Otimização de consultas

Lênisson Nasiloski Bebber

Consulta 3

```
select d.dep_id, e.nome, max(e.salario)
from departamentos d
     inner join empregados e
     on (d.dep_id = e.dep_id)
group by d.dep_id, e.nome
```

Média: **6.76**

Desvio padrão: **0.013**

Plano

1. Seq Scan on departamentos - escaneia toda a tabela sequencialmente
2. Hash - efetua o hash de departamentos para ser usado no join departamentos-empregados
3. Parallel Seq Scan on empregados - escaneia toda a tabela sequencialmente
4. Hash Cond - aplicada função hash nos valores da condição do join e.dep_id d.dep_id
5. **Hash Join** - aplicado algoritmo de join usando os valores de Hash Cond
6. Partial HashAggregate - agrega tupals para a operação GROUP BY usando tabela hash.
7. Sort - necessário ordenar os dados por conta dos Workers paralelos
8. Gather Merge - combina o output ordenados dos Workers

Hash Join

Implementação de join que utiliza uma hashtable com os valores das colunas a serem pareados.

É checado para cada coluna se o valor dos operadores do join existe na hashtable.

Consulta 6

```
select e.nome, e.dep_id
from empregados chf
     inner join empregados e
     on (e.supervisor_id = chf.emp_id)
where chf.dep_id <> e.dep_id
```

Média: **5.73**

Desvio padrão: **0.03**

Plano

1. Seq Scan on empregados chf - escaneia toda a tabela sequencialmente
2. Hash - efetua o hash de empregados para ser usado no join empregados-empregados
3. Seq Scan on empregados e - escaneia toda a tabela sequencialmente
4. **Hash Join** - aplicado algoritmo de join usando os valores de Hash Cond e respeitando o Join Filter (chf.dep_id <> e.dep_id)

Consulta 9

```
select e.dep_id, e.nome, e.salario, avg(e.salario)
over (partition by e.dep_id)
from empregados e
```

Média: **10.39**

Desvio padrão: **0.04**

Plano

1. Seq Scan on empregados e - escaneia toda a tabela sequencialmente
2. Sort - efetua o sort utilizando external merge para poder utilizar window function(agregador)
3. WindowAgg - executa a window function utilizando os valores ordenados

Otimizando as consultas

Configurações do PostgreSQL

```
shared_buffers = 128MB --> shared_buffers = 6GB
```

Resultado:

Consulta 3

```
Tempo médio: 6.76 --> 6.53
```

Consulta 6

```
Tempo médio: 5.73 --> 5.70
```

Consulta 9

```
Tempo médio: 10.39 --> 10.03
```

Configurações do PostgreSQL

```
work_mem = 4MB(default) --> work_mem = 1GB
```

Resultado:

Consulta 3

```
Tempo médio: 6.53 --> 2.63
```

Consulta 6

```
Tempo médio: 5.70 --> 5.99
```

Consulta 9

```
Tempo médio: 10.03 --> 7.65
```

Consulta 9

Criação de índice para evitar sort

```
CREATE INDEX idx_depto ON empregados (dep_id ASC);
CLUSTER empregados USING idx_depto ;
```

Resultado:

```
Tempo médio: 7.65 --> 7.49
```

1. Resultado não é acentuado pois a maior parte do tempo é gasta com scan sequencial

Consulta 9

Nova consulta:

```
WITH salariomedio
AS
(
SELECT distinct emp.dep_id, avg(emp.salario) as salariom
FROM empregados emp
group by emp.dep_id
)
select e.dep_id, e.nome, e.salario, sm.salariom
from empregados e
    inner join salariomedio sm
    on sm.dep_id = e.dep_id
```

Consulta 9

Resultado:

```
Tempo médio: 7.49 --> 5.77
```

1. Resultado melhor pois Postgres consegue utilizar a consulta do CTE paralelamente(Gather Merge)

Consulta 3

Criação de índice para evitar sort

```
CREATE INDEX idx_depto ON empregados (dep_id ASC);
CLUSTER empregados USING idx_depto;

CREATE INDEX idx_deptodp ON departamentos (dep_id ASC);
CLUSTER departamentos USING idx_deptodp;
```

Consulta 3

Resultado:

```
Tempo médio: 2.63 --> 2.65
```

Consulta 3

Nova consulta

```
WITH salariomaxdep
AS
(
select e.dep_id, max(e.salario) salario
from empregados e
group by e.dep_id
)
select e.dep_id, e.nome, sl.salario
from empregados e
    inner join salariomaxdep sl
    on (sl.dep_id = e.dep_id
    and sl.salario = e.salario)
```

Consulta 3

Resultado:

```
Tempo médio: 2.63 --> 1.73
```

1. Resultado melhor, pois elimina a maior parte do scan sequencial

Consulta 6

Criação de índice para melhorar scan e evitar função Hash

```
CREATE INDEX idx_emp ON empregados (emp_id);
CLUSTER empregados USING idx_emp;
```

Consulta 6

Resultado:

```
Tempo médio: 5.99 --> 4.45
```

Referências

www.postgresql.org
www.enterprisedb.com
www.postgresqltutorial.com
www.pgmustard.com