Pemrosesan Query Terdistribusi & Optimasi Query Terdistribusi

TUJUAN

Mahasiswa memahami Pemrosesan Query Tersdistribusi dan Optimasi Query Tersdistribusi

Mahasiswa mengetahui penerapan Optimasi Query Terdistribusi

Optimasi Query

Optimasi Query

1. Optimasi *query* adalah suatu proses untuk menganalisis *query* untuk menentukan sumber-sumber apa saja yang digunakan oleh *query* tersebut dan apakah penggunaan dari sumber tersebut dapat dikurangi tanpa mengubah output.
2. Optimisasi *query* adalah sebuah prosedur untuk meningkatkan strategi evaluasi dari suatu *query* untuk membuat evaluasi tersebut menjadi lebih efektif.
3. Optimisasi *query* mencakup beberapa teknik seperti transformasi *query* ke dalam bentuk logika yang sama, memilih jalan akses yang optimal dan mengoptimumkan penyimpanan data.

Hindari Mismatch Tipe Data

Hindari *mismatch* tipe data untuk pengindeksan kolom

Sebelum Optimasi	Setelah Optimasi
select name,age,city,state from employee where employee_id='1000';	select name,age,city,state from employee where employee_id=**1000**;
Waktu yang dibutuhkan : **2.3 sec**	Waktu yang dibutuhkan : **0.3 sec**

Menentukan Kondisi dengan Where

Menentukan kondisi pada Where bukan Having

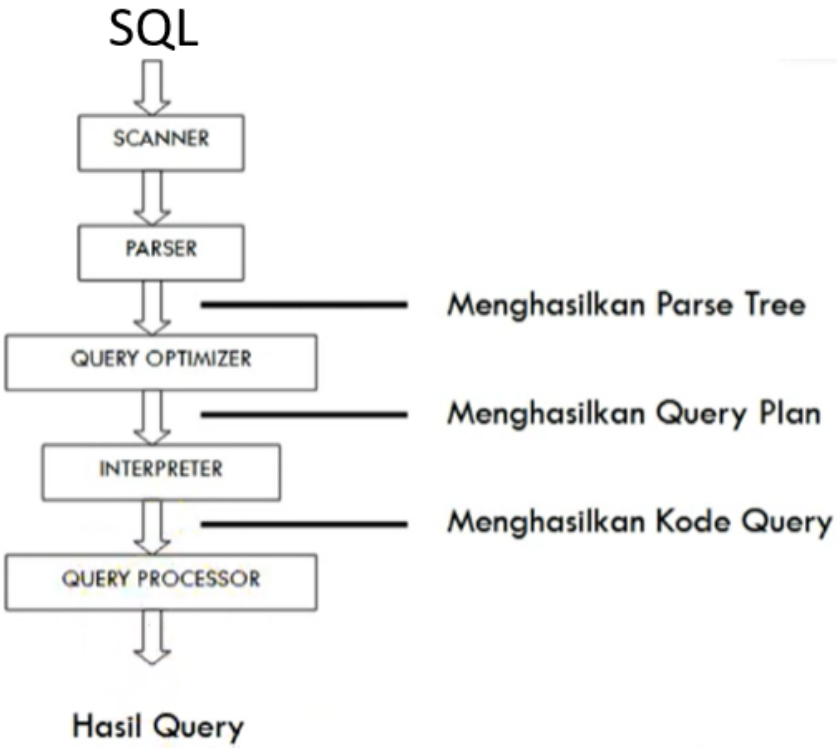
Sebelum Optimasi	Setelah Optimasi
select name, count(1) from employee group by name having name='karthi';	select name, count(1) from employee where name='karthi' group by name;
Waktu yang dibutuhkan = **2.2 sec**	Waktu yang dibutuhkan = **0.3 sec**

Cost Based Query Optimization

Tahapan-tahapan pada *Cost-Based Query Optimization*:

1. *Parsing*
2. Transformasi (*Transformation*)
3. Implementasi (*Implementation*)
4. Perencanan pemilihan skenario *query* berdasarkan estimasi biaya (*Plan selection based on cost estimates*)

Diagram Alir Query



Penjelasan Diagram Alir Query

1. *Query Parser*, melakukan verifikasi terhadap validitas dari statemen SQL, kemudian menterjemahkan *Query* kedalam struktur internal menggunakan kalkulus relasional.
2. *Query Optimizer*, mencari ekspresi terbaik dari beberapa ekspresi aljabar yang berbeda. Kriteria yang digunakan adalah "Murahnya Biaya".
3. *Code Generator/Interpreter*, menterjemahkan *Query Plan* yang dihasilkan oleh *Query Optimizer* menjadi *Query Code* untuk kemudian dikirimkan ke *Query Processor*.
4. *Query Processor*, mengeksekusi *Query Code* yang didapat dari *Code Generator*.

Phisycal Plan

1. Yang termasuk biaya dari Physical Plan adalah waktu penggunaan prosesor serta waktu komunikasi.
2. Faktor yang paling penting untuk dipertimbangkan adalah proses baca-tulis (I/O) pada *storage* penyimpanan (*harddisk*), karena hal ini adalah aksi yang paling banyak mengkonsumsi waktu.
3. Beberapa biaya yang lain yang masih berhubungan:
   a. Operasi-operasi (*join, union*, dan *intersection*)
   b. Urutan operasi
4. Penggunaan *join*, *union*, atau *intersection* harus dibatasi dan diminimalkan untuk menghasilkan *Physical Plan* terbaik

Projection

1. *Projection* menghasillkan tuple untuk setiap *argument tuple*.
2. Perubahan yang terjadi pada ukuran output adalah panjang dari *tuple-tuple* yang ada.
3. Misalkan, untuk relasi/tabel 'R'
   a. Relasi (20.000 tuple): R(a, b, c)
   b. Setiap *Tuple* (190 *bytes*): *header* = 24 *bytes*, a = 8 *bytes*, b = 8 *bytes*, c = 150 *bytes*
   c. Setiap *Block* (1024): *header* = 24 *bytes*

Projection (2)

4. Kita dapat memasukkan 5 *Tuple* kedalam 1 *Block*.
   a. 5 *Tuples* * 190 *bytes* = 950 *bytes* → dapat menggunakan 1 *Block*.
   b. Untuk 20.000 *Tuple*, kita membutuhkan 4.000 *Block* (20.000 / 5 = 4000)
5. Menggunakan *Projection* menghasilkan eliminasi dari kolom c (150 *bytes*), kita dapat mengestimasi bahwa setiap *Tuple* akan berkurang menjadi 40 *bytes* (190 – 150 = 40)

Projection (3)

6. Sekarang, estimasinya adalah 25 *Tuple* dalam 1 *Block*
7. 25 *Tuple* * 40 *bytes/tuple* = 1000 *bytes* → dapat dimasukkan kedalam 1 *block*
8. Dengan 20.000 *Tuple*, estimasi yang baru adalah 800 *Block* (20.000 *Tuple* / 25 *Tuple per Block* = 800 *Block*)

Interaksi Query dalam DBMS

1. Bagaimana *Query* berinteraksi dengan DBMS?
   a. *Interactive users*
   b. *Query* yang tertanam (*embedded query*) didalam program yang ditulis dalam C, C++, Java, dan lain-lain
2. Apa perbedaan diantara keduanya?

Interactive Users

1. *Interactive Users* adalah *user*/pengguna yang mengakses/meng-*query* database dengan mengirimkan perintah SQL langsung ke DBMS. (Untuk *Oracle* biasanya menggunakan *SQL Plus*).
2. Ketika ada *query* dari *interactive users, query* tersebut akan langsung melewati *Query Parser, Query Optimizer, Code Generator,* dan *Query Processor* setiap kali dieksekusi.

Embedded Query

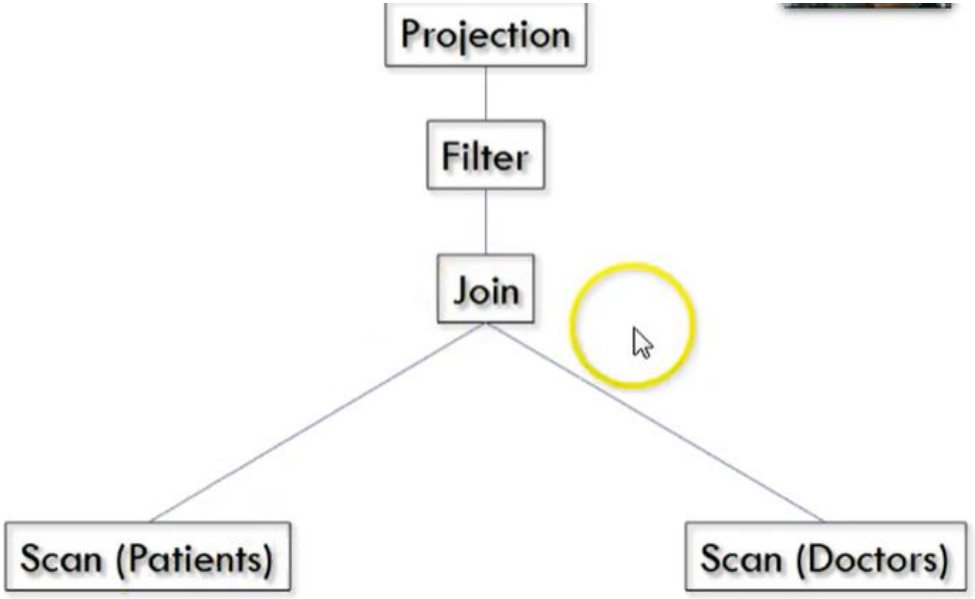
1. Ketika ada *embedded query, query* tersebut tidak harus melewati *Query Parser, Query Optimizer, Code Generator,* dan *Query Processor* setiap kali dieksekusi.
2. Pada *Embedded Query*, permintaan-permintaan (*calls*) yang dibangkitkan oleh *Code Generator* disimpan didalam basis data. Setiap kali *query* yang berada didalam program dijalankan saat *run-time, Query Processor* memanggil *calls* yang telah tersimpan didalam database.
3. Proses optimasi tidak memiliki ketergantungan dalam *embedded query.*

Cost Based Query Optimization: Algrebaic Expresion

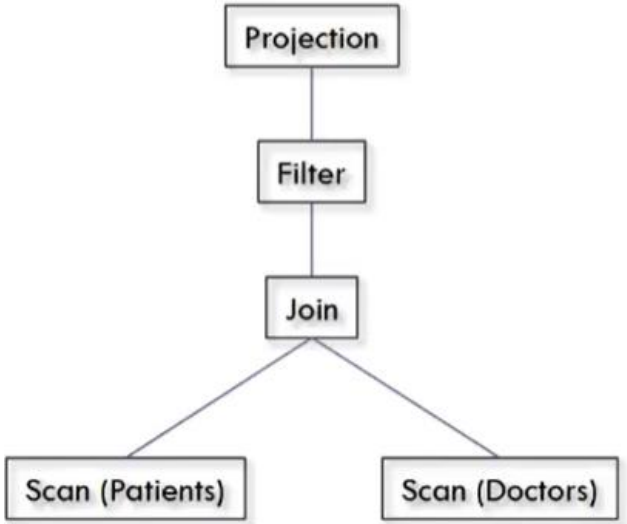
Jika kita mempunyai *query* seperti berikut:

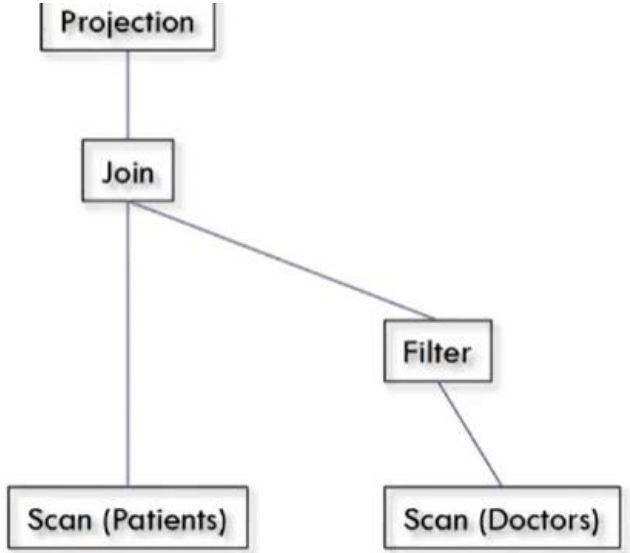
```
SELECT p.pname, d.dname
FROM Patients p, Doctors d
WHERE p.doctor = d.dname
AND d.gender = 'M'
```

Algrebaic Expresion

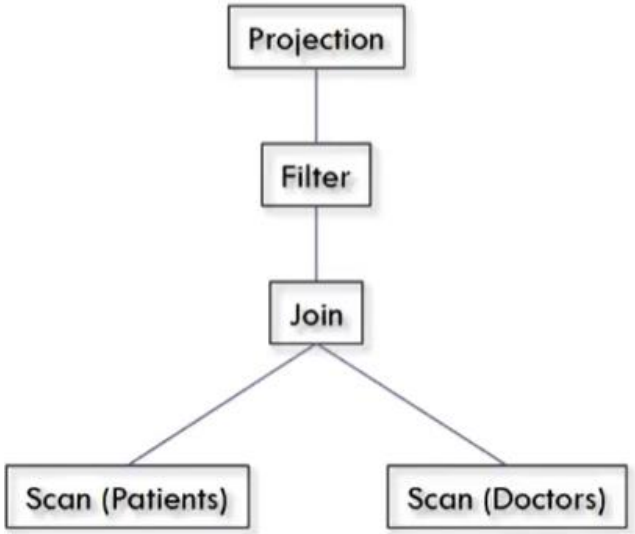


Cost Based Query Optimization: Implementation

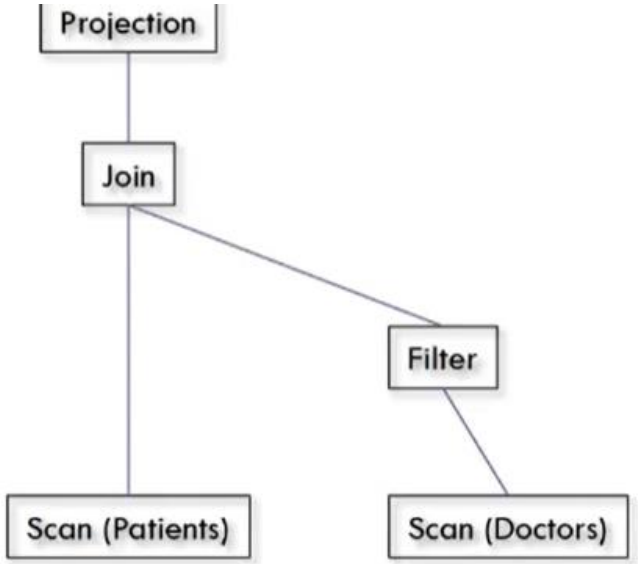




Cost Based Query Optimization: Plan Selection Based on Cost



Estimated Cost = 100ms



Estimated Cost = 50ms

Referensi

Sinaga D. Optimasi Query. Retrieved from URL: https://slideplayer.info/slide/13644720

Ahsan A.S. Optimasi Query. Retrieved from URL: https://docplayer.info/30891722-Optimasi-query-by-ahmad-syauqi-ahsan.html