JOBSHEET

B6SIS D6T6 Į6NJUT

Jurusan Teknologi Informasi

POLITEKNIK NEGERI MALANG

TAHUN AJARAN 2025/2026

PERTEMUAN 4

Index dan Optimasi Query pada PostgreSQL

Team Teaching Basis Data Lanjut:

- Candra Bella Vista, S.Kom., MT.
- Moch Zawaruddin Abdullah, S.ST., M.Kom.
- Yan Watequlis Syaifudin, ST., MMT., PhD.
- Yoppy Yunhasnawa, S.ST., M.Sc.

Mata Kuliah	:	Basis Data Lanjut
Program Studi	:	D4 – Teknik Informatika / D4 – Sistem Informasi Bisnis
Semester	:	3 (tiga)
Pertemuan ke-	:	4

MATERI 4: INDEX DAN OPTIMASI QUERY
Mata Kuliah: Basisdata Lanjut

Tujuan Pembelajaran

Setelah menyelesaikan materi ini, mahasiswa diharapkan mampu:

1. Memahami konsep dasar indeks dalam database relasional.
2. Mengidentifikasi jenis-jenis indeks yang tersedia di PostgreSQL.
3. Menyusun query untuk memeriksa indeks yang ada pada suatu tabel.
4. Membuat indeks baru untuk meningkatkan performa query.
5. Menghapus indeks yang tidak diperlukan agar database tetap efisien.
6. Menambahkan constraint unik dengan memanfaatkan indeks.

Konsep Dasar

Indeks dalam PostgreSQL berfungsi sebagai struktur data tambahan yang mempercepat pencarian, pengurutan, maupun filtering data pada tabel. Konsep utamanya meliputi:

1. **Definisi Index**: Struktur yang menyimpan pointer ke data tabel sehingga query dapat menemukan baris lebih cepat tanpa harus memindai seluruh tabel (sequential scan).
2. **Jenis Index**: PostgreSQL mendukung beberapa tipe indeks, seperti B-Tree (default), Hash, GiST, SP-GiST, GIN, dan BRIN, yang masing-masing sesuai untuk kasus tertentu.
3. **Pemeriksaan Index**: Indeks yang ada dapat diperiksa melalui katalog sistem (pg_indexes) atau perintah \d nama_tabel.
4. **Pembuatan Index**: Indeks dibuat dengan perintah CREATE INDEX, misalnya untuk mempercepat pencarian berdasarkan kolom tertentu.
5. **Penghapusan Index**: Indeks yang tidak diperlukan dapat dihapus dengan DROP INDEX.
6. **Constraint Unik**: Ketika menambahkan constraint UNIQUE, PostgreSQL secara otomatis membuat indeks untuk menjamin keunikan data.

Manfaat

Menguasai konsep dan penggunaan indeks memberikan manfaat penting dalam pengelolaan database, antara lain:

1. Performa Query yang Lebih Cepat: Query dengan kondisi WHERE, JOIN, atau ORDER BY dapat dieksekusi lebih efisien.
2. Efisiensi Sistem: Mengurangi beban sequential scan pada tabel besar sehingga server database lebih responsif.

3. **Integritas Data**: Constraint unik membantu memastikan tidak ada duplikasi data pada kolom yang seharusnya unik.
4. **Fleksibilitas Desain Database**: Dosen maupun mahasiswa dapat menyesuaikan indeks sesuai kebutuhan analisis dan pola query.
5. **Pemeliharaan Database**: Mengetahui kapan harus membuat atau menghapus indeks membantu menjaga keseimbangan antara kecepatan query dan overhead penyimpanan.

Petunjuk Praktikum: Index di PostgreSQL

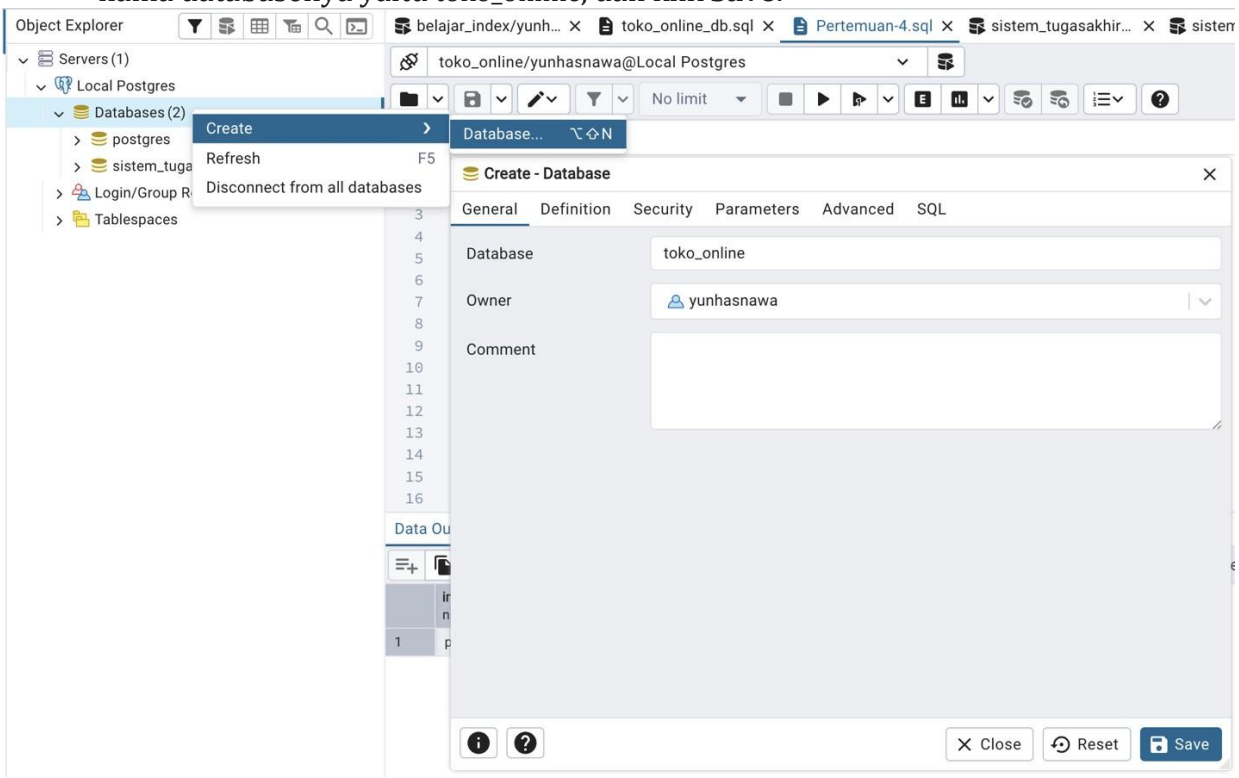
Penjelasan Topik-1 hingga Topik-4 terdapat pada slide materi presentasi.

Topik-5: Latihan Dasar-dasar Index & Penggunaannya

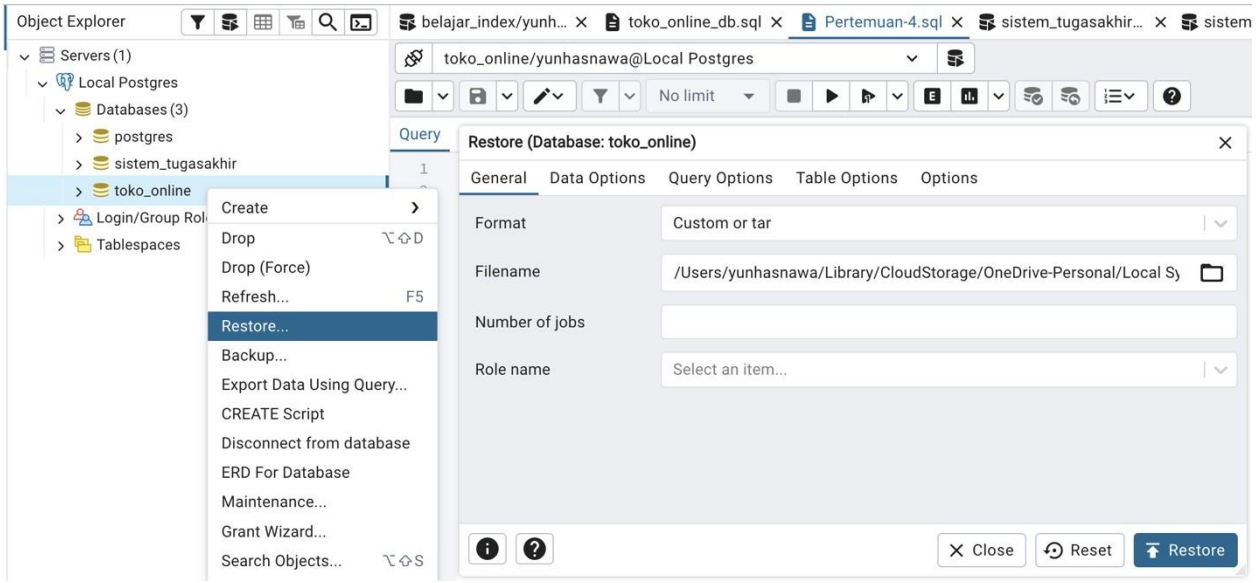
Pada bagian ini kita akan mempelajari bagaimana caranya membuat index dan mengamati dampak dari penggunaan index terhadap performa query pada suatu tabel. Untuk bisa mencoba petunjuk praktikum ini, pastikan Anda sudah memiliki database toko_online di server PostgreSQL Anda.

Jika Anda belum memiliki database tersebut, ikuti langkah berikut ini:

- Unduh file toko_online.backup yang sudah disediakan bersama petunjuk praktikum ini.
- Buat database baru di server PostgreSQL Anda dengan pgAdmin, dengan cara klik kanan ikon **Databases** di Object Explorer, lalu pilih menu **Create** -> **Database** . Lalu isikan nama databasenya yaitu toko_online, dan klik **Save**.



- Berikutnya, restore backup database dengan cara klik kanan pada database toko_online di bawah ikon **Databases** di Object Explorer. Lalu, pilih **Restore**.
- Pada kotak dialog yang muncul, pastikan pada kolom **Format** terisi Custom or tar. Dan pada kolom isian **Filename**, navigasikan ke tempat dimana file toko_online.backup berada, lalu klik **Restore**.



⚠️ **Catatan:** Jika pada saat restore terdapat error, biasanya error tersebut terkait dengan schema default public yang otomatis dibuat setiap kali kita membuat database baru. Anda dapat mengabaikan error tersebut jika di dalam database toko_online Anda sudah terdapat 3 tabel yaitu: pelanggan, produk, dan transaksi.

1. Membuat Index Dasar

Pertama-tama, cobalah jalankan sintaksis EXPLAIN ANALYZE berikuit ini dengan menggunakan Query Tool, lalu perhatikan hasilnya.

```
-- Query tanpa index
EXPLAIN ANALYZE
SELECT * FROM pelanggan WHERE email = 'pelanggan12345@mail.com';
```

Perhatikan hasilnya di baris pertama, di sana terdapat tulisan Seq Scan on. Ini berarti ketika query tersebut dijalankan, maka PostgreSQL akan melakukan pencarian secara sekuensial, berurutan baris demi baris. Hal ini disebabkan pada tabel tersebut tidak terdapat index pada kolom email.

Data Output | Messages | Notifications

Showing row

	QUERY PLAN text
1	Seq Scan on pelanggan (cost=0.00..1138.00 rows=1 width=49) (actual time=1.621..6.281 rows=1 loops=1)
2	Filter: ((email)::text = 'pelanggan12345@mail.com'::text)
3	Rows Removed by Filter: 49999
4	Planning Time: 0.074 ms
5	Execution Time: 6.299 ms

⚠️ **Catatan:** Perhatikan juga output pada bagian **Execution Time**, catat hasilnya.

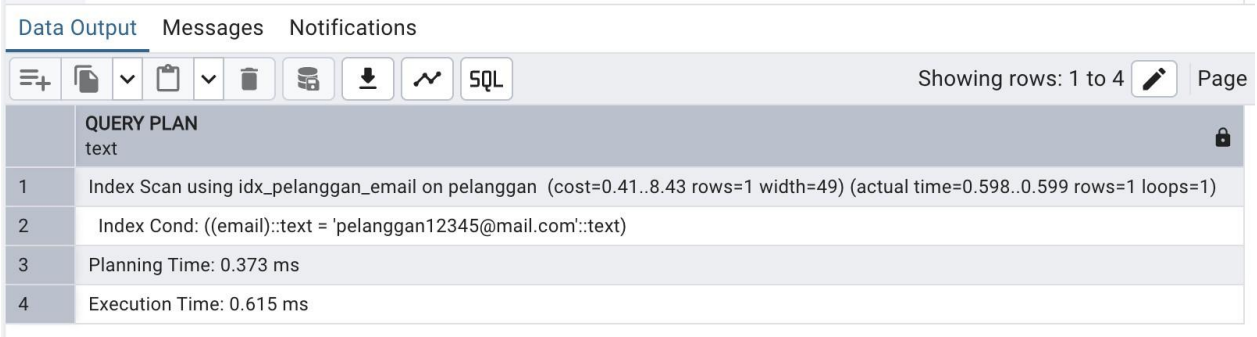
Selanjutnya, mari kita membuat index di kolom email dengan menggunakan sintaksis CREATE INDEX berikut ini.

```
-- Membuat index
CREATE INDEX idx_pelanggan_email ON pelanggan(email);
```

Sekarang jalankan lagi SQL Sebelumnya dan bandingkan hasil pada output yang dihasilkan.

```
-- Query ulang (lebih cepat dengan Index Scan)
EXPLAIN ANALYZE
SELECT * FROM pelanggan WHERE email = 'pelanggan12345@mail.com';
```

Perhatikan pada baris pertama, sekarang menjadi Index scan using…. Lalu, perhatikan juga execution time-nya sekarang menjadi jauh lebih cepat.



	QUERY PLAN text
1	Index Scan using idx_pelanggan_email on pelanggan (cost=0.41..8.43 rows=1 width=49) (actual time=0.598..0.599 rows=1 loops=1)
2	Index Cond: ((email)::text = 'pelanggan12345@mail.com'::text)
3	Planning Time: 0.373 ms
4	Execution Time: 0.615 ms

2. Index Tunggal vs Multi-Kolom

Terkadang, satu index saja pada tabel bisa jadi masih kurang mengoptimasi query yang kita jalankan. Sehingga tak jarang kita perlu membuat index di lebih dari satu kolom.

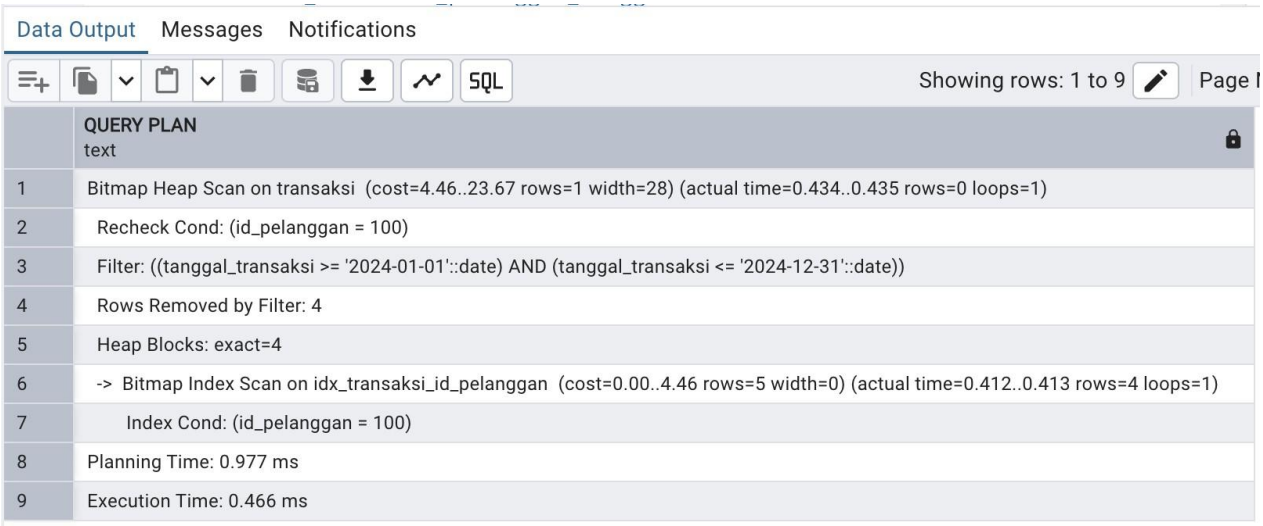
Mari kita coba membuat index tunggal di tabel transaksi pada kolom id_pelanggan.

```
-- Index tunggal pada id_pelanggan
CREATE INDEX idx_transaksi_id_pelanggan ON transaksi(id_pelanggan);
```

Selanjutnya jalankan sintaksis EXPLAIN ANALYZE berikut dan **catat execution time-nya**.

```
-- Query dengan 2 kondisi (kadang index tunggal tidak cukup)
EXPLAIN ANALYZE
SELECT * FROM transaksi
WHERE id_pelanggan = 100 AND tanggal_transaksi BETWEEN '2024-01-01' AND '2024-12-31';
```

Hasil EXPLAIN ANALYZE:



Data Output | Messages | Notifications

Showing rows: 1 to 9 | Page

	QUERY PLAN text
1	Bitmap Heap Scan on transaksi (cost=4.46..23.67 rows=1 width=28) (actual time=0.434..0.435 rows=0 loops=1)
2	Recheck Cond: (id_pelanggan = 100)
3	Filter: ((tanggal_transaksi >= '2024-01-01'::date) AND (tanggal_transaksi <= '2024-12-31'::date))
4	Rows Removed by Filter: 4
5	Heap Blocks: exact=4
6	-> Bitmap Index Scan on idx_transaksi_id_pelanggan (cost=0.00..4.46 rows=5 width=0) (actual time=0.412..0.413 rows=4 loops=1)
7	Index Cond: (id_pelanggan = 100)
8	Planning Time: 0.977 ms
9	Execution Time: 0.466 ms

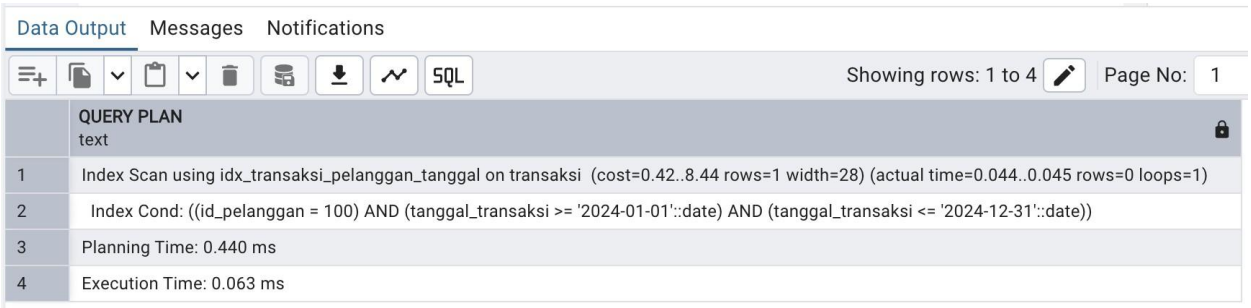
Sekarang mari kita buat satu lagi index berjenis multi-kolom pada tabel transaksi, di kolom id_pelanggan dan tanggal_transaksi sekaligus.

```
-- Index multi-kolom
CREATE INDEX idx_transaksi_pelanggan_tanggal
ON transaksi(id_pelanggan, tanggal_transaksi);
```

Berikutnya jalankan kembali SQL EXPLAIN ANALYZE yang sebelumnya, lalu perhatikan output-nya.

```
-- Query ulang (harus lebih efisien)
EXPLAIN ANALYZE
SELECT * FROM transaksi
WHERE id_pelanggan = 100 AND tanggal_transaksi BETWEEN '2024-01-01' AND '2024-12-31';
```

Perhatikan saat ini execution time dari query tersebut menjadi jauh lebih cepat.



Data Output | Messages | Notifications

Showing rows: 1 to 4 | Page No: 1

	QUERY PLAN text
1	Index Scan using idx_transaksi_pelanggan_tanggal on transaksi (cost=0.42..8.44 rows=1 width=28) (actual time=0.044..0.045 rows=0 loops=1)
2	Index Cond: ((id_pelanggan = 100) AND (tanggal_transaksi >= '2024-01-01'::date) AND (tanggal_transaksi <= '2024-12-31'::date))
3	Planning Time: 0.440 ms
4	Execution Time: 0.063 ms

3. Unique Index & Constraint

Untuk bisa mengoptimalkan query di database secara umum, dan di PostgreSQL khususnya, kita dianjurkan untuk memaksimalkan penggunaan fitur constraint pada tabel. Constraint adalah fitur yang 'membatasi' data seperti apa yang diinputkan pada tabel. Adapun beberapa constraint apabila diaktifkan pada tabel, akan secara otomatis membuat index juga. Sebagai contoh yaitu pada constraint UNIQUE.

Sekarang mari kita coba periksa apakah di tabel pelanggan sudah ada constraint UNIQUE. Untuk melakukannya, jalankan SQL berikut (menggunakan sintaksis CTE):

```
WITH cte AS (
  SELECT
    conrelid::regclass::text AS table_name,
    conname AS constraint_name,
    contype AS constraint_type,
    conindid::regclass AS index_name
  FROM
    pg_constraint
  WHERE
    contype IN ('p', 'u', 'f', 'c')
)
SELECT * FROM cte WHERE "table_name" = 'pelanggan';
```

Perhatikan outputnya, periksa di kolom constraint_name dan constraint_type. Apabila ada constraint SELAIN **pkey** (Primary Key) catat nama constraint-nya tersebut, dan hapus dengan menggunakan SQL berikut ini.

```
-- Hapus constraint
ALTER TABLE pelanggan DROP CONSTRAINT nama_constraint_anda CASCADE;
```

Selanjutnya periksa apakah di tabel pelanggan ada index-nya, dengan menggunakan SQL berikut ini.

```
-- Periksa apakah ada index tertentu
SELECT indexname, indexdef FROM pg_indexes WHERE tablename = 'pelanggan';
```

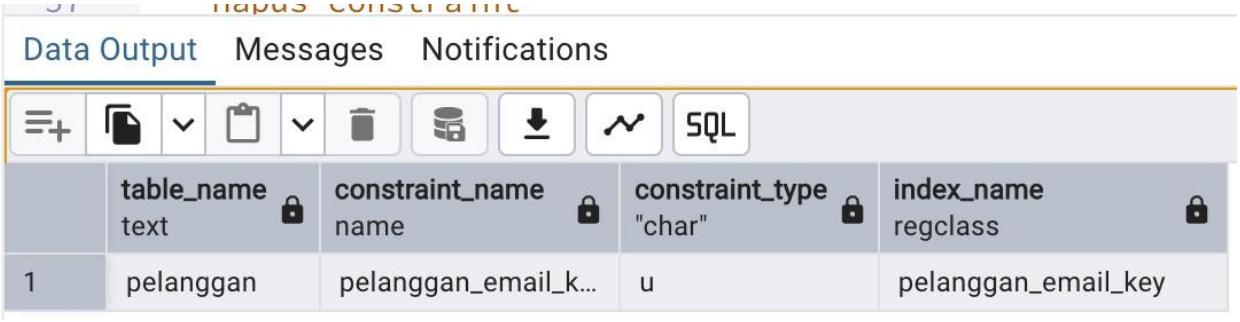
Perhatikan kolom indexname pada hasil yang ditampilkan. Jika terdapat index SELAIN pkey di sana, maka sama seperti pada CONSTRAINT tadi, hapuslah terlebih dahulu index yang ada dengan menggunakan perintah SQL berikut ini.

```
-- Apabila ada index-nya, hapus dulu:
DROP INDEX IF EXISTS nama_index_anda;
DROP INDEX IF EXISTS nama_index_anda_yang_lain_jika_ada;
```

Selanjutnya, tanpa membuat index secara manual, mari kita tambahkan constraint UNIQUE pada tabel pelanggan kita, di kolom email.

```
-- Membuat constraint.
ALTER TABLE pelanggan ADD CONSTRAINT pelanggan_email_key UNIQUE (email);
```

Hasil ADD CONSTRAINT:



⚠ **Catatan:** Pada SQL tersebut penamaan UNIQUE constraint mengikuti aturan (konvensi): <nama_tabel>_<nama_kolom>_<jenis_constraint>. Di PostgreSQL jenis CONSTRAINT-nya antara lain:

Jenis Constraint	Konvensi Penamaan
UNIQUE	_key
PRIMARY KEY	_pkey
FOREIGN KEY	_fkey
CHECK	_check
EXCLUSION	_excl

Setelah itu, cobalah untuk memeriksa keberadaan constraint yang baru saja Anda buat tersebut dengan menggunakan perintah SQL sebelumnya. Maka akan ditunjukkan hasil seperti berikut ini.

Setelah itu periksa juga keberadaan index pada tabel pelanggan, dengan menggunakan perintah pengecekan index yang sebelumnya. Maka akan terdapat satu UNIQUE INDEX yang dibuat secara **otomatis** di tabel tersebut.

Sekarang mari kita coba untuk memeriksa apakah memang benar ada index yang otomatis dibuat di tabel pelanggan kita tersebut.

```
-- Cek index:
EXPLAIN ANALYZE SELECT email FROM pelanggan WHERE email LIKE 'pelanggan12345@mail.com';
```

Perhatikan pada output yang dihasilkan, terdapat kata-kata: "**Index Only Scan**...", hal tersebut menandakan query-nya dieksekusi memang benar menggunakan index, dimana kita tidak pernah secara eksplisit membuat indeks tersebut. Sekarang, mari kita coba menginputkan data duplikat untuk menguji constraint UNIQUE yang kita buat.

```
-- Mencoba insert duplikat (akan gagal)
INSERT INTO pelanggan (nama, email, kota)
VALUES ('Pelanggan Baru', 'pelanggan12345@mail.com', 'Jakarta');
```

Hasilnya akan error, karena datanya tidak bisa diinputkan, disebabkan sudah ada data dengan email yang sama di dalam tabel kita.

Data Output | Messages | Notifications

```
ERROR:  duplicate key value violates unique constraint "idx_pelanggan_email_unique"
Key (email)=(pelanggan12345@mail.com) already exists.

SQL state: 23505
Detail: Key (email)=(pelanggan12345@mail.com) already exists.
```

4. Partial Index

Index juga bisa dibuat pada sebagian data saja alih-alih ke keseluruhan tabel. Index semacam ini biasanya berguna pada data yang berkaitan dengan waktu. Pada contoh berikut ini kita akan membuat index pada sebagian data di tabel transaksi.

Jalankan SQL berikut untuk membuat index hanya pada data yang masuk **setelah tanggal 1 Agustus 2025**.

```
-- Membuat index hanya untuk transaksi pada bulan Agustus ke belakang
CREATE INDEX idx_transaksi_recent
ON transaksi(tanggal_transaksi)
WHERE tanggal_transaksi > DATE '2025-08-01';
```

Cek performa query pada tanggal-tanggal yang masuk dalam batasan pembuatan index.

```
-- Query yang cocok dengan index
EXPLAIN ANALYZE
SELECT * FROM transaksi
WHERE tanggal_transaksi > DATE '2025-08-01';
```

Berikut ini adalah hasil dari EXPLAIN ANALYZE di atas, catat 'Execution Time'-nya.

Data Output | Messages | Notifications

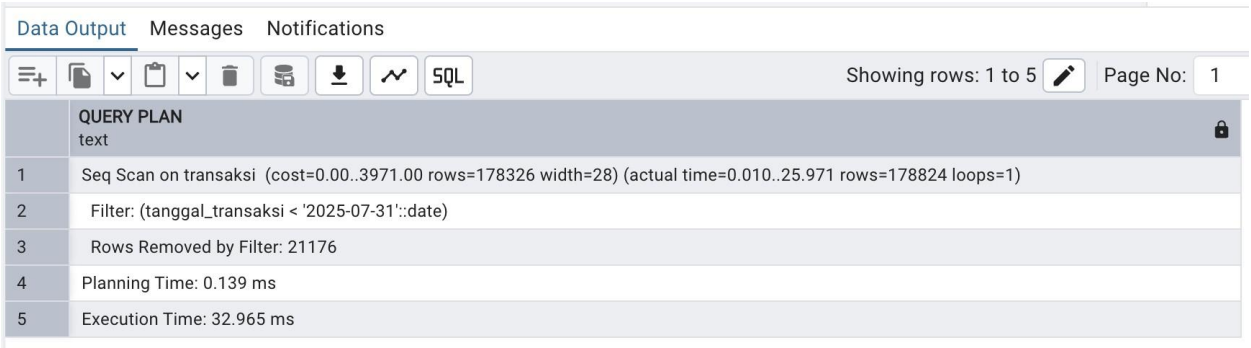
Showing rows: 1 to 6 | Page No: 1

	QUERY PLAN text
1	Index Only Scan using pelanggan_email_key on pelanggan (cost=0.41..8.43 rows=1 width=23) (actual time=0.156..0.157 rows=1 loops=1)
2	Index Cond: (email = 'pelanggan12345@mail.com'::text)
3	Filter: ((email)::text ~~ 'pelanggan12345@mail.com'::text)
4	Heap Fetches: 0
5	Planning Time: 0.179 ms
6	Execution Time: 0.173 ms

Bandingkan dengan query pada tanggal-tanggal yang tidak masuk dalam batasan pembuatan index, misalkan pada tanggal sebelum 31 Agustus 2025.

```
-- Query di luar jangkauan (index tidak digunakan)
EXPLAIN ANALYZE
SELECT * FROM transaksi
WHERE tanggal_transaksi < CURRENT_DATE - INTERVAL '200 days';
```

Perhatikan hasil query pada tanggal yang tidak tercakup dalam index, dimana waktu eksekusinya sangat lambat.


Data Output Messages Notifications

Showing rows: 1 to 5 Page No: 1

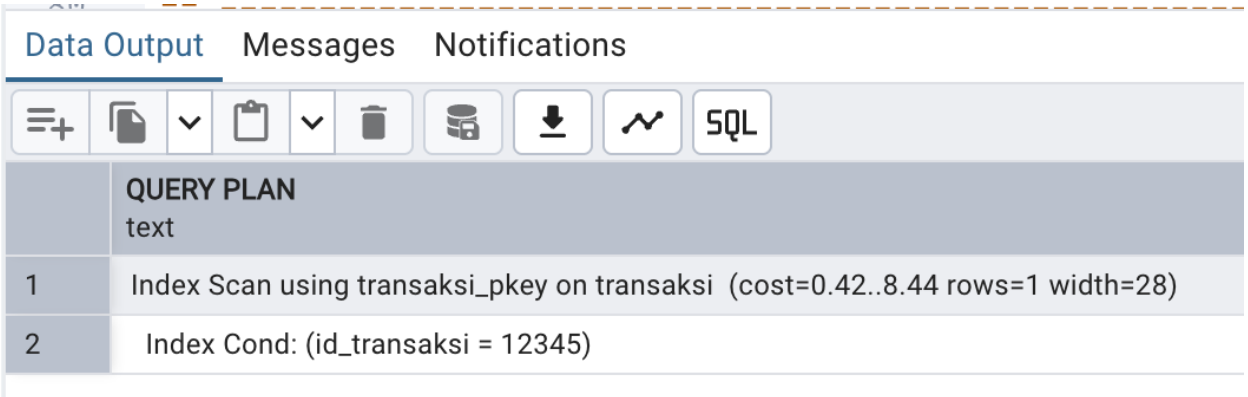
	QUERY PLAN text
1	Seq Scan on transaksi (cost=0.00..3971.00 rows=178326 width=28) (actual time=0.010..25.971 rows=178824 loops=1)
2	Filter: (tanggal_transaksi < '2025-07-31'::date)
3	Rows Removed by Filter: 21176
4	Planning Time: 0.139 ms
5	Execution Time: 32.965 ms

5. Membaca Query Plan

Terdapat perbadaan pada sintaksis EXPLAIN dan EXPLAIN ANALYZE. Untuk memahami perbedaan tersebut, jalankan SQL sintaksis EXPLAIN berikut ini.

```
-- Tanpa eksekusi nyata
EXPLAIN SELECT * FROM transaksi WHERE id_transaksi = 12345;
```

EXPLAIN akan menganalisis performa suatu query *_tanpa menjalankannya_*.

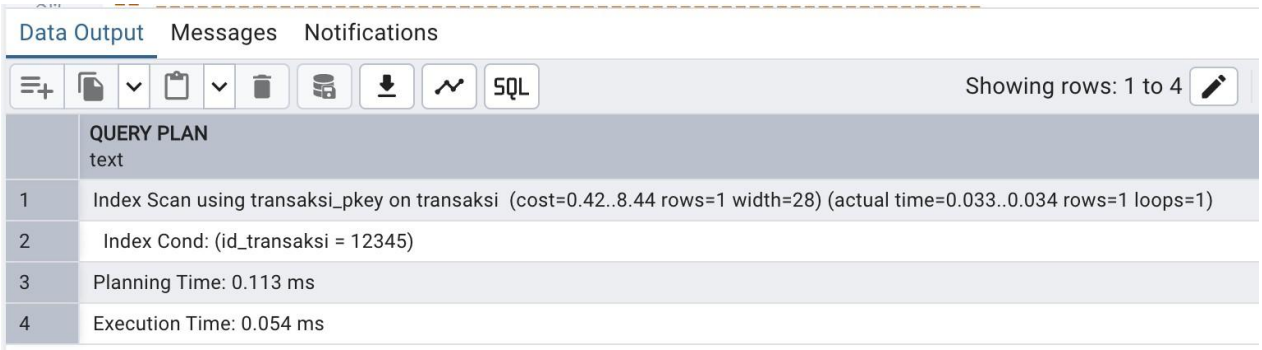

Data Output Messages Notifications

	QUERY PLAN text
1	Index Scan using transaksi_pkey on transaksi (cost=0.42..8.44 rows=1 width=28)
2	Index Cond: (id_transaksi = 12345)

Perhatikan bedanya dengan hasil dari sintaksis EXPLAIN ANALYZE berikut.

```
-- Dengan eksekusi nyata
EXPLAIN ANALYZE SELECT * FROM transaksi WHERE id_transaksi = 12345;
```

Pada output hasil eksekusi SQL tersebut, akan terdapat kata-kata Planning Time dan Execution Time. Hal ini menunjukkan, sintaksis SELECT * FROM ... benar-benar dijalankan dan kemudian dianalisis hasilnya.


Data Output Messages Notifications

Showing rows: 1 to 4

	QUERY PLAN text
1	Index Scan using transaksi_pkey on transaksi (cost=0.42..8.44 rows=1 width=28) (actual time=0.033..0.034 rows=1 loops=1)
2	Index Cond: (id_transaksi = 12345)
3	Planning Time: 0.113 ms
4	Execution Time: 0.054 ms

Topik-6: Latihan Optimasi Query & Trade-off Index

Pada bagian ini kita akan memahami konsep dimana index tidaklah selalu baik. Terdapat 'trade-off', atau harga yang harus dibayar saat kita menerapkan index. Maka dari itu, penting bagi kita untuk memahami kapan waktu atau kasus, maupun query yang tepat/optimal jika kita menerapkan index.

6. Dampak Negatif Index pada INSERT

Setiap index yang kita buat, akan memerlukan biaya (resource) tambahan. Oleh karena itu, tidak serta merta kemudian semua kolom di dalam tabel kita, harus dibuat index-nya. Mari kita coba menambahkan index ke 2 kolom yang sebenarnya kurang cocok di-index, pada tabel transaksi.

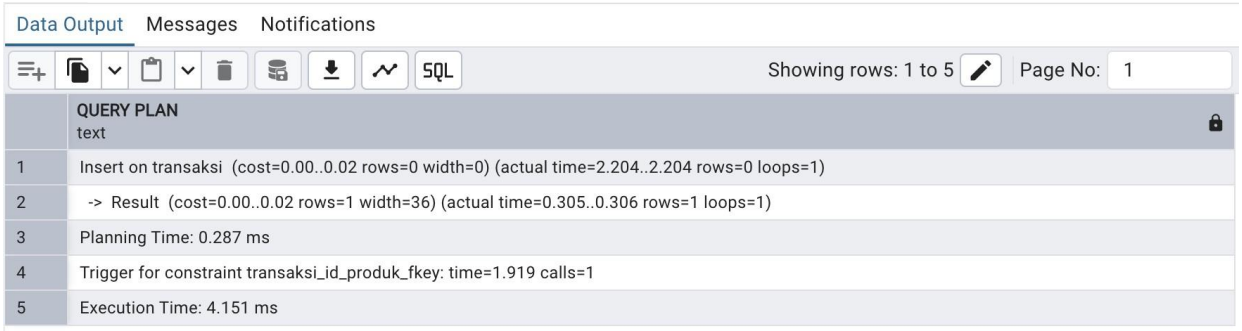
```
-- Tambahkan banyak index
CREATE INDEX idx_transaksi_total ON transaksi(total);
CREATE INDEX idx_transaksi_jumlah ON transaksi(jumlah);
```

⚠ **Catatan:** Sebagaimana layaknya daftar isi pada suatu buku matematika, akan lebih bermanfaat apabila meletakkan daftar bab (berupa teks) didalamnya, daripada meletakkan daftar angka. Demikian juga pada suatu tabel, index akan lebih optimal pada data yang berupa karakter (teks), yang sering kita gunakan di klausa WHERE, dibandingkan data yang berupa nilai (angka).

Sekarang, mari kita coba bandingkan performa sintaksis SELECT pada tabel tersebut.

```
-- Uji insert data baru
EXPLAIN ANALYZE
INSERT INTO transaksi (id_pelanggan, id_produk, tanggal_transaksi, jumlah, total)
VALUES (200, 10, CURRENT_DATE, 2, 250000);
```

Perhatikan hasilnya.



Data Output | Messages | Notifications

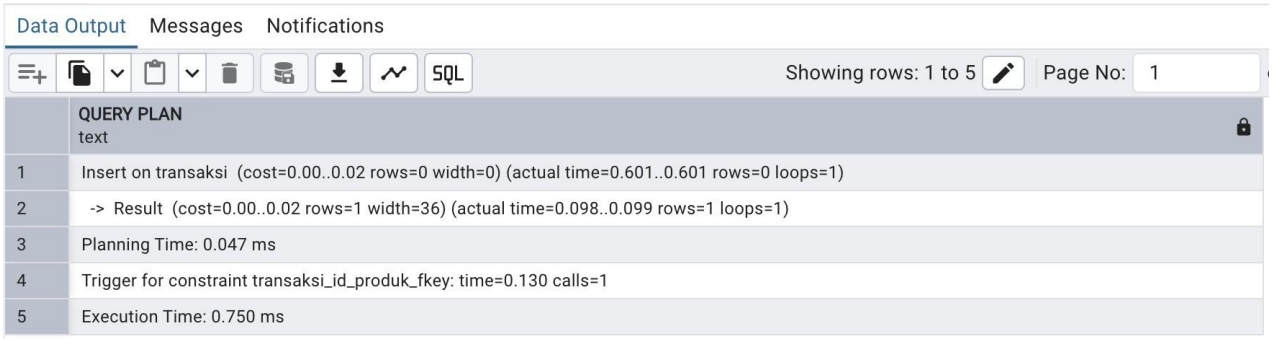
Showing rows: 1 to 5 | Page No: 1

	QUERY PLAN text
1	Insert on transaksi (cost=0.00..0.02 rows=0 width=0) (actual time=2.204..2.204 rows=0 loops=1)
2	-> Result (cost=0.00..0.02 rows=1 width=36) (actual time=0.305..0.306 rows=1 loops=1)
3	Planning Time: 0.287 ms
4	Trigger for constraint transaksi_id_produk_fkey: time=1.919 calls=1
5	Execution Time: 4.151 ms

Sekarang, hapus semua index yang kita buat sebelumnya tadi.

```
-- Drop index untuk melihat perbandingan
DROP INDEX idx_transaksi_total;
DROP INDEX idx_transaksi_jumlah;
```

Lalu jalankan sintaksis EXPLAIN ANALYZE sebelumnya. Lalu perhatikan pada output yang dihasilkan, terlihat di sana performa SELECT-nya justru lebih baik.



Data Output | Messages | Notifications

Showing rows: 1 to 5 | Page No: 1

	QUERY PLAN text
1	Insert on transaksi (cost=0.00..0.02 rows=0 width=0) (actual time=0.601..0.601 rows=0 loops=1)
2	-> Result (cost=0.00..0.02 rows=1 width=36) (actual time=0.098..0.099 rows=1 loops=1)
3	Planning Time: 0.047 ms
4	Trigger for constraint transaksi_id_produk_fkey: time=0.130 calls=1
5	Execution Time: 0.750 ms

7. Menggunakan ANALYZE

Untuk bisa bekerja dengan terpercaya, EXPLAIN maupun EXPLAIN ANALYZE mengandalkan data statistik dari tabel-tabel yang dipasangi index. Penting bagi kita untuk memperbarui statistik tersebut apabila dirasa sudah lama statistik tersebut tidak di-update.

Untuk meng-update statistik pada tabel tertentu, digunakan sintaksis seperti berikut ini.

```
-- Update statistik tabel
ANALYZE transaksi;
```

Apabila statistik sudah di-update, kita dapat menjalankan sintaksis EXPLAIN maupun EXPLAIN ANALYZE dengan hasil yang terpercaya.

```
-- Jalankan query setelah ANALYZE
EXPLAIN ANALYZE
SELECT * FROM transaksi
WHERE tanggal_transaksi BETWEEN '2024-06-01' AND '2024-06-30';
```

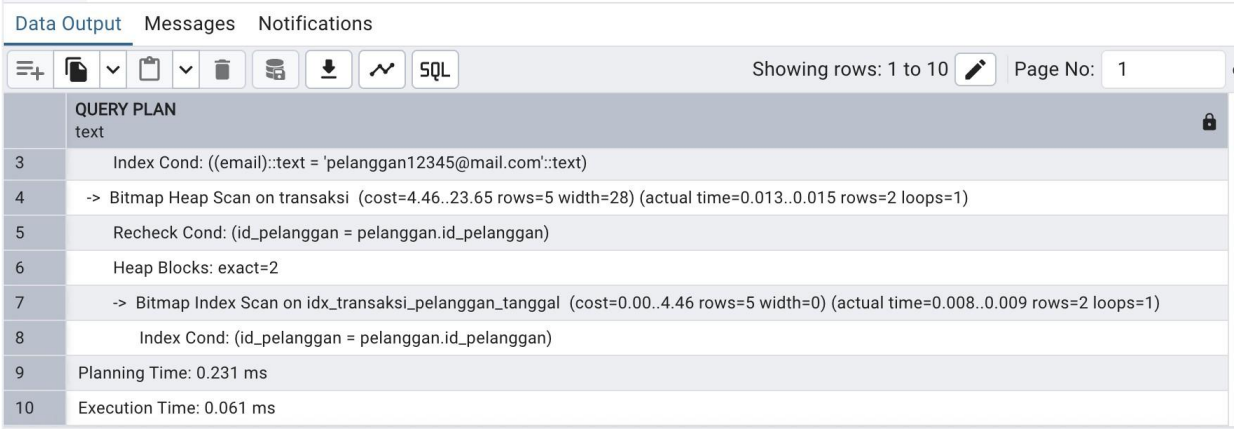
8. Query Rewrite

Pada sebagian query, index bisa termanfaatkan dengan baik dan di sebagian yang lain terkadang index tidak termanfaatkan dengan baik. Oleh karena itu, upayakan untuk memilih sintaksis SQL yang memanfaatkan index dengan baik apabila Anda ingin mengoptimasi sistem Anda.

Sebagai contoh, berikut ini adalah sintaksis query SELECT ... IN untuk menemukan data transaksi dari pelanggan yang memiliki email pelanggan12345@mail.com. Mari kita EXPLAIN ANALYZE query tersebut.

```
-- Dengan IN
EXPLAIN ANALYZE
SELECT * FROM transaksi
WHERE id_pelanggan IN (
   SELECT id_pelanggan FROM pelanggan WHERE kota = 'Jakarta'
);
```

Perhatikan waktu yang dibutuhkan untuk menjalankan query tersebut.


Data Output Messages Notifications

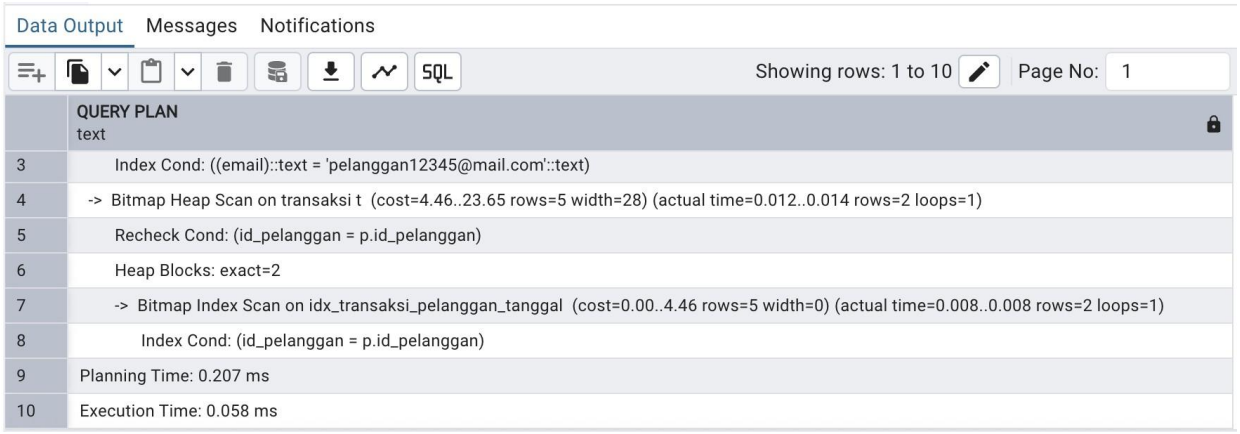
Showing rows: 1 to 10 Page No: 1

	QUERY PLAN text
3	Index Cond: ((email)::text = 'pelanggan12345@mail.com'::text)
4	-> Bitmap Heap Scan on transaksi (cost=4.46..23.65 rows=5 width=28) (actual time=0.013..0.015 rows=2 loops=1)
5	Recheck Cond: (id_pelanggan = pelanggan.id_pelanggan)
6	Heap Blocks: exact=2
7	-> Bitmap Index Scan on idx_transaksi_pelanggan_tanggal (cost=0.00..4.46 rows=5 width=0) (actual time=0.008..0.009 rows=2 loops=1)
8	Index Cond: (id_pelanggan = pelanggan.id_pelanggan)
9	Planning Time: 0.231 ms
10	Execution Time: 0.061 ms

Sekarang mari kita bandingkan dengan sintaksis SELECT ... JOIN berikut, yang sebenarnya tujuannya sama.

```
-- Dengan JOIN
EXPLAIN ANALYZE
SELECT t.*
FROM transaksi t
JOIN pelanggan p ON t.id_pelanggan = p.id_pelanggan
WHERE p.kota = 'Jakarta';
```

Amati hasilnya, dimana waktu eksekusi query tersebut sedikit lebih cepat daripada query sebelumnya.



Penutup

Setelah menyelesaikan petunjuk praktikum ini, seharusnya Anda sekarang sudah memahami bahwa:

- Index mempercepat pencarian, join, dan filter data.
- Ada berbagai tipe index: tunggal, multi-kolom, unik, dan partial.
- Index membawa trade-off: mempercepat SELECT tapi memperlambat INSERT/UPDATE/DELETE.

Optimasi query tidak hanya soal index, tapi juga soal menulis query dengan benar dan menjaga statistik tabel tetap up-to-date.

Soal Latihan Praktikum 4

Soal-1: Membuat Index Dasar

- Tabel pelanggan memiliki kolom email yang sering digunakan dalam pencarian.
- Tuliskan SQL untuk membuat sebuah index bernama idx_pelanggan_email pada kolom tersebut.
- Setelah itu, jalankan query berikut dan amati perbedaan hasil EXPLAIN ANALYZE sebelum dan sesudah index dibuat:
    - SELECT * FROM pelanggan WHERE email = 'pelanggan100@mail.com';

Soal-2: Index Multi Kolom

- Tabel transaksi memiliki kolom id_pelanggan dan tanggal_transaksi.
- Buatlah index multi-kolom pada kedua kolom tersebut untuk mempercepat query berikut:
    - SELECT * FROM transaksi WHERE id_pelanggan = 50 AND tanggal_transaksi BETWEEN '2024-01-01' AND '2024-12-31';
    - Tuliskan SQL untuk membuat index tersebut dan jelaskan mengapa index multi-kolom lebih efisien dibanding index tunggal pada id_pelanggan saja.

Soal-3: Unique Constraint

- Pada tabel pelanggan, setiap email harus unik agar tidak terjadi duplikasi data.
- Tuliskan SQL untuk menambahkan constraint UNIQUE pada kolom email.
- Lalu, coba jalankan perintah berikut:
    - INSERT INTO pelanggan (nama, email, kota) VALUES ('Pelanggan Baru', 'pelanggan100@mail.com', 'Jakarta');
- Apa hasilnya jika ternyata email tersebut sudah ada di tabel?

Soal-4: Partial Index

- Anda diminta untuk membuat index pada tabel transaksi hanya untuk data setelah tanggal 1 Januari 2025.
- Tuliskan SQL untuk membuat partial index tersebut.
- Kemudian, bandingkan hasil EXPLAIN ANALYZE untuk query berikut dengan dan tanpa partial index:
    - SELECT * FROM transaksi WHERE tanggal_transaksi > '2025-01-01';

Query Mahasiswa:

1.

```
toko_online=# CREATE INDEX idx_pelanggan_email ON pelanggan(email);
CREATE INDEX
```

2.

```
toko_online=# CREATE INDEX idx_transaksi_pelanggan_tanggal
toko_online-# ON transaksi(id_pelanggan, tanggal_transaksi);
CREATE INDEX
```

3.

```
toko_online=# ALTER TABLE pelanggan
toko_online-# ADD CONSTRAINT pelanggan_email_key UNIQUE(email);
ALTER TABLE
```

4.

```
toko_online=# CREATE INDEX idx_transaksi_after_20250101
toko_online-# ON transaksi(tanggal_transaksi)
toko_online-# WHERE tanggal_transaksi > DATE '2025-01-01';
CREATE INDEX
```

Hasil Eksekusi:

1.

```
toko_online=# SELECT * FROM pelanggan WHERE email = 'pelanggan100@mail.com';
 id_pelanggan |     nama      |         email          |  kota
--------------+---------------+------------------------+---------
          100 | Pelanggan 100 | pelanggan100@mail.com | Jakarta
(1 row)
```

Sebelum membuat index: pada baris pertama hasil EXPLAIN akan muncul Seq Scan on pelanggan. artinya PostgreSQL memindai seluruh tabel baris demi baris. Execution Time relatif lebih besar.

Setelah membuat index: pada baris pertama hasil EXPLAIN akan berubah menjadi Index Scan using idx_pelanggan_email. Execution Time seharusnya jauh lebih kecil untuk pencarian berdasarkan kolom email.

2.

```
toko_online=# EXPLAIN ANALYZE
toko_online-# SELECT * FROM transaksi
toko_online-# WHERE id_pelanggan = 50
toko_online-#   AND tanggal_transaksi BETWEEN '2024-01-01' AND '2024-12-31';
                                                                 QUERY PLAN

-------------------------------------------------------------------------------------------------------------------------------------------
-----------------
 Index Scan using idx_transaksi_pelanggan_tanggal on transaksi  (cost=0.42..8.44 rows=1 width=28) (actual time=0.146..0.14
9 rows=2 loops=1)
   Index Cond: ((id_pelanggan = 50) AND (tanggal_transaksi >= '2024-01-01'::date) AND (tanggal_transaksi <= '2024-12-31'::
date))
 Planning Time: 4.089 ms
 Execution Time: 0.182 ms
(4 rows)
```

Index tunggal pada id_pelanggan (CREATE INDEX idx_transaksi_id_pelanggan ON transaksi(id_pelanggan);) hanya mempercepat pencarian berdasarkan id_pelanggan. Namun ketika query menambahkan kondisi rentang pada tanggal_transaksi, planner mungkin harus memfilter banyak baris hasil index tunggal sehingga tidak optimal.

Index multi-kolom (id_pelanggan, tanggal_transaksi) menyimpan urutan berdasarkan id_pelanggan lalu tanggal_transaksi. Untuk query dengan id_pelanggan = <nilai> dan rentang pada tanggal_transaksi, index ini memungkinkan PostgreSQL langsung menelusuri blok index yang relevan (mengurangi jumlah baris yang harus dibaca) sehingga mengurangi I/O dan mengurangi execution time.

3.

```
toko_online=# INSERT INTO pelanggan (nama, email, kota)
toko_online-# VALUES ('Pelanggan Baru', 'pelanggan100@mail.com', 'Jakarta');
ERROR:  duplicate key value violates unique constraint "pelanggan_email_key"
DETAIL:  Key (email)=(pelanggan100@mail.com) already exists.
```

Jika email 'pelanggan100@mail.com' sudah ada, maka INSERT tersebut gagal dan PostgreSQL akan mengembalikan error duplicate key. Dengan adanya UNIQUE constraint, database menjamin tidak ada duplikasi pada kolom email.

4.

```
toko_online=# EXPLAIN ANALYZE
toko_online-# SELECT * FROM transaksi WHERE tanggal_transaksi > '2025-01-01';
                                                          QUERY PLAN
------------------------------------------------------------------------------------------------------------------------------
 Seq Scan on transaksi  (cost=0.00..3972.00 rows=140823 width=28) (actual time=0.033..16.482 rows=140050 loops=1)
   Filter: (tanggal_transaksi > '2025-01-01'::date)
   Rows Removed by Filter: 59950
 Planning Time: 2.570 ms
 Execution Time: 20.691 ms
(5 rows)
```

Untuk baris-baris transaksi dengan tanggal_transaksi > '2025-01-01', planner dapat memakai partial index sehingga menghasilkan Index Scan pada idx_transaksi_after_20250101 dan execution time turun. Untuk query yang mencari tanggal di luar rentang partial index (mis. sebelum 2025-01-01), partial index tidak digunakan. Planner akan melakukan Seq Scan atau strategi lain, sehingga execution time bisa jauh lebih besar.

Refleksi Pembelajaran

Nama Mahasiswa: Narendra Daniswara Alamsyah
NIM: 244107060095
KELAS: SIB-2G

Hal yang sudah dipahami:

- Definisi Index
- Jenis Index
- Pemeriksaan Index
- Pembuatan Index
- Penghapusan Index
- Constraint Unik

Kesulitan yang dialami: Kebutuhan sistem PostgresSQL yang terbaru untuk dapat me-restore file backupnya.