

Nomor Kelompok : 13

Kelas : K1

NIM : 18222043

Nama : Ricky Wijaya

NIM : 18222037

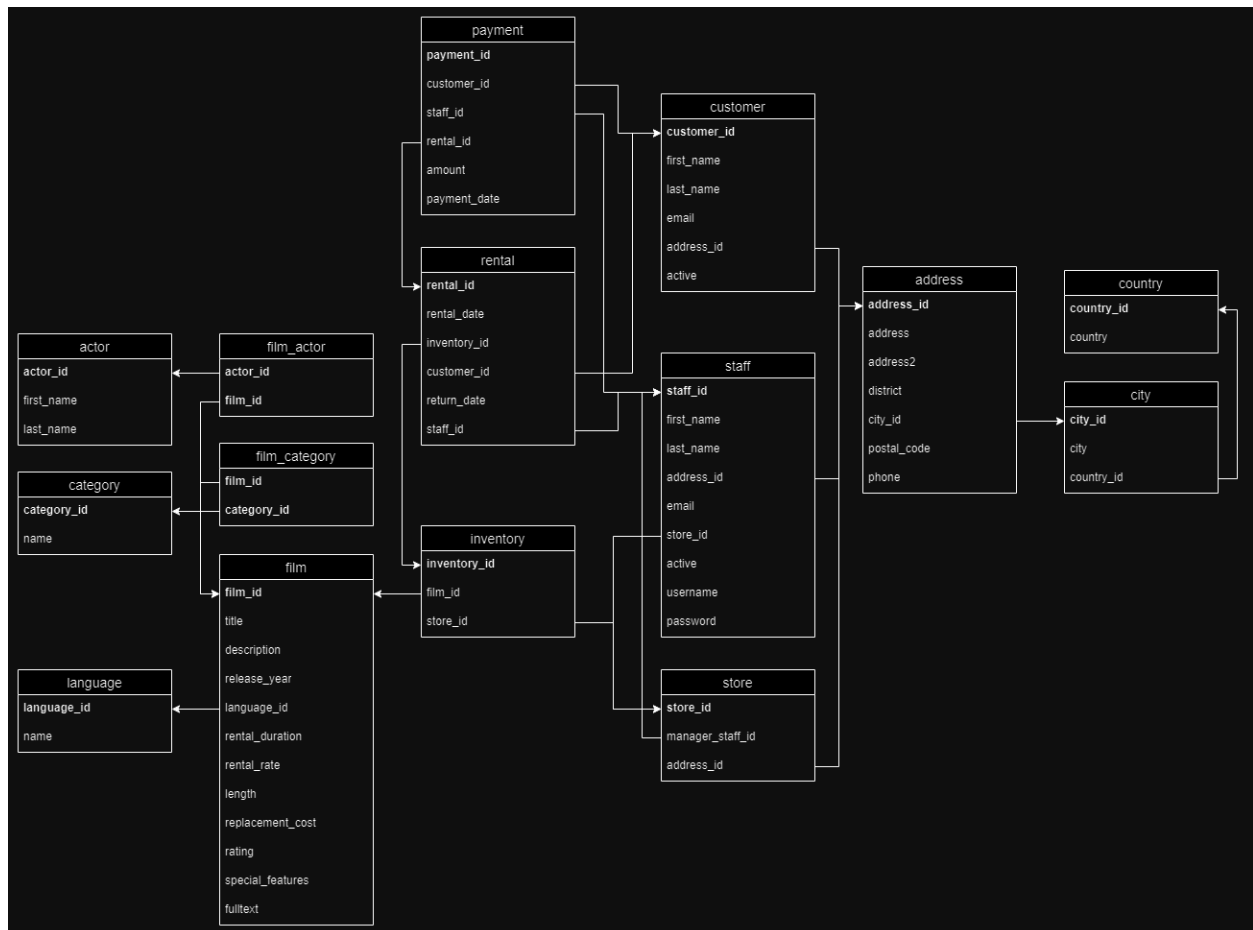
Nama : Alfandito Rais

## Lembar Kerja Praktikum 2 II2250 Manajemen Basis Data STI

Materi: *Schema Tuning & Index Tuning*

### I. Skema Basis Data

Diberikan skema basis data sebagai berikut yang tersimpan dalam database pagila.



## II. Soal

(**Note:** Pastikan telah terdapat database bernama pagila di dalam komputer yang digunakan. Jika belum, buatlah sebuah database bernama pagila dan import pagila.sql ke dalam database tersebut!)

### 1. Indexing

Query berikut memakan waktu yang cukup lama saat dieksekusi

```
SELECT c.name AS genre,
       COUNT(r.rental_id) AS total_rental
FROM rental r
LEFT JOIN inventory i ON r.inventory_id = i.inventory_id
LEFT JOIN film f ON f.film_id = i.film_id
LEFT JOIN film_category fc ON fc.film_id = f.film_id
LEFT JOIN category c ON c.category_id = fc.category_id
LEFT JOIN store s ON i.store_id = s.store_id
LEFT JOIN address a ON s.address_id = a.address_id
LEFT JOIN city ct ON a.city_id = ct.city_id
WHERE city = 'Woodridge'
      AND EXTRACT(YEAR FROM r.rental_date) < 2021
GROUP BY c.category_id
```

- Terapkan index composite pada basis data untuk mengoptimasi waktu query tersebut. Sertakan penjelasan kolom apa saja yang di-index.
- Analisa dan jelaskan hasil perbandingan waktu eksekusi query dengan dan tanpa index

Tampilkan screenshot perbandingan eksekusi query

Gunakan command **EXPLAIN ANALYZE** untuk menganalisis waktu eksekusi query

**Jawaban:**

<b>Query Pembuatan Index</b>	<pre>CREATE INDEX city_compost ON city(city_id, city);  CREATE INDEX date_2 ON rental(rental_id, rental_date);</pre>
<b>Penjelasan Index</b>	Kami menggunakan B+ Tree index pada atribut (city_id,city) karena pada WHERE CLAUSE mencari CITY dengan nama 'Woodridge' sehingga

	<p>dengan index tersebut mempercepat query karena tidak perlu melakukan FULL TABLE SCAN pada atribut city</p> <p>Kami juga menggunakan B+Tree index pada atribut (rental_id,rental_date) karena pada WHERE CLAUSE menggunakan fungsi YEAR(rental_date) sehingga pada pemrosesan rental_date tidak perlu melakukan full table scan dan hanya perlu mengakses index table</p>
--	---

SS Sebelum Index

```
QUERY PLAN
-----
GroupAggregate (cost=133.94..134.10 rows=9 width=80) (actual time=49.146..50.270 rows=16 loops=1)
  Group Key: c.category_id
  --> Sort (cost=133.94..133.96 rows=9 width=76) (actual time=49.062..49.487 rows=8121 loops=1)
    Sort Key: c.category_id
    Sort Method: quicksort  Memory: 654kB
    --> Nested Loop (cost=36.33..133.80 rows=9 width=76) (actual time=0.454..46.044 rows=8121 loops=1)
      --> Nested Loop Left Join (cost=36.04..129.57 rows=8 width=76) (actual time=0.401..26.990 rows=2311 loops=1)
        --> Nested Loop Left Join (cost=35.91..128.25 rows=8 width=6) (actual time=0.393..20.580 rows=2311 loops=1)
          --> Nested Loop Left Join (cost=35.63..125.51 rows=8 width=0) (actual time=0.384..12.144 rows=2311 loops=1)
            --> Nested Loop (cost=35.36..122.77 rows=8 width=6) (actual time=0.371..2.022 rows=2311 loops=1)
              --> Nested Loop (cost=1.33..17.10 rows=1 width=4) (actual time=0.082..0.185 rows=1 loops=1)
                --> Hash Join (cost=1.04..16.36 rows=2 width=6) (actual time=0.042..0.147 rows=2 loops=1)
                  Hash Cond: (a.address_id = s.address_id)
                  --> Seq Scan on address a (cost=0.00..13.03 rows=603 width=6) (actual time=0.014..0.063 rows=603 loops=1)
                  --> Hash (cost=1.02..1.02 rows=2 width=6) (actual time=0.014..0.015 rows=2 loops=1)
                    Buckets: 1024  Batches: 1  Memory Usage: 9kB
                    --> Seq Scan on store s (cost=0.00..1.02 rows=2 width=6) (actual time=0.006..0.008 rows=2 loops=1)
                --> Memoize (cost=0.29..0.36 rows=1 width=4) (actual time=0.016..0.016 rows=0 loops=2)
                  Cache Key: a.city_id
                  Cache Mode: logical
                  Hits: 0  Misses: 2  Evictions: 0  Overflows: 0  Memory Usage: 1kB
                  --> Index Scan using city_pkey on city ct (cost=0.28..0.35 rows=1 width=4) (actual time=0.012..0.012 rows=0 loops=1)
                    Index Cond: (city_id = a.city_id)
                    Filter: ((city)::text = 'Woodridge'::text)
                    Rows Removed by Filter: 0
              --> Bitmap Heap Scan on inventory i (cost=34.03..82.77 rows=2290 width=8) (actual time=0.285..1.205 rows=2311 loops=1)
                Recheck Cond: (store_id = s.store_id)
                Heap Blocks: exact=21
                --> Bitmap Index Scan on idx_store_id_film_id (cost=0.00..33.46 rows=2290 width=0) (actual time=0.254..0.254 rows=2311 loops=1)
                  Index Cond: (store_id = s.store_id)
              --> Index Only Scan using film_pkey on film f (cost=0.28..0.34 rows=1 width=4) (actual time=0.003..0.003 rows=1 loops=2311)
                Index Cond: (film_id = i.film_id)
                Heap Fetches: 2311
            --> Index Only Scan using film_category_pkey on film_category fc (cost=0.28..0.33 rows=1 width=4) (actual time=0.003..0.003 rows=1 loops=2)
              Index Cond: (film_id = f.film_id)
              Heap Fetches: 2311
          --> Index Scan using category_pkey on category c (cost=0.14..0.16 rows=1 width=72) (actual time=0.002..0.002 rows=1 loops=2311)
            Index Cond: (category_id = fc.category_id)
        --> Index Scan using idx_fk_inventory_id on rental r (cost=0.29..0.52 rows=1 width=8) (actual time=0.003..0.007 rows=4 loops=2311)
          Index Cond: (inventory_id = i.inventory_id)
          Filter: (EXTRACT(year FROM rental_date) = '2021'::numeric)
```

Planning Time: 4.076 ms  
Execution Time: 50.412 ms  
(43 rows)

Time: 55.589 ms

SS Sesudah Index

<pre> QUERY PLAN GroupAggregate (cost=133.94..134.10 rows=9 width=80) (actual time=13.007..13.726 rows=16 loops=1)   Group Key: c.category_id   -&gt; Sort (cost=133.94..133.96 rows=9 width=76) (actual time=12.953..13.226 rows=8121 loops=1)     Sort Key: c.category_id     Sort Method: quicksort  Memory: 654kB     -&gt; Nested Loop (cost=36.33..133.88 rows=9 width=76) (actual time=0.095..12.094 rows=8121 loops=1)       -&gt; Nested Loop Left Join (cost=36.04..129.57 rows=8 width=76) (actual time=0.084..6.744 rows=2311 loops=1)         -&gt; Nested Loop Left Join (cost=35.91..128.25 rows=8 width=6) (actual time=0.002..5.068 rows=2311 loops=1)           -&gt; Nested Loop Left Join (cost=35.63..125.51 rows=8 width=8) (actual time=0.000..2.706 rows=2311 loops=1)             -&gt; Nested Loop (cost=35.36..122.77 rows=8 width=6) (actual time=0.078..0.525 rows=2311 loops=1)               -&gt; Nested Loop (cost=1.33..17.10 rows=1 width=4) (actual time=0.020..0.003 rows=1 loops=1)                 -&gt; Hash Join (cost=1.04..16.36 rows=2 width=6) (actual time=0.010..0.074 rows=2 loops=1)                   Hash Cond: (a.address_id = s.address_id)                   -&gt; Seq Scan on address a (cost=0.00..13.03 rows=603 width=6) (actual time=0.004..0.034 rows=603 loops=1)                   -&gt; Hash (cost=1.02..1.02 rows=2 width=6) (actual time=0.003..0.004 rows=2 loops=1)                     Buckets: 1024  Batches: 1  Memory Usage: 9kB                     -&gt; Seq Scan on store s (cost=0.00..1.02 rows=2 width=6) (actual time=0.002..0.002 rows=2 loops=1)                 -&gt; Memoize (cost=0.29..0.36 rows=1 width=4) (actual time=0.004..0.004 rows=0 loops=2)                   Cache Key: a.city_id                   Cache Mode: logical                   Hits: 0  Misses: 2  Evictions: 0  Overflows: 0  Memory Usage: 1kB                   -&gt; Index Scan using city_pkey on city ct (cost=0.28..0.35 rows=1 width=4) (actual time=0.003..0.003 rows=0 loops=1)                     Index Cond: (city_id = a.city_id)                     Filter: ((city)::text = 'Woodridge')::text                     Rows Removed by Filter: 0               -&gt; Bitmap Heap Scan on inventory i (cost=34.03..82.77 rows=2290 width=8) (actual time=0.057..0.283 rows=2311 loops=1)                 Recheck Cond: (store_id = s.store_id)                 Heap Blocks: exact=21                 -&gt; Bitmap Index Scan on idx_store_id_film_id (cost=0.00..33.46 rows=2290 width=0) (actual time=0.050..0.050 rows=2311 loops=1)                   Index Cond: (store_id = s.store_id)                 -&gt; Index Only Scan using film_pkey on film f (cost=0.28..0.34 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=2311)                   Index Cond: (film_id = i.film_id)                   Heap Fetches: 2311             -&gt; Index Only Scan using film_category_pkey on film_category fc (cost=0.28..0.33 rows=1 width=4) (actual time=0.001..0.001 rows=1 loops=2)               Index Cond: (film_id = f.film_id)               Heap Fetches: 2311           -&gt; Index Scan using category_pkey on category c (cost=0.14..0.16 rows=1 width=72) (actual time=0.000..0.000 rows=1 loops=2311)             Index Cond: (category_id = fc.category_id)         -&gt; Index Scan using idx_fk_inventory_id on rental r (cost=0.29..0.52 rows=1 width=8) (actual time=0.001..0.002 rows=4 loops=2311)           Index Cond: (inventory_id = i.inventory_id)           Filter: (EXTRACT(year FROM rental_date) &lt; '2021')::numeric) Planning Time: 0.797 ms Execution Time: 13.766 ms (43 rows) Time: 15.008 ms </pre>	
Perbandingan	54 ms → 15.008 ms

## 2. Observasi Query Plan

Pada bagian ini Anda akan melakukan eksekusi tiga jenis query yang memiliki hasil yang sama dan menganalisis kinerja untuk masing-masing query. Analisis kinerja DBMS terhadap ketiga query menggunakan fitur yang tersedia pada PostgreSQL. Berikan penjelasan mengenai hal-hal berikut.

- Proses eksekusi masing-masing query (Gunakan query EXPLAIN ANALYZE)
- Waktu eksekusi masing-masing query
- Analisis perbandingan ketiga query

### Query 1

```

WITH FavoriteFilm AS (
    SELECT
        f.film_id,
        COUNT(r.rental_id) AS total_rentals
    FROM
        film f
    JOIN
        inventory i ON f.film_id = i.film_id
    JOIN
        rental r ON i.inventory_id = r.inventory_id

```

```

        GROUP BY
            f.film_id
        ORDER BY
            total_rentals DESC
        LIMIT 1
    )
SELECT
    store.store_id,
    SUM(payment.amount) AS total_payment_amount
FROM
    payment
JOIN rental ON payment.rental_id = rental.rental_id
JOIN inventory ON rental.inventory_id = inventory.inventory_id
JOIN store ON inventory.store_id = store.store_id
JOIN FavoriteFilm ON inventory.film_id = FavoriteFilm.film_id
GROUP BY store.store_id;

```

## Query 2

```

SELECT
    s.store_id,
    SUM(p.amount) AS total_payment_amount
FROM
    payment p
JOIN rental r ON p.rental_id = r.rental_id
JOIN inventory i ON r.inventory_id = i.inventory_id
JOIN store s ON i.store_id = s.store_id
WHERE i.film_id = (
    SELECT i.film_id
    FROM rental r
    JOIN inventory i ON r.inventory_id = i.inventory_id
    GROUP BY i.film_id
    ORDER BY COUNT(r.rental_id) DESC
    LIMIT 1
)
GROUP BY s.store_id;

```

## Query 3

```

WITH FavoriteFilm AS (
    SELECT
        i.film_id
    FROM
        rental r
    JOIN
        inventory i ON r.inventory_id = i.inventory_id
    GROUP BY
        i.film_id
    ORDER BY
        COUNT(r.rental_id) DESC
    LIMIT 1
2), TotalPaymentPerStore AS (
    SELECT
        s.store_id,
        i.film_id,
        SUM(p.amount) AS total_payment_amount
    FROM
        payment p
    JOIN
        rental r ON p.rental_id = r.rental_id
    JOIN
        inventory i ON r.inventory_id = i.inventory_id
    JOIN
        store s ON i.store_id = s.store_id
    GROUP BY
        s.store_id, i.film_id
)
SELECT
    tpps.store_id,
    tpps.total_payment_amount
FROM
    TotalPaymentPerStore tpps
JOIN
    FavoriteFilm ff ON tpps.film_id = ff.film_id
ORDER BY
    tpps.total_payment_amount DESC;

```



	<p>dengan waktu 10.67 ms dan disusul oleh Query 1 dengan 12.58 ms dan Query 3 menjadi query paling lambat dengan waktu 25.37 ms</p> <p>Dari sisi query optimization pun dapat terlihat bahwa Query 3 paling kompleks karena menggunakan banyak join tetapi Query 2 paling tidak kompleks karena lebih sedikit menggunakan join. Selain itu, Query 3 lebih berat karena menggunakan 2 WITH CLAUSE sedangkan pada Query 1 menggunakan 1 WITH CLAUSE dan Query 2 paling efektif karena tidak menggunakan WITH CLAUSE</p>
--	---

### 3. Query Tuning

Analisis dan eksekusi query berikut pada DBMS.

```
WITH FilmActor AS (
  SELECT
    f.*
    , count(fa.actor_id) AS total_actor
  FROM
    film f, film_actor fa
  WHERE
    f.film_id = fa.film_id
  GROUP BY
    f.film_id
)
, FilmCategory AS (
  SELECT
    f.*
    , c.*
  FROM
    film f, film_category fc, category c
  WHERE
    f.film_id = fc.film_id
    AND fc.category_id = c.category_id
)
, final AS (
```



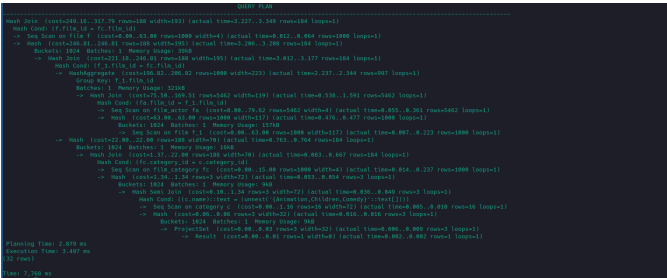
```

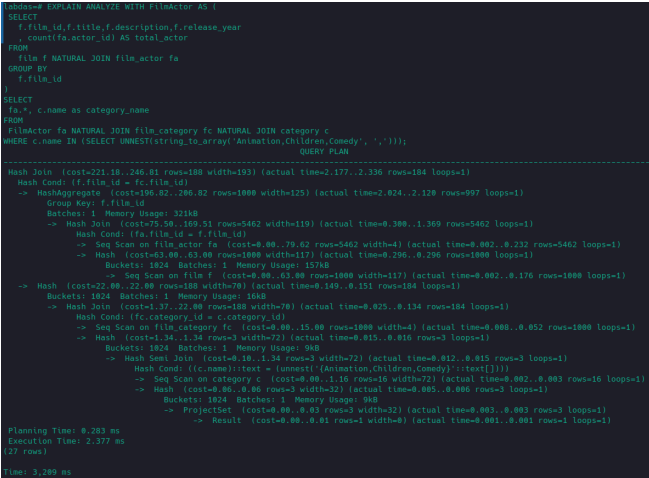
SELECT
    fa.film_id
    , fa.title
    , fa.description
    , fa.release_year
    , fa.total_actor
    , fc.name AS category_name
FROM
    FilmActor fa, FilmCategory fc
WHERE
    fa.film_id = fc.film_id
    AND fc.name IN (SELECT
UNNEST(string_to_array('Animation,Children,Comedy', ',')))
)
SELECT
    *
FROM
    FINAL

```

Melalui analisa Anda, apakah terdapat query yang memberikan output yang sama dengan query di atas tetapi memiliki waktu eksekusi yang lebih cepat? Jika ada, buktikan dengan memberikan satu query yang menurut anda paling efisien tanpa mengubah semantik dari query yang ada sebelumnya. Sertakan juga analisis mengapa query tersebut bisa lebih efisien.

Jawaban:

<p><b>Waktu Eksekusi &amp; Proses Eksekusi Query Soal (SS)</b></p>	
<p><b>Usulan Query</b></p>	<p>WITH FilmActor AS (  SELECT  f.film_id,f.title,f.description,f.release_year  , count(fa.actor_id) AS total_actor  FROM  film f NATURAL JOIN film_actor fa  GROUP BY</p>

	<pre> f.film_id ) SELECT fa.*, c.name FROM FilmActor fa NATURAL JOIN film_category fc NATURAL JOIN category c WHERE c.name IN (SELECT UNNEST(string_to_array('Animation,Children,Comedy', ','))); </pre>
<p><b>Waktu Eksekusi &amp; Proses Eksekusi Query Usulan (SS)</b></p>	 <pre> --Database Explain Analyze WITH EXPLAIN AS SELECT f.film_id, f.title, f.description, f.release_year, count(fa.actor_id) AS total_actor FROM FilmActor fa NATURAL JOIN FilmActor fa GROUP BY f.film_id SELECT fa.*, c.name AS category_name FROM FilmActor fa NATURAL JOIN film_category fc NATURAL JOIN category c WHERE c.name IN (SELECT UNNEST(string_to_array('Animation,Children,Comedy', ','))) --QUERY PLAN ----- Hash Join (cost=221.18..246.81 rows=188 width=193) (actual time=2.177..2.336 rows=184 loops=1)   Hash Cond: (f.film_id = fc.film_id)   -&gt; HashAggregate (cost=196.82..286.82 rows=1888 width=125) (actual time=2.824..2.128 rows=997 loops=1)     Group Key: f.film_id     Batches: 1 Memory Usage: 32148     -&gt; Hash Join (cost=75.58..169.51 rows=5462 width=119) (actual time=0.588..1.369 rows=5462 loops=1)       Hash Cond: (fa.film_id = f.film_id)       -&gt; Seq Scan on film_actor fa (cost=0.00..79.62 rows=5462 width=4) (actual time=0.002..0.232 rows=5462 loops=1)       -&gt; Hash (cost=63.66..63.66 rows=1888 width=117) (actual time=0.256..0.256 rows=1888 loops=1)         Buckets: 1824 Batches: 1 Memory Usage: 15736         -&gt; Seq Scan on film f (cost=0.00..63.66 rows=1888 width=117) (actual time=0.002..0.176 rows=1888 loops=1)     -&gt; Hash (cost=22.49..22.49 rows=188 width=78) (actual time=0.149..0.151 rows=184 loops=1)       Buckets: 1824 Batches: 1 Memory Usage: 1648       -&gt; Hash Join (cost=1.37..22.89 rows=188 width=78) (actual time=0.825..0.134 rows=184 loops=1)         Hash Cond: (fc.category_id = c.category_id)         -&gt; Seq Scan on film_category fc (cost=0.00..15.88 rows=1888 width=4) (actual time=0.888..0.852 rows=1888 loops=1)         -&gt; Hash (cost=1.34..1.34 rows=3 width=72) (actual time=0.815..0.816 rows=3 loops=1)           Buckets: 1824 Batches: 1 Memory Usage: 968           -&gt; Hash Semi Join (cost=0.18..1.34 rows=3 width=72) (actual time=0.812..0.815 rows=3 loops=1)             Hash Cond: ((c.name)::text = (unnest('Animation,Children,Comedy')::text))             -&gt; Seq Scan on category c (cost=0.00..1.15 rows=15 width=72) (actual time=0.802..0.803 rows=15 loops=1)             -&gt; Hash (cost=0.86..0.86 rows=3 width=32) (actual time=0.805..0.808 rows=3 loops=1)               Buckets: 1824 Batches: 1 Memory Usage: 968               -&gt; ProjectSet (cost=0.89..0.83 rows=3 width=32) (actual time=0.803..0.803 rows=3 loops=1)                 -&gt; Result (cost=0.88..0.81 rows=1 width=8) (actual time=0.801..0.801 rows=1 loops=1) Planning Time: 8.283 ms Execution Time: 2.377 ms (27 rows) Time: 3.289 ms </pre>
<p><b>Penjelasan Query Baru &amp; Analisis Perbandingan</b></p>	<p>Query baru yang kami buat menggunakan SQL Query Optimization</p> <ol style="list-style-type: none"> <li>Query Selection       <p>Pada klausa with kami hanya mengambil informasi yang dibutuhkan sehingga mengurangi waktu yang dibutuhkan untuk menghasilkan query tersebut</p> </li> <li>Mengurangi WITH CLAUSE       <p>Pada query yang kami berikan kami mengubah 2 klausa with menjadi 1 klausa dengan seperti itu akan mengurangi join dan akan menyebabkan waktu query menjadi lebih cepat</p> </li> <li>Mengubah Cartesian Product menjadi NATURAL JOIN       <p>Pada query kami, kami mengubah Cartesian Product dan WHERE Clause menjadi NATURAL JOIN, dengan seperti itu akan mengurangi I/O Cost yang terjadi karena menghindari “Full Scan Table”. Oleh karena itu, kami</p> </li> </ol>

	<p>mengubah</p> <pre>FROM film f, film_actor fa WHERE f.film_id = fa.film_id</pre> <p>Menjadi film f NATURAL JOIN film_category fa</p>
--	--

#### 4. Soal Tambahan (Khusus untuk kelompok dengan 3 orang)

Toko Pagila saat ini menjalin kerja sama dengan beberapa negara, diantaranya adalah India, Indonesia, dan Japan. Kerja sama ini akan berupaya untuk menyejahterakan staff Toko Pagila yang sering melayani rental dari customer yang berasal dari negara-negara tersebut dengan memberikan insentif tambahan dengan aturan sebagai berikut.

1. Setiap transaksi yang berasal dari negara India, maka staff akan diberikan insentif sebesar 1% dari amount transaksi.
2. Setiap transaksi yang berasal dari negara Indonesia, maka staff akan diberikan insentif sebesar 0,5% dari amount transaksi.
3. Setiap transaksi yang berasal dari negara Japan, maka staff akan diberikan insentif sebesar 2% dari amount transaksi.

Buatlah 2 query untuk menampilkan staff\_id, total\_insentif\_india, total\_insentif\_indonesia, total\_insentif\_japan, dan total\_insentif dan urutkan berdasarkan total\_insentif dari yang terbesar.

Selanjutnya, tampilkan penjelasan proses masing-masing query dengan perintah EXPLAIN ANALYZE. Menurut Anda, query manakah yang paling efisien dari dua query yang telah Anda buat? Berikan analisis Anda!

(Gunakan **CASE WHEN**)

**Jawaban**

<b>Query 1</b>	
<b>Query 2</b>	
<b>SS Hasil Perintah EXPLAIN ANALYZE Query 1</b>	
<b>SS Hasil Perintah EXPLAIN</b>	

<b>ANALYZE Query 2</b>	
<b>Query yang Paling Efisien dan Analisisnya</b>	

### III. Pembagian Tugas

<b>NIM</b>	<b>Nama</b>	<b>Tugas</b>