Plans d'exécution et EXPLAIN

Marc Cousin

DALIBO
L'Expertise PostgreSQL

Exécution d'une requête

- Analyse (parser)
- Réécriture (rewriter)
- **Planification (planner)**
- Exécution (executor)

DALIBO
L'Expertise PostgreSQL

SQL est déclaratif

```
SELECT x,y,z FROM t1 JOIN t2 ON (    )
 WHERE conditions
 GROUP BY …
```

- On dit ce qu'on veut
- Pas COMMENT on le veut
- C'est le travail de l'optimiseur (planner)

DALIBO L'Expertise PostgreSQL

Optimisation

- Examine tous les plans possibles
- Choisit le meilleur par rapport aux coûts calculés
- Prend en compte tout ce qui est connu au moment de la planification
- Complexité en fonction de la factorielle du nombre de tables
- Évolue à chaque version de PostgreSQL
- Pas de cache des plans d'exécution

Un exemple

```
SELECT empno,ename,job,dname
FROM emp JOIN dept
  ON (emp.deptno=dept.deptno)
WHERE loc='CHICAGO';
                          QUERY PLAN
-------------------------------------------------------------
 Hash Join  (cost=1.06..2.29 rows=4 width=27)
   Hash Cond: (emp.deptno = dept.deptno)
   ->  Seq Scan on emp  (cost=0.00..1.14 rows=14 width=23)
   ->  Hash  (cost=1.05..1.05 rows=1 width=14)
         ->  Seq Scan on dept  (cost=0.00..1.05 rows=1 width=14)
               Filter: ((loc)::text = 'CHICAGO'::text)
```

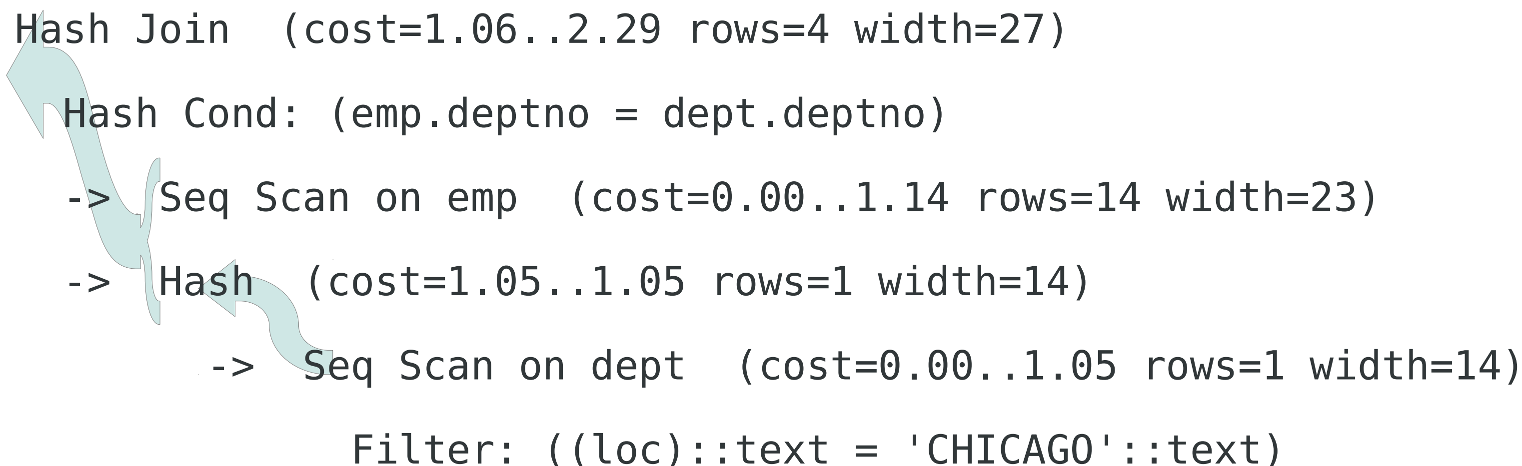

DALIBO L'Expertise PostgreSQL

Un plan

- Est composé de nœuds
- Qui produisent des données consommées par les nœuds parents
- Avec un nœud final qui retourne (éventuellement) les données à l'utilisateur
- Chaque nœud consomme les données de ses parents au fur et à mesure qu'elles sont produites

Un plan

- Un nœud consommant les données d'un autre
  - Se trouve directement à sa gauche
  - Et directement au dessus
- Les fils sont préfixés d'une flèche



```
                                 QUERY PLAN
-----------------------------------------------------------------
 Hash Join  (cost=1.06..2.29 rows=4 width=27)
   Hash Cond: (emp.deptno = dept.deptno)
   ->  Seq Scan on emp  (cost=0.00..1.14 rows=14 width=23)
   ->  Hash  (cost=1.05..1.05 rows=1 width=14)
         ->  Seq Scan on dept  (cost=0.00..1.05 rows=1 width=14)
               Filter: ((loc)::text = 'CHICAGO'::text)
```


DALIBO L'Expertise PostgreSQL

Les nœuds les plus courants

- Extraction de données
  - Seq Scan : parcours de toute la table
  - Index Scan : parcours d'index et enregistrement associé dans la table
  - Index Only Scan (9.2+) : parcours de l'index seul

DALIBO L'Expertise PostgreSQL

Les nœuds les plus courants

- Jointure
  - Nested loop (boucle imbriquée)
  - Hash Join (hachage de la table interne)
  - Merge Join (tri-fusion)
- Tri
  - Un seul nœud (Sort), bascule de mémoire à disque au besoin à l'exécution

DALIBO L'Expertise PostgreSQL

Et les autres

- Regroupement (Aggregate, HashAggregate, GroupAggregate)
- Limit
- Unique
- UNION ALL (Append), Except, Intersect
- InitPlan, Subplan…

Ce qui est important

- Certains nœuds ont des coûts de démarrage important
- Certains nœuds ont besoin de récupérer tout ce qu'a émis le nœud précédent avant de pouvoir produire leurs propres données (tris)
- Certains nœuds sont exécutés de nombreuses fois (Nested Loop)

DALIBO
L'Expertise PostgreSQL

Ce qui est important

- L'optimiseur utilise des statistiques pour deviner le comportement de la requête. Il peut donc se tromper
- L'optimiseur peut aussi mal comprendre certaines façons de rédiger les requêtes
- Le plan d'exécution permet
  - d'afficher ce que l'optimiseur a décidé
  - où l'optimiseur s'est trompé

DALIBO L'Expertise PostgreSQL

Un plan simple

```
Hash Join  (cost=1.06..2.29 rows=4 width=27)
  Hash Cond: (emp.deptno = dept.deptno)
  ->  Seq Scan on emp  (cost=0.00..1.14 rows=14 width=23)
  ->  Hash  (cost=1.05..1.05 rows=1 width=14)
        ->  Seq Scan on dept  (cost=0.00..1.05 rows=1 width=14)
              Filter: ((loc)::text = 'CHICAGO'::text)
```

Se lit :

Parcourir dept séquentiellement, en ne gardant que les enregistrements pour lesquels loc='CHICAGO'

- Coût pour récupérer le premier enregistrement : 0
- Coût pour récupérer le dernier enregistrement:1.05
- 1 enregistrement produit

Un plan simple

```
Hash Join  (cost=1.06..2.29 rows=4 width=27)
  Hash Cond: (emp.deptno = dept.deptno)
  ->  Seq Scan on emp  (cost=0.00..1.14 rows=14 width=23)
  ->  Hash  (cost=1.05..1.05 rows=1 width=14)
        ->  Seq Scan on dept  (cost=0.00..1.05 rows=1 width=14)
              Filter: ((loc)::text = 'CHICAGO'::text)
```

Se lit :

Mettre ces enregistrement dans une table de hachage

- Coût de production du premier enregistrement : 1.05
- Coût de production du dernier enregistrement : 1.05
- 1 enregistrement produit

DALIBO L'Expertise PostgreSQL

Un plan simple

```
Hash Join  (cost=1.06..2.29 rows=4 width=27)
  Hash Cond: (emp.deptno = dept.deptno)
  ->  Seq Scan on emp  (cost=0.00..1.14 rows=14 width=23)
  ->  Hash  (cost=1.05..1.05 rows=1 width=14)
        ->  Seq Scan on dept  (cost=0.00..1.05 rows=1 width=14)
              Filter: ((loc)::text = 'CHICAGO'::text)
```

Se lit :

Lire séquentiellement emp

- Coût de production du premier enregistrement : 0
- Coût de production du dernier enregistrement : 1.14
- 14 enregistrements produits

Un plan simple

```
Hash Join  (cost=1.06..2.29 rows=4 width=27)
  Hash Cond: (emp.deptno = dept.deptno)
  ->  Seq Scan on emp  (cost=0.00..1.14 rows=14 width=23)
  ->  Hash  (cost=1.05..1.05 rows=1 width=14)
        ->  Seq Scan on dept  (cost=0.00..1.05 rows=1 width=14)
              Filter: ((loc)::text = 'CHICAGO'::text)
```

Se lit :

Joindre emp en le parcourant séquentiellement au hachage du scan séquentiel de dept

- Coût de production du premier enregistrement : 1.05
- Coût de production du dernier enregistrement : 2.29
- 4 enregistrements produits

L'optimiseur a bon ?

Option ANALYZE à EXPLAIN

```
EXPLAIN (ANALYZE) SELECT empno,ename,job,dname
FROM emp JOIN dept
  ON (emp.deptno=dept.deptno)
WHERE loc='CHICAGO';
                                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Hash Join  (cost=1.06..2.29 rows=4 width=27) (actual time=0.050..0.069 rows=6 loops=1)
   Hash Cond: (emp.deptno = dept.deptno)
   ->  Seq Scan on emp  (cost=0.00..1.14 rows=14 width=23) (actual time=0.008..0.016 rows=14 loops=1)
   ->  Hash  (cost=1.05..1.05 rows=1 width=14) (actual time=0.021..0.021 rows=1 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 1kB
         ->  Seq Scan on dept  (cost=0.00..1.05 rows=1 width=14) (actual time=0.013..0.015 rows=1 loops=1)
               Filter: ((loc)::text = 'CHICAGO'::text)
               Rows Removed by Filter: 3
 Total runtime: 0.128 ms
```

Il ne s'est pas trompé sur l'estimation du nombre d'enregistrements...

DALIBO L'Expertise PostgreSQL

L'optimiseur a bon ?

- Sur les plans longs, on peut aussi regarder le temps d'exécution des étapes par rapport à leur coût estimé

```
EXPLAIN (ANALYZE) SELECT empno,ename,job,dname
FROM emp JOIN dept
  ON (emp.deptno=dept.deptno)
WHERE loc='CHICAGO';
                                              QUERY PLAN
-----------------------------------------------------------------------------------------------------------
 Hash Join  (cost=1.06..2.29 rows=4 width=27) (actual time=0.050..0.069 rows=6 loops=1)
   Hash Cond: (emp.deptno = dept.deptno)
   ->  Seq Scan on emp  (cost=0.00..1.14 rows=14 width=23) (actual time=0.008..0.016 rows=14 loops=1)
   ->  Hash  (cost=1.05..1.05 rows=1 width=14) (actual time=0.021..0.021 rows=1 loops=1)
         Buckets: 1024  Batches: 1  Memory Usage: 1kB
         ->  Seq Scan on dept  (cost=0.00..1.05 rows=1 width=14) (actual time=0.013..0.015 rows=1 loops=1)
               Filter: ((loc)::text = 'CHICAGO'::text)
               Rows Removed by Filter: 3
 Total runtime: 0.128 ms
```


DALIBO
L'Expertise PostgreSQL

À la main ?

explain.depesz.com

explain.depesz.com

A tool for finding a real cause for slow queries.

new explain · history · help · about · contact · Donate

Result: rFS

options

HTML · TEXT · STATS

#	exclusive	inclusive	rows x	rows	loops	node
1.	0.032	0.069	↓ 1.5	6	1	→ Hash Join (cost=1.06..2.29 rows=4 width=27) (actual time=0.050..0.069 rows=6 loops=1) Hash Cond: (emp.deptno = dept.deptno)
2.	0.016	0.016	↑ 1.0	14	1	→ Seq Scan on emp (cost=0.00..1.14 rows=14 width=23) (actual time=0.008..0.016 rows=14 loops=1)
3.	0.006	0.021	↑ 1.0	1	1	→ Hash (cost=1.05..1.05 rows=1 width=14) (actual time=0.021..0.021 rows=1 loops=1) Buckets: 1024 Batches: 1 Memory Usage: 1kB
4.	0.015	0.015	↑ 1.0	1	1	→ Seq Scan on dept (cost=0.00..1.05 rows=1 width=14) (actual time=0.013..0.015 rows=1 loops=1) Filter: ((loc)::text = 'CHICAGO'::text) Rows Removed by Filter: 3

Problèmes courants

SELECT * FROM t1 WHERE c1=1 AND c2=1

- c1 et c2 corrélés (code postal et ville)
- Pas de statistiques là-dessus pour le moment
- Si possible, éviter les clauses where sur les colonnes inutiles
- Créer une colonne supplémentaire ou un index fonction des colonnes corrélées et interroger dessus :-(
- Un des sujets de développement: http://wiki.postgresql.org/wiki/Cross_Columns_Stats

Questions ?

Marc Cousin, Dalibo

marc.cousin@dalibo.com

Problèmes courants

```
SELECT * FROM t1 WHERE c1+0 = c1;
SELECT * FROM t1 WHERE c2||'' = c2;
```

- N'utilise pas les index
- N'utilise pas les statistiques, estimé à 0,5 % des enregistrements totaux
- Mauvaise pratique de toutes façons
- Si pas le choix, index sur fonction, qui aura les statistiques

Problèmes courants

SELECT * FROM t1 WHERE c2 LIKE 'x%';

- varchar_pattern_ops, text_pattern_ops
- CREATE INDEX idx ON t1 (c2 varchar_pattern_ops)
- Attention au collationnement (9.1+)

Problèmes courants

```
CREATE FUNCTION … IMMUTABLE | STABLE | VOLATILE
```

- IMMUTABLE : à paramètres donnés, la fonction retourne toujours la même valeur (addition, concaténation, logarithme…)
- STABLE : à paramètres donnés, dans une requête, retourne le même résultat, et ne modifie pas la base (fonction de recherche dans la base)
- VOLATILE : le reste

DALIBO L'Expertise PostgreSQL

Volatilité

```
create function plusdeux (a int) returns int language plpgsql as
$$
BEGIN
  RETURN a+2;
END
$$;
```

```
explain analyze select * from t1 where c1<plusdeux(5);
```

```
                                            QUERY PLAN
------------------------------------------------------------------------------
Seq Scan on t1  (cost=0.00..267.50 rows=333 width=4) (actual time=0.095..5.037
rows=6 loops=1)
   Filter: (c1 < plusdeux(5))
   Rows Removed by Filter: 994
Total runtime: 5.087 ms
```

Exécuté pour chaque ligne !

DALIBO L'Expertise PostgreSQL

Volatilité

```
create function plusdeux (a int) returns int stable language plpgsql as
$$
BEGIN
  RETURN a+2;
END
$$;

explain analyze select * from t1 where c1<plusdeux(5);
```

```
                                          QUERY PLAN
--------------------------------------------------------------------------------
 Index Only Scan using t1_pkey on t1  (cost=0.25..8.62 rows=7 width=4) (actual
time=0.050..0.051 rows=6 loops=1)
   Index Cond: (c1 < plusdeux(5))
   Heap Fetches: 6
 Total runtime: 0.074 ms
```

C'est une constante.
On peut utiliser un index.
Mais le planner ne connaît pas sa valeur au moment du calcul du plan

Presque 100 fois plus rapide

DALIBO L'Expertise PostgreSQL

Volatilité

```
create function plusdeux (a int) returns int immutable language plpgsql as
$$
BEGIN
  RETURN a+2;
END
$$;

explain analyze select * from t1 where c1<plusdeux(5);
```

```
                                  QUERY PLAN
--------------------------------------------------------------------------------
 Index Only Scan using t1_pkey on t1  (cost=0.00..8.37 rows=7 width=4) (actual
time=0.011..0.015 rows=6 loops=1)
   Index Cond: (c1 < 7)
   Heap Fetches: 6
 Total runtime: 0.051 ms
```

C'est une constante, connue dès la planification. On peut donc choisir le meilleur plan possible

DALIBO L'Expertise PostgreSQL