

PROGRAM STUDI TEKNIK INFORMATIKA  
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA  
INSTITUT TEKNOLOGI BANDUNG

# **Praktikum 2**

## ***Database Performance Tuning***

Jumat, 11 Oktober 2024

Oleh:

13522018 - Ibrahim Ihsan Rasyid

13522053 - Erdianti Wiga Putri Andini

**IF3140 - Sistem Basis Data**  
**2024**

# Petunjuk

1. Kerjakan setiap soal praktikum ini dengan baik
2. Untuk kebutuhan perbaikan kinerja dari query, Anda dipersilakan menggunakan kaskas *query executor* untuk melakukan simulasi pengaksesan basis data secara bersamaan untuk sejumlah 30 akses. Gunakan dengan query awal dan query akhir
3. Penggunaan query-executor.jar sebagai berikut
  - a. Pada direktori yang sama dengan jar file, buat query.sql
  - b. Masukkan query Anda ke dalam query.sql
  - c. Jalankan jar file dengan perintah java -jar query-executor.jar
4. Query pengujian pada praktikum ini adalah query dengan makna semantik yang sama dengan query awal. Silakan dibuat sendiri jika tidak diberikan
5. Nama database: **airport**
6. **Dilarang Menyontek**

# AirportDB



# 1. Performance Tuning Theory

Jawab pertanyaan berikut ini dengan baik. Seluruh *query* tidak perlu (karena data tidak tersedia) dijalankan pada basis data. Namun, setiap *query* yang dipertanyakan harus ditulis secara SINGKAT namun LENGKAP dan TEPAT. Jawaban panjang tidak akan menambah nilai.

- a. Materi IF2240 Basis Data mengajarkan bahwa dalam memodelkan suatu basis data kita harus berusaha memodelkan tabel relasional hingga ke bentuk normal tertinggi untuk menghindari adanya redundansi. Lalu, mengapa ada konsep denormalisasi? Kapan kita harus melakukan denormalisasi?

Denormalisasi digunakan untuk meningkatkan performa dengan mengurangi jumlah join. Proses ini biasanya digunakan apabila membutuhkan untuk mengakses data yang sering untuk diakses antar tabel. Teknik ini akan bermanfaat saat pemrosesan query kompleks. Teknik ini juga digunakan saat kecepatan pengambilan data diperhatikan.

- b. Apa satu rintangan utama yang akan dihadapi setelah melakukan denormalisasi? Apa solusi atas rintangan itu?

Membuat tabel menjadi lebih besar karena teknik ini lebih memilih untuk membuat banyak tabel-tabel baru sebagai hasil join daripada menggunakan sedikit tabel dengan tujuan memperkecil banyaknya penggunaan join pada query. Selain itu juga merusak susunan database karena adanya perubahan pada tabel-tabel yang ada di dalam database. Adanya redundansi dengan tujuan eksekusi memori lebih cepat. Solusinya adalah

- c. Basis data airport mendapat penambahan data yang sangat signifikan bahkan mencapai ratusan ribu data baru setiap harinya. Apabila query berikut ini sering kali dijalankan pada basis data airport setiap harinya, *metode performance tuning* apa yang paling tepat untuk digunakan? Sertakan atribut mana yang terlibat dalam implementasi *performance tuning*. Tanpa menyebut atribut, jawaban tidak dinilai. Alasan tidak perlu dituliskan.

Frequent Query	Jawaban
<pre>SELECT COUNT (flight_id) FROM airline NATURAL JOIN flight WHERE airlinename = {name}</pre>	Indexing di atribut airlinename
<pre>SELECT username, password FROM employee WHERE employee_id = {id}</pre>	-

<pre>SELECT * FROM flight_log WHERE log_date = {date}</pre>	Indexing di atribut log_date
<pre>SELECT * FROM passenger WHERE passenger_id = {id}</pre>	-

- d. Nomor paspor Marie Antoinette yang berprofesi sebagai junior DBA adalah XX13579 (data *dummy*). Buatlah *query* untuk mendapatkan total rute penerbangan yang berbeda (bukan total *flight*) yang telah dinaiki oleh Marie Antoinette. (Hati hati, terdapat banyak sekali penumpang yang bernama Marie Antoinette).

```
SELECT DISTINCT COUNT(f.departure) FROM passenger p
JOIN booking b ON p.flight_id = b.flight_id
JOIN flight f ON b.flight_id = f.flight_id
WHERE p.firstname = 'Marie'
AND p.lastname = 'Antoinette'
AND p.passportno = 'XX13579';
```

- e. Apabila terdapat 1 juta penumpang dalam tabel *passenger* dan *passengerdetails*, dan terdapat 5 juta data yang ada pada tabel *booking*. Usulkan metode (cukup sebutkan metode dan atribut terlibat saja, tanpa alasan) *performance tuning* yang sesuai jika *query* pada persoalan d sering kali dijalankan untuk penumpang lain.

--

## 2. Calling my employee

Manager Didot sangat sering mengakses data detail kontak karyawan seperti nama depan, nama belakang, alamat email, dan nomor telepon untuk menjaga komunikasi yang baik dengan para karyawannya. Data karyawan yang lain seperti alamat, gaji, dan seterusnya hanya diakses sesekali saat ingin memberikan *reward* kepada karyawan, atau karena keperluan khusus lainnya.

Pada bagian ini, buatlah query untuk mendapatkan data yang sering diperlukan Manager Didot! Gunakan perintah `explain` dan `timing` untuk mendapatkan variabel analisis!

Query Masukan	EXPLAIN SELECT firstname, lastname, emailaddress, telephoneno FROM employee;
SS Hasil Query	
<pre>airport=# SELECT firstname, lastname, emailaddress, telephoneno FROM employee; Time: 4.125 ms  airport=# EXPLAIN SELECT firstname, lastname, emailaddress, telephoneno FROM employee;                QUERY PLAN ----- Seq Scan on employee (cost=0.00..36.00 rows=1000 width=55) (1 row)  Time: 1.245 ms</pre>	

Berdasarkan informasi yang diberikan dan hasil query Manager Didot, bantu Manager Didot melakukan performance tuning terhadap tabel `employee` **TANPA MERUBAH TABEL AWAL** (Membuat tabel baru diperbolehkan). Gunakan `EXPLAIN` untuk melakukan analisis perbandingan cost sebelum dan sesudah melakukan performance tuning.

Penjelasan Permasalahan
Terdapat kolom-kolom pada tabel <code>employee</code> yang jarang diakses, sehingga saat ada pemanggilan query yang melibatkan tabel ini akan memperlambat time execution karena harus tetap membaca semua kolom. Padahal kolom yang sering diakses hanya beberapa saja.
Saran Perbaikan dan Justifikasinya
Schema tuning yang digunakan adalah <b><i>vertical splitting</i></b> . Ketika tabel memiliki banyak kolom tetapi tidak semua selalu dibutuhkan, dapat memecah tabel menjadi tabel yang lebih kecil secara vertikal untuk meminimalkan jumlah data yang dibaca dari disk. Dengan membagi tabel, hanya perlu mengakses data yang dibutuhkan dalam query tertentu. Dalam kasus ini, kami memecah tabel <code>employee</code> dan membuat tabel baru bernama <code>data_employe</code> yang hanya berisi kolom-kolom yang sering diakses.

Query untuk Melakukan Perbaikan	CREATE TABLE data_employee AS (SELECT firstname, lastname, emailaddress, telephoneno FROM employee);
SS Query Perbaikan	
<pre>airport=# CREATE TABLE data_employee AS       (SELECT firstname, lastname, emailaddress, telephoneno FROM employee); SELECT 1000 Time: 11.194 ms</pre>	
SS Query Masukan setelah Perbaikan	
<pre>airport=# SELECT * FROM data_employee; Time: 3.686 ms</pre>	
<pre>airport=# EXPLAIN SELECT * FROM data_employee;               QUERY PLAN -----  Seq Scan on data_employee  (cost=0.00..21.00 rows=1000 width=55) (1 row)  Time: 2.239 ms</pre>	

### 3. Overbooked

Seiring dengan pertumbuhan sistem, tabel `booking` mengalami penurunan kinerja akibat meningkatnya jumlah catatan. Pak Jay ingin melakukan peningkatan kinerja database airport. Diketahui *query* yang paling sering digunakan pada tabel tersebut adalah pengecekan berdasarkan `flight_id`. Diketahui `flight_id` kelipatan 3 dan 5 sangat jarang diakses.

Pada bagian ini, buatlah *query* untuk mendapatkan data yang sering diakses! Perlihatkan waktu *query*nya.

Query	SELECT * FROM booking WHERE flight_id % 3 != 0 AND flight_id % 5 != 0;
SS Hasil Query	
<pre>airport=# SELECT * FROM booking WHERE flight_id%3 != 0 AND flight_id%5 != 0; Time: 10.731 ms  airport=# EXPLAIN SELECT * FROM booking WHERE flight_id%3 != 0 AND flight_id%5 != 0           QUERY PLAN ----- Seq Scan on booking  (cost=0.00..456.72 rows=15926 width=35)   Filter: (((flight_id % '3'::bigint) &lt;&gt; 0) AND ((flight_id % '5'::bigint) &lt;&gt; 0)) (2 rows)  Time: 1.881 ms</pre>	

Berdasarkan informasi yang diberikan di soal dan hasil *query* Pak Jay, bantu Pak Jay melakukan *performance tuning* terhadap tabel `booking` **TANPA MERUBAH TABEL AWAL** (Membuat tabel baru diperbolehkan).

Gunakan `\timing` untuk analisis perbandingan waktu setelah melakukan *performance tuning*. Jika data terlalu panjang, gunakan `\q` untuk melihat hasil dari *timing*.

Penjelasan Permasalahan dan Jenis <i>Schema Tuning</i> yang dilakukan
Terdapat tabel yang sering menggunakan data tertentu pada atribut <code>flight_id</code> sehingga lebih baik untuk dibuat suatu tabel baru agar mengurangi <i>cost</i> dan <i>time execution</i> saat mengakses tabel tersebut. Jenis <i>schema tuning</i> yang diaplikasikan adalah <b>horizontal splitting</b> dimana teknik untuk mengoptimasinya adalah dengan membagi sebuah tabel berdasarkan row. Row disini didefinisikan sebagai ketentuan untuk membagi tabel menjadi 2 bagian (bagian yang sering diakses dan bagian yang jarang diakses). Memisahkan data berdasarkan rentang tertentu dan dapat dipisahkan ke table terpisah sehingga <i>query</i> pada table baru lebih cepat karena datanya lebih kecil.
Saran Perbaikan dan Justifikasinya



Saran perbaikannya adalah **horizontal splitting** dimana teknik untuk mengoptimasinya adalah dengan membagi sebuah tabel berdasarkan row. Row disini didefinisikan sebagai ketentuan untuk membagi tabel menjadi 2 bagian (bagian yang sering diakses dan bagian yang jarang diakses). Memisahkan data berdasarkan rentang tertentu dan dapat dipisahkan ke table terpisah sehingga query pada table baru lebih cepat karena datanya lebih kecil.

Query untuk Melakukan Perbaikan	CREATE TABLE access_flight AS (SELECT * FROM booking WHERE flight_id % 3 != 0 AND flight_id % 5 != 0);
---------------------------------	--

### SS Query Perbaikan

```
airport=# CREATE TABLE access_flight AS (SELECT * FROM booking WHERE flight_id % 3 != 0 AND flight_id % 5 != 0);
SELECT 7838
Time: 14.358 ms
```

Query Testing Setelah Perbaikan	SELECT * FROM access_flight;
---------------------------------	------------------------------

### SS Query Testing setelah Perbaikan

```
airport=# SELECT * FROM access_flight;
Time: 9.800 ms

airport=# EXPLAIN SELECT * FROM access_flight;
               QUERY PLAN
-----
Seq Scan on access_flight  (cost=0.00..144.38 rows=7838 width=35)
(1 row)

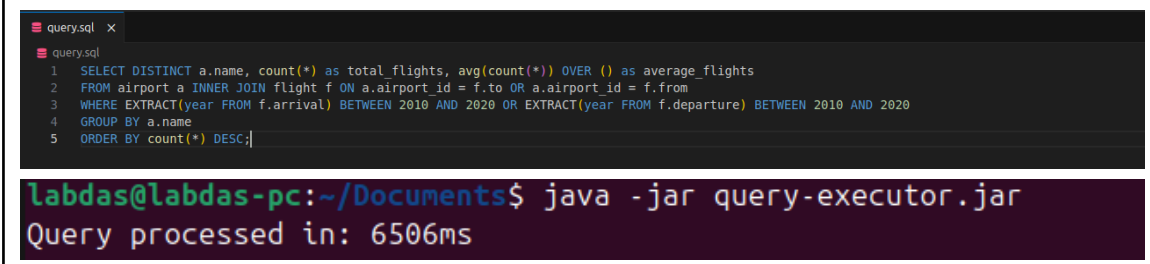
Time: 1.222 ms
```

## 4. Terlalu Banyak untuk Hal Stagnan

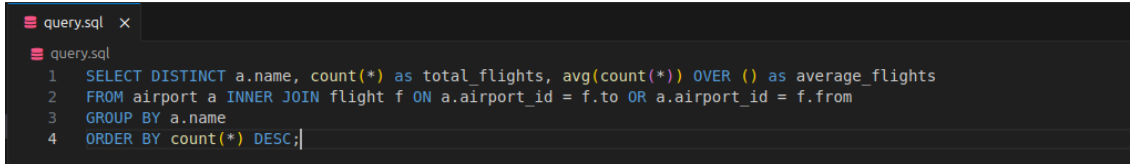
Sebuah *service provider* menggunakan akses basis data AirportDB untuk menampilkan nama bandara dan penerbangan totalnya pada tahun 2010 s.d. 2020. Query yang digunakan seperti di bawah. Tugas Anda:

1. Analisis permasalahan jika query tersebut dijalankan terus menerus (misalkan via pemanggilan API yang dapat disimulasikan *load*-nya dengan penggunaan *query-executor.jar*)! Tampilkan waktu untuk sebagai salah satu variabel analisis!
2. Berikan solusi untuk mengatasi permasalahan yang Anda temukan!

Asumsikan sekarang adalah tahun 2024 dan query tidak dapat diubah

Query	<pre>SELECT DISTINCT a.name, count(*) as total_flights, avg(count(*)) OVER () as average_flights FROM airport a INNER JOIN flight f ON a.airport_id = f.to OR a.airport_id = f.from WHERE EXTRACT(year FROM f.arrival) BETWEEN 2010 AND 2020 OR EXTRACT(year FROM f.departure) BETWEEN 2010 AND 2020 GROUP BY a.name ORDER BY count(*) DESC;</pre>
SS Hasil Query dan Hasil query-executor.jar	
	

Penjelasan Permasalahan
<p>Ditemukan bahwa tahun yang tertera pada arrival dan departure di skema basis data ini hanyalah 2015 (tidak ada tahun lain). Adanya eksekusi query EXTRACT YEAR di setiap kali pengecekan row menjadi memakan waktu lebih lama karena query tersebut dieksekusi 2 kali untuk tiap row.</p> <pre>airport=# SELECT DISTINCT EXTRACT(year FROM flight.departure) FROM flight WHERE EXTRACT(year FROM flight.departure) BETWEEN 2010 AND 2020; extract ----- 2015 (1 row)  Time: 50.599 ms</pre>
Saran Perbaikan dan Justifikasinya
<p>Tidak perlu dilakukan EXTRACT YEAR untuk mengecek apakah tahun arrival or departure di antara 2010 dan 2020 karena hanya ada satu tahun di basis data ini.</p>

Query untuk Melakukan Perbaikan	CREATE INDEX IF NOT EXISTS idx_flight_departure ON flight (departure);
SS Query Perbaikan	
 <pre> query.sql x query.sql 1 SELECT DISTINCT a.name, count(*) as total_flights, avg(count(*)) OVER () as average_flights 2 FROM airport a INNER JOIN flight f ON a.airport_id = f.to OR a.airport_id = f.from 3 GROUP BY a.name 4 ORDER BY count(*) DESC; </pre>	
Query Pengujian	SELECT DISTINCT a.name, count(*) as total_flights, avg(count(*)) OVER () as average_flights FROM airport a INNER JOIN flight f ON a.airport_id = f.to OR a.airport_id = f.from GROUP BY a.name ORDER BY count(*) DESC;
SS Query Masukan dan Hasil query-executor.jar setelah Perbaikan	
Kesimpulan Akhir dan Alasan	

## 5. Oopsie!

Pak Julala sedang melakukan eksplorasi terhadap sistem *database* bandara tempat ia bekerja. Namun, Pak Julala secara asal-asalan memasukkan *query* yang ia dapatkan setelah berbincang panjang dengan CetJipiti.

```
DO $$
DECLARE
    constraint_record RECORD;
BEGIN
    FOR constraint_record IN
        SELECT conname
        FROM pg_constraint
        WHERE conrelid = 'airport.weatherdata'::regclass
    LOOP
        EXECUTE format('ALTER TABLE airport.weatherdata DROP
CONSTRAINT %I', constraint_record.conname);
    END LOOP;
END $$;
```

**Notes:** Praktikan perlu mengeksekusi query ini terlebih dahulu sebelum melanjutkan soal!

Di saat yang bersamaan, Pak Julala diminta untuk mengeksekusi *query* dibawah ini oleh atasannya untuk mengakses data yang dibutuhkan.

```
(SELECT
    log_date,
    station,
    AVG(temp) AS average_temperature,
    AVG(humidity) AS average_humidity,
    AVG(airpressure) AS average_airpressure
FROM
    weatherdata
WHERE
    log_date BETWEEN '2015-12-01' AND '2015-12-31'
    AND humidity IS NOT NULL
    AND log_date IN (
        SELECT log_date FROM weatherdata ORDER BY temp DESC LIMIT 100
    )
GROUP BY
    log_date, station
ORDER BY
    log_date ASC, average_temperature DESC, average_humidity DESC,
    average_airpressure DESC
LIMIT 20)

INTERSECT

(SELECT
    log_date,
    station,
    AVG(temp) AS average_temperature,
    AVG(humidity) AS average_humidity,
    AVG(airpressure) AS average_airpressure
FROM
    weatherdata
```

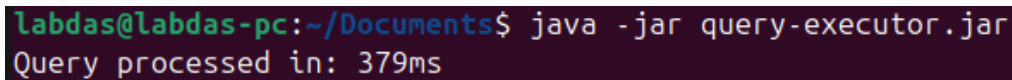
```

WHERE
    log_date BETWEEN '2015-12-01' AND '2015-12-31'
    AND humidity IS NOT NULL
    AND log_date IN (
        SELECT log_date FROM weatherdata ORDER BY humidity DESC LIMIT
100
    )
GROUP BY
    log_date, station
ORDER BY
    log_date ASC, average_temperature DESC, average_humidity DESC,
average_airpressure DESC
LIMIT 20);

```

Namun, menurut atasan Pak Julala, waktu yang dibutuhkan untuk pengambilan data tersebut dirasa masih belum cukup cepat. Berikan pemahamanmu tentang mengapa hal tersebut bisa terjadi! Selanjutnya, bantulah Pak Julala untuk mencari cara agar pengambilan data tersebut dilakukan secara lebih efisien! Gunakan `query-executor.jar` untuk menguji waktu eksekusi query!

**Notes:** *time* adalah salah satu fungsi bawaan dari PostgreSQL. Maka dari itu, gunakan tanda petik ('...') untuk mengakses atribut *time* pada tabel yang bersangkutan.

Query Masukan	[Query yang ditampilkan di atas]
SS Hasil Query Masukan	
	

Penjelasan Permasalahan
Secara keseluruhan, kedua query tersebut sama. Hanya memiliki perbedaan di bagian ORDER BY temp dan ORDER BY humidity. Waktu yang dibutuhkan untuk pengambilan data tersebut dirasa belum cukup cepat karena menjalankan query yang 'hampir' sama sebanyak 2 kali saat pemanggilan query tersebut.
Saran Perbaikan dan Justifikasinya
ORDER BY antara kedua query tersebut digabung saja. Log_date yang pertama dicari dari select log_date berdasarkan temp terlebih dahulu, anggap hasilnya adalah A. Lalu log_date dari A akan dicari berdasarkan humidity. Barulah hasilnya diproyeksi menjadi hasil.

Query untuk Melakukan Perbaikan	<pre> SELECT     log_date,     station,     AVG(temp) AS average_temperature,     AVG(humidity) AS average_humidity,     AVG(airpressure) AS average_airpressure FROM     weatherdata WHERE     log_date BETWEEN '2015-12-01' AND '2015-12-31'     AND humidity IS NOT NULL     AND log_date IN (         ((SELECT log_date FROM weatherdata ORDER BY temp         DESC LIMIT 100) IN         (SELECT log_date FROM weatherdata ORDER BY         humidity DESC LIMIT 100))     ) GROUP BY     log_date, station ORDER BY     log_date ASC, average_temperature DESC,     average_humidity DESC, average_airpressure DESC LIMIT 20; </pre>
SS Query Perbaikan	
SS Query Masukan setelah Perbaikan	

## 6. Query yang lambat

Pak Cello akhir-akhir ini sedang merasa frustrasi dikarenakan *query* yang dilakukannya akhir-akhir ini berjalan dengan sangat lambat. Anda sebagai *junior DBA* diperintahkan untuk memberikan solusi terhadap permasalahan berikut : *query* yang dilakukan Pak Cello biasanya melibatkan *join* 3 tabel yaitu tabel *passenger*, *passenger\_details*, dan *booking* karena Pak Cello seringkali membutuhkan informasi nomor telepon, email, dan total pengeluaran *passenger* tersebut. *Query* yang digunakan Pak Cello dapat dilihat di bawah. Bantulah Pak Cello untuk menyelesaikan permasalahan ini. Pastikan perbaikan yang dilakukan pada database juga mempertahankan *constraint* yang ada dan *integrity* dari database. Gunakan *query-executor.jar* untuk menguji waktu eksekusi *query*!

**NOTE : Pastikan konsistensi data terjaga setelah mengimplementasikan solusi yang diberikan (Contoh dari konsistensi data : Apabila terdapat 2 tabel dengan atribut yang sama, maka apabila salah satu atribut yang sama tersebut di-update pada salah satu tabel, tabel lain harus menjaga konsistensi itu).**

Query	<pre>SELECT   CONCAT(p.firstname, ' ', p.lastname),   p.passportno,   pd.emailaddress,   pd.telephoneno,   SUM(b.price) AS total_spending FROM   passenger p NATURAL JOIN   passengerdetails pd NATURAL JOIN   booking b GROUP BY   p.passenger_id,   pd.emailaddress,   pd.telephoneno;</pre>
SS Hasil Query	

```

airport=# SELECT
  CONCAT(p.firstname, ' ', p.lastname),
  p.passportno,
  pd.emailaddress,
  pd.telephoneno,
  SUM(b.price) AS total_spending
FROM
  passenger p
NATURAL JOIN
  passengerdetails pd
NATURAL JOIN
  booking b
GROUP BY
  p.passenger_id,
  pd.emailaddress,
  pd.telephoneno;
Time: 51.346 ms

```

### Penjelasan Permasalahan

Permasalahannya terletak pada ketidakefektifan query saat dijalankan karena harus melibatkan 3 join, apalagi antar ketiga table ini saling terdapat foreign key. Apabila terdapat perubahan data dalam salah satu table, akan memakan waktu lama untuk mengubah data dalam table lainnya.

### Saran Perbaikan dan Justifikasinya

Sebaiknya dilakukan denormalisasi pada tabel passenger untuk meminimalkan join pada query yang sering dilakukan sebagaimana yang dilakukan oleh Pak Cello

Query  
untuk  
Melakukan  
Perbaikan

### SS Query Perbaikan

Query  
Pengujian  
(eksekusi  
satu per  
satu)

```

SELECT * FROM passenger LIMIT 100;

UPDATE passengerdetails SET emailaddress =
'TonyJones@gmail.com' WHERE passenger_id = 14865;

SELECT * FROM passenger ORDER BY total_spending ASC LIMIT 5;

SELECT * FROM booking WHERE passenger_id = 14865;

UPDATE booking SET price = 20 WHERE booking_id = 92589;

```

### SS Query Pengujian setelah Perbaikan



**Pembagian tugas**

<b>NIM</b>	<b>No yang Dikerjakan</b>
13522018	2, 3, 4
13522053	1, 2, 3, 5