PostgreSQL, jouons avec l'optimiseur

Pierre Ducroquet
31 mai 2016

Objectifs de cette présentation

- Comprendre comment PostgreSQL choisit ses plans d'exécution
- Savoir identifier et corriger les parties coûteuses dans les requêtes
- Jouer sur les bons paramètres pour influencer le choix du plan

Notre première requête

```
SELECT id FROM demo_pgday WHERE id = 42;
```

- Combien de plans trouvez-vous ?

Notre première requête

```
SELECT id FROM demo_pgday WHERE id = 42;
```

- Combien de plans trouvez-vous ?
  - On ne peut pas répondre à cette question sans le schéma :
    - demo_pgday(id integer)
    - create index on demo_pgday(id);

Demandons à PostgreSQL

```
EXPLAIN ANALYZE SELECT id FROM demo_pgday WHERE id = 42;

                                QUERY PLAN
-------------------------------------------------------------------------
 Index Only Scan using demo_pgday_id_idx on demo_pgday  (cost=0.43..8.45
rows=1 width=4) (actual time=0.011..0.011 rows=1 loops=1)
   Index Cond: (id = 42)
   Heap Fetches: 1
 Planning time: 0.041 ms
 Execution time: 0.023 ms
(5 rows)
```

Demandons à PostgreSQL

```
set enable_indexonlyscan to off;
EXPLAIN ANALYZE SELECT id FROM demo_pgday WHERE id = 42;

                                QUERY PLAN
-----------------------------------------------------------------------
 Index Scan using demo_pgday_id_idx on demo_pgday  (cost=0.43..8.45
rows=1 width=4) (actual time=0.013..0.014 rows=1 loops=1)
   Index Cond: (id = 42)
 Planning time: 0.041 ms
 Execution time: 0.024 ms
(4 rows)
```

Demandons à PostgreSQL

```
set enable_indexscan to off;
EXPLAIN ANALYZE SELECT id FROM demo_pgday WHERE id = 42;

                            QUERY PLAN
------------------------------------------------------------------
 Bitmap Heap Scan on demo_pgday  (cost=4.44..8.45 rows=1 width=4) (actual time=0.015..0.016 rows=1 loops=1)
   Recheck Cond: (id = 42)
   Heap Blocks: exact=1
   ->  Bitmap Index Scan on demo_pgday_id_idx  (cost=0.00..4.44 rows=1 width=0) (actual time=0.010..0.010 rows=1 loops=1)
         Index Cond: (id = 42)
 Planning time: 0.041 ms
 Execution time: 0.030 ms
(7 rows)
```

Demandons à PostgreSQL

```
set enable_bitmapscan to off;
EXPLAIN ANALYZE SELECT id FROM demo_pgday WHERE id = 42;

                                QUERY PLAN
-------------------------------------------------------------------------
 Seq Scan on demo_pgday  (cost=0.00..33850.01 rows=1 width=4) (actual
time=0.021..159.371 rows=1 loops=1)
   Filter: (id = 42)
   Rows Removed by Filter: 2000000
 Planning time: 0.089 ms
 Execution time: 159.393 ms
(5 rows)
```

Les plans pour cette requête simple

- Seq Scan
- Bitmap Heap Scan
- Index Scan
- Index Only Scan

Les options enable_seqscan, enable_indexscan et autres sont à but de debug uniquement !

Comment PostgreSQL choisit ?

- Chaque requête possible est évaluée et un coût lui est associé
- Le coût n'a pas d'unité
- Les options comme enable_seqscan changent arbitrairement le coût de l'opération pour l'exclure de fait
- On choisit la requête la moins coûteuse…

Les paramètres de calcul des coûts

- Les paramètres les plus connus : random_page_cost et seq_page_cost
- Les plus rares : cpu_tuple_cost, cpu_index_tuple_cost et cpu_operator_cost
- L'oublié : effective_cache_size (valeur par défaut changée en 9.4)

Ça se corse…

- `SELECT * FROM account WHERE login LIKE 'ab1%' LIMIT 1000;`
- `SELECT * FROM account WHERE login LIKE 'ab1%' ORDER BY id LIMIT 1000;`
- Pourquoi la deuxième requête prend 3 secondes quand la première prend 1 milliseconde ?

Le pouvoir des statistiques

- L'optimiseur doit savoir le résultat des opérations pour connaître le coût des nœuds parents… Impossible !
- Les statistiques compensent en fournissant un échantillon de données

pg_stats

- Vue système indiquant notamment…
  - n_distinct : le nombre de valeurs distinctes (divisé par le nombre de lignes si négatif, pour représenter un contenu qui «varie»)
  - most_common_vals et freqs : les valeurs les plus fréquentes
  - histogram_bounds : répartition de la population
  - correlation : corrélation entre l'ordre physique et l'ordre logique
  - most_common_elems, freqs et histogram : pour les tableaux

Et si il se trompe ?

- L'analyseur a un seuil, sinon pg_statistics serait plus lourd que la table… Mais il peut être trop bas.
  - En cas de large déséquilibre des valeurs
  - Sur une antique version de PostgreSQL (< 8.4) avec une cible trop basse
- default_statistics_target à 100
- `ALTER TABLE tbl ALTER COLUMN c SET STATISTICS 42;`
- Démonstration !

Démonstration...

- Table words : id serial, w character varying
- Contient l’ensemble des mots des release notes de PostgreSQL 9.4

Démonstration...

```
explain analyze select * from words where w = 'The';
explain analyze select * from words where w = 'the';
```

- On remarque que ces deux requêtes ont fait un index scan.
- Les statistiques nous montrent que « the » représente 3,4 % des mots alors que « The » est dérisoire (moins de 0,1%)

Démonstration...

- Quand on ajoute un `limit 5` aux requêtes, PostgreSQL va choisir un Seq Scan pour « the » qui est très fréquent
  - Il estime qu'il va trouver plus vite des données en lisant la table qu'en lisant l'index...

Et les jointures dans tout ça ?

- Les jointures multiplient les plans d'exécution possible
  - L’ordre des jointures doit être déterminé et optimisé...
- À partir de 12 tables à joindre dans une requête, GEQO se déclenche

Et les jointures dans tout ça ?

- Hash-join
  - H = Hash(t1) ; for row in t2 { join(row, H) }
- Mergejoin
  - Sort t1 ; sort t2 ; for row in (t1 & t2) { join (r1, r2) }
- Nested loop
  - For each row1 in t1 { for each row2 in t2 { join(row1, row2) } }

Et les jointures dans tout ça ?

- Les formules montrent l'importance des statistiques pour l'optimisation
- Une mauvaise estimation du nombre de lignes peut donner un plan désastreux

Comment corriger un problème

- PostgreSQL ne gère pas les «optimization hints»
- Signaler les bugs (et être à jour)
- Penser à l'organisation des données logique et physique
- Réécrire les requêtes, notamment avec des CTE, permet de contourner beaucoup de problèmes
- Évitez les ORM idiots qui ne font que des SELECT *...

Merci pour votre attention !

Des questions ?