4. RIGHT JOIN / RIGHT OUTER JOIN



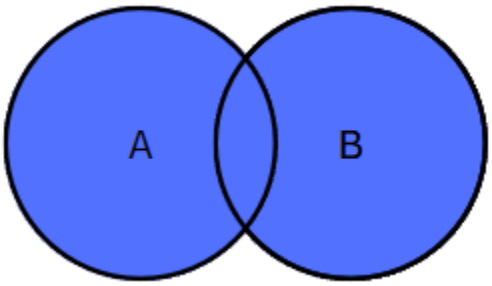
Gambar 3: Visualisasi RIGHT JOIN

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. Mirip seperti **LEFT JOIN**, bedanya **RIGHT JOIN** memasukkan semua data di sebelah kanan.

Contoh 14:
Misalkan kita ingin kebalikannya. Kita ingin seluruh id kurir dan status dari transaksi yang pernah diurusnya. Kita bisa mendapatkannya dengan kode berikut.

```
SELECT STATUS, K.id_kurir
FROM TRANSAKSI T RIGHT JOIN KURIR K ON T.ID_KURIR = K.ID_KURIR;
```

5. FULL JOIN / FULL OUTER JOIN



Gambar 4: Visualisasi FULL 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.

Contoh 15:
Misalkan kali ini kita ingin keduanya. Kita ingin menampilkan seluruh data transaksi dan data id_kurir. Kita bisa menampilkannya dengan kode berikut.

```
SELECT id_transaksi, K.id_kurir
FROM KURIR K FULL OUTER JOIN TRANSAKSI T ON K.ID_KURIR = T.ID_KURIR;
```

6. NATURAL JOIN

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

Contoh 16:
Apabila Anda perhatikan, pasangan-pasangan kolom join pada contoh 12 memiliki nama yang sama (sama-sama id_produk dan id_transaksi). Dengan demikian, kita dapat membuat querynya kembali hanya menggunakan natural join seperti berikut.

```
SELECT *
FROM PRODUK P
NATURAL JOIN TRANSAKSI_PRODUK TP
NATURAL JOIN TRANSAKSI T;
```

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 17:
Misalkan kita ingin menampilkan daftar id_transaksi dari rental yang telah menerima pembayaran. Oleh sebab itu, id_kurir tidak akan bernilai NULL. Sehingga kita dapat melakukan sebagai berikut:

```
SELECT id_transaksi
FROM TRANSAKSI
WHERE id_kurir IS NOT NULL;
```

Perhatikan bahwa hasil tersebut mungkin tidak unik.

Contoh 18:
Dengan menggunakan ide yang sama, misalkan kita ingin mendapatkan daftar id_transaksi dari rental yang masih dalam proses “Menunggu Pembayaran”. Oleh sebab itu, id_kurir akan bernilai NULL. Sehingga kita dapat melakukan sebagai berikut:

```
SELECT id_transaksi
```

```
FROM TRANSAKSI
WHERE id_kurir IS NULL;
```

Perhatikan bahwa hasil tersebut mungkin tidak unik.

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 19:
Misalkan kita ingin menampilkan nama produk yang memiliki status transaksi ‘Diproses’. Query yang kita dapat lakukan adalah sebagai berikut:

```
SELECT P.nama
FROM PRODUK P
WHERE P.id_produk IN (
       SELECT P.id_produk
       FROM PRODUK P, TRANSAKSI_PRODUK TP, TRANSAKSI T
       WHERE P.id_produk = TP.id_produk AND
       TP.id_transaksi = T.id_transaksi AND
       T.status = 'Diproses'
);
```

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 20:

Misalnya kita ingin menampilkan id_toko dan nama toko dengan transaksi yang masih belum ‘Selesai’. Sehingga kita dapat melakukan sebagai berikut:

```
SELECT T.id_toko, T.nama
FROM TOKO T
WHERE NOT EXISTS (
      SELECT *
      FROM TRANSAKSI TR
      WHERE TR.status = 'Selesai' AND
      TR.id_toko = T.id_toko
);
```

IV. Fungsi Aggregate, Grouping and Having Clause

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

Contoh 21:
Misalkan kita ingin menampilkan semua pembayaran yang memiliki **nominal terendah**. Sehingga dapat menggunakan **MIN** sebagai berikut:

```
SELECT min(nominal)
FROM PEMBAYARAN;
```

Contoh 22:
Misalkan kita ingin menampilkan semua pembayaran yang memiliki **nominal terbesar**. Sehingga dapat menggunakan **MAX** sebagai berikut:

```
SELECT max(nominal)
FROM PEMBAYARAN;
```

Contoh 23:
Misalkan kita ingin menampilkan **banyaknya** pembayaran yang tercatat dalam database. Sehingga dapat menggunakan **COUNT** sebagai berikut:

```
SELECT COUNT(*)
```

```
FROM PEMBAYARAN;
```

Contoh 24:
Misalkan kita ingin menampilkan **total nominal** pembayaran dari seluruh pembayaran yang tercatat dalam database. Sehingga dapat menggunakan **SUM** sebagai berikut:

```
SELECT sum(nominal)
FROM PEMBAYARAN;
```

Contoh 25:
Misalkan, kita ingin menampilkan **rata-rata** pembayaran dari seluruh pembayaran yang tercatat di database. Sehingga dapat menggunakan **AVG** sebagai berikut:

```
SELECT avg(nominal)
FROM PEMBAYARAN;
```

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 26:
Misalkan kita ingin menampilkan rata-rata harga produk pada setiap brand tertentu.

```
SELECT B.nama, AVG(harga)
FROM BRAND B, PRODUK P
WHERE B.id_brand = P.id_brand
GROUP BY B.id_brand;
```

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

Contoh 27:
Misalkan kita ingin menampilkan brand dan berapa banyak jenis produk yang berasal dari brand tersebut, serta hanya tampilkan brand yang memiliki minimal 4 jenis produk.

```
SELECT B.nama, COUNT(P.id_produk)
FROM BRAND B, PRODUK P
```

```
WHERE B.id_brand = P.id_brand
GROUP BY B.id_brand
HAVING COUNT(P.id_produk) >= 4;
```

Latihan Soal 1

1. **[SQL]** Jalankan SQL Query pada Contoh 1 hingga Contoh 27 di atas dan cantumkan hasilnya pada laporan
2. **[SQL]** Tampilkan nama brand yang memiliki panjang nama antara 3 hingga 7 karakter (inklusif). Anda wajib menggunakan LIKE.
3. **[SQL]** Tampilkan seluruh nama pengguna dan jumlah **nominal** pembayaran yang pernah dilakukannya. Jika tidak pernah melakukan pembayaran, tampilkan nominal 0. Urutkan dari yang paling banyak mengeluarkan uang untuk pembayaran. Apabila jumlah pembayaran sama, urutkan berdasarkan namanya dari A – Z.
4. **[SQL]** Tampilkan jumlah transaksi yang terjadi setiap bulan. Urutkan dari yang paling banyak terjadi transaksi.
5. **[SQL]** Tampilkan jumlah transaksi yang terjadi berdasarkan status transaksi dari toko yang beralamat di wilayah ‘Dki Jakarta’.
6. **[SQL]** Tampilkan rata-rata penjualan dari setiap toko.
7. **[SQL]** Tampilkan nama pembeli yang sudah menerima produk pada transaksi yang tercatat, yaitu yang memiliki status “Diterima Pembeli” **atau** “Selesai”.
8. **[SQL]** Tampilkan nama kurir dan jumlah pengiriman yang dilakukan (termasuk yang tidak pernah melakukan pengiriman).
9. **[SQL]** Tentukan metode pembayaran yang paling diminati oleh pembeli.
10. **[SQL]** Tentukan brand produk yang paling diminati oleh pembeli.

View

VIEW adalah tabel virtual yang dibuat menggunakan SQL *query* dan diturunkan dari tabel lain dalam skema. 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 28:
Misalnya, untuk membuat view yang dapat menampilkan nama toko beserta IDnya, kita bisa menjalankan *query* berikut

```
CREATE VIEW daftar_toko AS
SELECT id_toko, nama
FROM toko;
```

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

Contoh 29:
Misalnya, *query* select ke view daftar_toko sebagai berikut.

```
SELECT * FROM daftar_toko;
```

Contoh 30:
View biasanya hanya untuk penyimpanan sementara. Untuk menghapus view daftar_toko yang telah dibuat sebelumnya, kita bisa menjalankan *query* berikut

```
DROP VIEW daftar_toko;
```

FAKULTAS ILMU KOMPUTER

CSGE602070 Basis Data
Semester Genap 2023/2024
Tutorial 2
Advanced SQL & Indexing

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 sintaks bersifat opsional (boleh tidak
digunakan).
Tanda "|" menyatakan pilihan yang dapat digunakan.
```

Contoh 31:

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

```
CREATE INDEX index_status_transaksi ON transaksi(status);
```

Contoh 32:

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

```
CREATE INDEX index_nama_pembeli ON pembeli USING HASH (nama);
```

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

Contoh 33:

Kali ini, kita buat index komposit untuk tabel **KURIR** berdasarkan id_kurir dan nama_kurir yang diurutkan secara *ascending*.

```
CREATE INDEX index_kurir ON kurir (id_kurir, nama_kurir ASC);
```

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

Contoh 34:

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

```
CREATE INDEX index_nama_produk ON produk (nama);
CREATE INDEX index_id_brand_produk ON produk (id_brand);
```

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 35:

Sebagai contoh, perintah berikut akan menghapus beberapa index yang telah dibuat sebelumnya.

```
DROP INDEX index_status_transaksi;
DROP INDEX index_nama_pembeli;
DROP INDEX index_kurir;
DROP INDEX index_nama_produk;
DROP INDEX index_id_brand_produk;
```

Explain

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

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

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

Contoh 36:

Sebagai contoh, jalankan perintah berikut.

```
EXPLAIN ANALYZE
SELECT *
FROM KURIR
WHERE nama_perusahaan = 'JNE';
```

Berikut adalah hasil eksekusi *query* di atas. *Execution time* menunjukkan waktu yang dibutuhkan untuk mengeksekusi *query* tersebut

```
                                                QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Seq Scan on kurir  (cost=0.00..12.12 rows=1 width=452) (actual time=0.030..0.035 rows=6 loops=1)
   Filter: ((nama_perusahaan)::text = 'JNE'::text)
   Rows Removed by Filter: 14
 Planning Time: 0.111 ms
 Execution Time: 0.059 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 28 hingga Contoh 36 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 **id produk** dan **nama produk** beserta **total transaksi** pada produk tersebut. Jika suatu produk tidak memiliki transaksi, nama produk tetap ditampilkan, namun totalnya ditampilkan 0. Urutkan secara ascending berdasarkan id produk.
    d. **[SQL]** Buatlah View yang menyimpan **nama pembeli** (apabila lebih dari 1, pisahkan dengan koma) yang melakukan transaksi pada produk dan

dikelompokkan berdasarkan id brand dan **nama brand** dari produk. (HINT: string_agg)

3. **Indexing dan Analyze**
   Diberikan query berikut.

```
SELECT *
FROM PEMBELI
ORDER BY tanggal_lahir DESC;
```

```
SELECT *
FROM PRODUK
WHERE harga > 3000000;
```

```
SELECT *
FROM KURIR
WHERE nama_kurir
LIKE ‘W%’;
```

```
SELECT *
FROM TRANSAKSI
ORDER BY tanggal_transaksi
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_tanggal_lahir_pembeli** pada tabel **PEMBELI** kolom **tanggal_lahir**.
      ii. **index_harga_produk** pada tabel **PRODUK** kolom **harga**.
      iii. **index_nama_kurir** pada tabel **KURIR** kolom **nama_kurir**.

CSGE602070 Basis Data
Semester Genap 2023/2024
Tutorial 2
Advanced SQL & Indexing

iv. **index_tanggal_transaksi** pada tabel **TRANSAKSI** kolom **tanggal_transaksi**.

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				