SGBD : Bases de données avancées [M3106C]

Hocine ABIR

20 mars 2014

IUT Villetaneuse
E-mail: abir@iutv.univ-paris13.fr

TABLE DES MATIÈRES

2 Optimisation des Requêtes sous `PostgreSQL` 1
2.1 Génération du Plan d'Exécution 1
2.2 Coût Temporel 3
2.3 Opérateurs 5
2.4 Jointures Explicites 5
2.5 Paramètres Dynamiques 8
2.6 Statistiques 9
2.7 Maintenance d'une Base de Données 12

CHAPITRE

DEUX

Optimisation des Requêtes sous PostgreSQL

2.1 Génération du Plan d'Exécution

2.1.1 Commande `EXPLAIN`

La commande `EXPLAIN` :

`EXPLAIN [ANALYZE] query`

1. Permet de visualiser le plan d'exécution généré pour la requête `query` et les coûts de chaque étape (sous-plan),
2. Ne peut être utilisée que pour des requêtes *DML* (et non *DDL*) : SELECT, INSERT, UPDATE, DELETE, EXECUTE, ou DECLARE.
3. Montre les coûts, le nombre de tuples et la taille des tuples.
4. L'option `ANALYZE` permet d'exécuter la requête `query` pour déteminer le temps effectif de chaque étape du plan.

Le plan d'exécution est présenté suivant le parcours préfixe, c'est à dire :

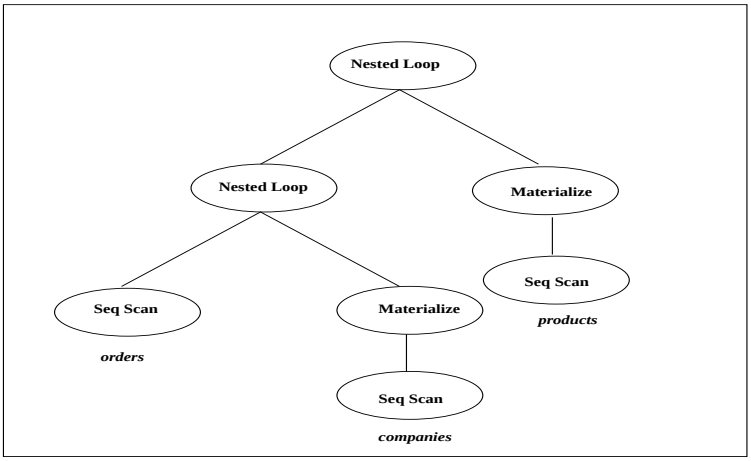
->Description du Noeud N
    -> Description du Fils Gauche de N
    -> Description du Fils Droit de N

Une description de Noeud comporte :
- La description de l'opération suivi de
- Quatre estimations de coûts :
    1. Coût Temporel : temps est mesuré en *blocs* disques.
        (a) Temps de réponse (startup cost) : temps nécessaire pour obtenir le premier tuple de l'opération,
        (b) Temps total de l'opération.
    2. Coût Spacial
        (a) Taille du résulat : nombre de *tuples* produits par l'opération (sélectivité de l'opérateur)
        (b) Taille moyenne des tuples produits en *octets*

2.1.2 Exemple EXPLAIN

```
=> EXPLAIN SELECT co_name,pr_desc,ord_qty
    FROM companies,orders,products
    WHERE co_id=ord_company
      AND   pr_code=ord_product
      AND  ord_qty > 60;
```

```
                                QUERY PLAN
-------------------------------------------------------------------------------
 Nested Loop (cost=4.21..955.31 rows=376 width=52)
   Join Filter: (("inner".pr_code)::text = ("outer".ord_product)::text)
   -> Nested Loop (cost=3.10..869.60 rows=376 width=39)
         Join Filter: ("inner".co_id = "outer".ord_company)
         -> Seq Scan on orders (cost=0.00..20.50 rows=376 width=17)
               Filter: (ord_qty > 60)
         -> Materialize (cost=3.10..4.10 rows=100 width=30)
               -> Seq Scan on companies (cost=0.00..3.00 rows=100 width=30)
   -> Materialize (cost=1.11..1.21 rows=10 width=31)
         -> Seq Scan on products (cost=0.00..1.10 rows=10 width=31)
(10 rows)
```



2.1.3 Exemple EXPLAIN ANALYZE

```
=> EXPLAIN analyze SELECT co_name,pr_desc,ord_qty
        FROM companies,orders,products
        WHERE co_id=ord_company
           AND   pr_code=ord_product
           AND  ord_qty > 60;
```

```
                                QUERY PLAN
-----------------------------------------------------------------------------
 Nested Loop (cost=4.21..955.31 rows=376 width=52)
             (actual time=0.527..131.395 rows=380 loops=1)
   Join Filter: (("inner".pr_code)::text = ("outer".ord_product)::text)
   -> Nested Loop (cost=3.10..869.60 rows=376 width=39)
                  (actual time=0.455..118.936 rows=380 loops=1)
         Join Filter: ("inner".co_id = "outer".ord_company)
```

```
9         ->  Seq Scan on orders (cost=0.00..20.50 rows=376 width=17)
10                            (actual time=0.031..1.484 rows=380 loops=1)
11                Filter : (ord_qty > 60)
12        ->  Materialize  (cost=3.10..4.10 rows=100 width=30)
13                         (actual time=0.001..0.132 rows=100 loops=380)
14              ->  Seq Scan on companies (cost=0.00..3.00 rows=100 width=30)
15                         (actual time=0.005..0.213 rows=100 loops=1)
16    ->  Materialize  (cost=1.11..1.21 rows=10 width=31)
17                     (actual time=0.001..0.014 rows=10 loops=380)
18        ->  Seq Scan on products (cost=0.00..1.10 rows=10 width=31)
19                               (actual time=0.005..0.028 rows=10 loops=1)
20  Total runtime: 150.390 ms
21 (11 rows)
```

- `loops` : indique le nombre de fois que l'opérateur est exécuté et pour chaque exécution on a :
- `actual time` : temps d'exécution de l'opérateur
- `rows` : tuples produit lors de l'exécution de l'opérateur

2.2 Coût Temporel

2.2.1 Paramètres des Coûts

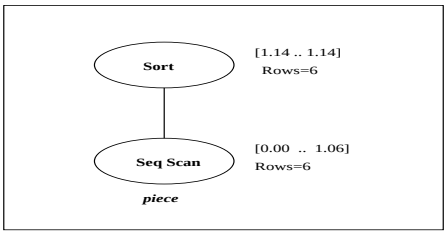
Paramètre	Description	Valeur par défaut
cpu_tuple_cost	temps de traitement d'un tuple	0.01 (1/100 bloc)
cpu_index_tuple_cost	temps de traitement d'un tuple d'un index	0.001
cpu_operator_cost	temps de traitement d'un opérateur de la clause WHERE	0.0025
random_page_cost	temps de lecture non-séquentielle d'un bloc	4

2.2.2 Exemple

```
=> select * from piece;
 pi_numero | pi_nom  | pi_poids | pi_couleur
-----------+---------+----------+------------
 P1        | Vis6    |    12.60 | rouge
 P2        | Vis8    |    17.00 | vert
 P3        | Boulon7 |    17.20 | bleu
 P4        | Came    |    14.00 | rouge
 P5        | Came    |    12.10 | bleu
 P6        | Ressort |    18.30 | rouge
(6 rows)
```

```
=> explain select * from piece;
                      QUERY PLAN
-------------------------------------------------------
 Seq Scan on piece  (cost=0.00..1.06 rows=6 width=34)
(1 row)

=> EXPLAIN SELECT * FROM piece ORDER BY  pi_numero;
                         QUERY PLAN
------------------------------------------------------------
 Sort  (cost=1.14..1.15 rows=6 width=34)
   Sort Key: pi_numero
   ->  Seq Scan on piece  (cost=0.00..1.06 rows=6 width=34)
(3 rows)
```




```
=> SELECT reltuples,relpages
    FROM  pg_class
    WHERE relname='piece';
 reltuples | relpages
-----------+----------
         6 |        1
(1 row)

=> SHOW CPU_TUPLE_COST;
 cpu_tuple_cost
----------------
 0.01
(1 row)
```

Le coût d'une opération est déterminé par :

$$Cout = Nombre\_de\_Blocs\_de\_la\_Relation$$
$$+Nombre\_de\_Tuples\_de\_la\_Relation \times \texttt{CPU\_TUPLE\_COST}$$

Soit : $1 + 6 \times 0.01 = 1.06$

Plusieurs plans peuvent être générés pour une même requête :
- l'évolution de la base de données : statistiques, chemins d'accès, etc. peuvent modifier les choix effectués par l'optimiseur,
- l'utilisateur peut donner des directives à l'optimiseur pour le forcer à effectuer d'autres choix.

2.3 Opérateurs

Opérateur	Description	Temps de Réponse
Seq Scan		Non
Sort	ORDER BY, Unique, etc	Oui
Index Scan	sauf hash index	Non
Result	requête sur une non-table, constante dans WHERE	Non
Unique	DISTINCT, UNION	Oui
Aggregate	COUNT, SUM, MIN, MAX, AVG, etc	Oui
Group	clause GROUP BY	Oui
Append	UNION, héritage	Non
Nested Loop	INNER JOIN, LEFT OUTER JOIN	Non
Merge Join	INNER JOIN, LEFT OUTER JOIN, RIGHT OUTER JOIN	Oui
Hash	opérateur Hash Join	Oui
Hash Join	INNER JOIN, LEFT OUTER JOIN	Oui
Subquery Scan	UNION	Non
Tid Scan	requête avec "ctid ="	Non
Materialize	subselects, etc	Oui
Function Scan	fonctions	Non
SetOp Intersect	INTERSECT	Oui
SetOp Intersect All	INTERSECT ALL	Oui
SetOp Except	EXCEPT	Oui
SetOp Except All	EXCEPT ALL	Oui

2.4 Jointures Explicites

Une jointure explicite est un moyen syntaxique de choisir un plan d'exécution. Ce choix peut porter :

1. la priorite des jointures
2. le choix de la table droite(inner) ou gauche (outer)

```
SELECT ...
    FROM from_item , ...
    WHERE condition

from_item :
```

```
->     table
->     from_item JOIN from_item ON condition
->     sub_select
```

2.4.1 Paramètres

Sous-Requêtes et Vues : `from_collapse_limit`

Quand une requête comporte des sous-requêtes :
- `SELECT`s ou
- Vues

Ces sous-requêtes sont fusionnées avec la requête parente. Cette reécriture est controlée par le paramètre `from_collapse_limit` dont la valeur par défaut est 8 :

```
# show from_collapse_limit;
from_collapse_limit
---------------------
8
(1 row)
```

Exemple :

```
SELECT *
  FROM tab1, tab2,
       (SELECT *
           FROM tab3, tab4, tab5
           WHERE conditions1) AS ss
  WHERE conditions2;
```

L'optimiseur reécrit la requête en :

```
SELECT *
  FROM tab1, tab2, tab3, tab4, tab5
WHERE conditions1
  AND conditions2;
```

Cette reécriture permet à l'optimiseur de générer un meilleur plan d'exécution en augmenant son espace de recherche. Par la même occasion, le temps pour générer ce plan augmente ! Dans l'exemple, la clause `FROM` passe de deux tables à cinq tables.

Opérateurs `JOIN`s : `join_collapse_limit`

Les requêtes comportant des jointures explicites (opérateurs `JOIN`, sauf `FULL JOIN`s) sont reécrites sous forme d'une liste d'items de clause `FROM`. Le nombre d'items limite de la clause `FROM` résultante est controlée par le paramètre `join_collapse_limit`. Par défaut :

```
# SHOW join_collapse_limit;
 join_collapse_limit
---------------------
 8
(1 row)
```

Pour forcer l'optimiseur à respecter l'ordre des jointures explicites, `join_collapse_limit` doit être positionné à 1 (pas de reécriture) :

```
# SET join_collapse_limit to 1;
```

2.4.2 Priorité des jointures

On considère le plan suivant :

```
=> explain select fo_nom,pi_nom,co_quantite
             from fournisseur,piece,commande
             where fo_numero=co_fournisseur
                 and pi_numero=co_piece;
```

```
                              QUERY PLAN
-----------------------------------------------------------------------
 Nested Loop (cost=0.00..12.29 rows=5 width=47)
   Join Filter: ("inner".fo_numero = "outer".co_fournisseur)
   -> Nested Loop (cost=0.00..6.73 rows=5 width=31)
         Join Filter: ("inner".pi_numero = "outer".co_piece)
         -> Seq Scan on commande (cost=0.00..1.05 rows=5 width=16)
         -> Seq Scan on piece (cost=0.00..1.06 rows=6 width=15)
   -> Seq Scan on fournisseur (cost=0.00..1.05 rows=5 width=16)
(7 lignes)
```

L' expression algébrique évaluée est :

$$(\, commande \bowtie piece \,) \bowtie fournisseur \tag{2.1}$$

Si on veut obtenir le plan de l'expression (2.2) à partir de (2.1) :

$$(\, fournisseur \bowtie commande \,) \bowtie piece \tag{2.2}$$

on doit rendre explicite l'expression (2.2) dans la requête SQL :

```
=> explain select fo_nom,pi_nom,co_quantite
        from piece join
             ( fournisseur join commande on fo_numero=co_fournisseur)
             on pi_numero=co_piece;
```

```
                              QUERY PLAN
-----------------------------------------------------------------------
 Nested Loop (cost=0.00..12.29 rows=5 width=47)
   Join Filter: ("inner".pi_numero = "outer".co_piece)
   -> Nested Loop (cost=0.00..6.61 rows=5 width=32)
         Join Filter: ("outer".fo_numero = "inner".co_fournisseur)
         -> Seq Scan on fournisseur (cost=0.00..1.05 rows=5 width=16)
         -> Seq Scan on commande (cost=0.00..1.05 rows=5 width=16)
```

```
   -> Seq Scan on piece (cost=0.00..1.06 rows=6 width=15)
(7 lignes)
```

2.4.3 Table Inner-Outer

On considère le plan suivant :

```
=> explain select fo_nom,co_quantite
           from fournisseur, commande
           where fo_numero=co_fournisseur;
```

```
                          QUERY PLAN
------------------------------------------------------------------
 Nested Loop (cost=0.00..6.61 rows=5 width=26)
   Join Filter: ("outer".fo_numero = "inner".co_fournisseur)
   -> Seq Scan on fournisseur (cost=0.00..1.05 rows=5 width=16)
   -> Seq Scan on commande (cost=0.00..1.05 rows=5 width=10)
(4 lignes)
```

L' expression algébrique évaluée est :

$$fournisseur \bowtie commande \tag{2.3}$$

Si on veut obtenir le plan de l'expression (2.4) à partir de (2.3) :

$$commande \bowtie fournisseur \tag{2.4}$$

on doit rendre explicite l'expression (2.4) dans la requête SQL comme suit :

```
=> explain select fo_nom,co_quantite
             from (select * from commande) as c , fournisseur
             where fo_numero=co_fournisseur;
```

```
                          QUERY PLAN
------------------------------------------------------------------
 Nested Loop (cost=0.00..6.61 rows=5 width=26)
   Join Filter: ("inner".fo_numero = "outer".co_fournisseur)
   -> Seq Scan on commande (cost=0.00..1.05 rows=5 width=10)
   -> Seq Scan on fournisseur (cost=0.00..1.05 rows=5 width=16)
(4 lignes)
```

2.5 Paramètres Dynamiques

2.5.1 Méthodes d'accès

1. ENABLE_SEQSCAN
2. ENABLE_TIDSCAN
3. ENABLE_INDEXSCAN
4. ENABLE_BITMAPSCAN

2.5.2 Opérateurs

1. `ENABLE_NESTLOOP`
2. `ENABLE_HASHJOIN`
3. `ENABLE_MERGEJOIN`
4. `ENABLE_SORT`
5. `ENABLE_BITMAPJOIN`

2.5.3 Commande `SET`

modifier la valeur d'un paramètre.

2.5.4 Commande `SHOW`

afficher la valeur d'un paramètre.

2.6 Statistiques

2.6.1 Vue des Statistiques `pg_stats`

Nom	Desciption
attname	colonne décrite
tablename	Nom de la table contenant la colonne
null_frac	Fraction des valeurs nulles
avg_width	taille moyenne (bytes) des données de la colonne
n_distinct	>0 : nombre estimé de valeur distinctes de la colonne. <0 : - Fraction du nombre estimé de valeur distinctes de la colonne
most_common_vals	liste des valeurs les plus répondues de la colonne. (si omise alors aucune valeur n'est plus commune que les autres.)
most_common_freqs	liste des fréquences des valeurs les plus communes, i.e., number of occurrences of each divided by total number of rows.
histogram_bounds	liste de valeurs qui divise les valeurs de la colonne en groupes ayant approximativement une même cardinalité

2.6.2 Configuration

Paramètre	Desciption
STATS_START_COLLECTOR	Controle la génération dynamique de statistiques
STATS_COMMAND_STRING	génération de statistiques sur la commande en cours (`pg_stat_activity`)
STATS_BLOCK_LEVEL	génération de statistiques sur les blocs
STATS_ROW_LEVEL	génération de statistiques sur les tuples

2.6.3 Exemples

Exemple 1

```
# explain select * from etudiant;
                        QUERY PLAN
------------------------------------------------------------
 Seq Scan on etudiant  (cost=0.00..8.42 rows=142 width=95)
(1 row)
```

```
# SELECT reltuples, relpages FROM pg_class
# WHERE relname = 'etudiant';
 reltuples | relpages
-----------+----------
       142 |        7
(1 row)
```

```
# explain select * from etudiant where et_id<140;
                        QUERY PLAN
------------------------------------------------------------
 Seq Scan on etudiant  (cost=0.00..8.78 rows=39 width=95)
   Filter: (et_id < 140)
(2 rows)
```

```
# SELECT histogram_bounds FROM pg_stats
# WHERE tablename='etudiant' AND attname='et_id';
               histogram_bounds
-----------------------------------------------
 {101,115,129,144,158,172,186,200,214,228,243}
(1 row)
```

L'histogramme divise l'intervalle des valeurs de *et_id* en classes de même fréquence, le planneur localise la classe de la valeur de la clause *where* et comptabilise une partie de cette classe et les classes précédente.

La valeur 140 est dans la troisième (3) classe : (129,144). En suppossant une distribution linéaire des valeurs dans chaque classe, le planneur détermine la sélectivité comme suit :

```
selectivity = (2 + (142 - clas[3].min)/
                   (clas[3].max - clas[3].min))/num_clas
            = (2 + (142 - 129)/(144 - 129))/10
            = (2 + 13/15)/10 = 0.28
```

c'est à dire, 2 classes plus une fraction linéaire de la troisième, divisée par le nombre de classes.

Le nombre estimé de tuples peut être calculé comme le produit de la sélectivité et de la cardinalité de *etudiant* :

```
rows = rel_cardinality * selectivity
     = 142 * 0.28
     = 39
```

Exemple 2

```
# explain select * from etudiant where et_prenom='Kevin';
                       QUERY PLAN
-------------------------------------------------------------
 Seq Scan on etudiant  (cost=0.00..8.78 rows=3 width=95)
   Filter: ((et_prenom)::text = 'Kevin'::text)
(2 rows)
```

Les valeurs les plus communes (most common values) MCVs, sont utilisées pour déterminer la sélectivité.

```
# SELECT null_frac, n_distinct, most_common_vals, most_common_freqs
# FROM pg_stats
# WHERE tablename='etudiant' AND attname='et_prenom';

null_frac        |  0
n_distinct       |  -0.788732
most_common_vals | {Julien,Guillaume,Kevin,Mehdi,Mickael,
                      Mohamed,Romain,Sebastien,Benoit,Damien}
most_common_freqs| {0.0352113,0.0211268,0.0211268,0.0211268,0.0211268,
                      0.0211268,0.0211268,0.0211268,0.0140845,0.0140845}
(1 row)


selectivity = mcf[3]
            = 0.021
rows  = 142 * 0.021
      = 3
```

Exemple 3

Considérons la même requête avec un nom d'étudiant qui n'est pas dans la liste MCV :

```
# explain select * from etudiant where et_prenom='xxxx';
                       QUERY PLAN
-------------------------------------------------------------
 Seq Scan on etudiant  (cost=0.00..8.78 rows=1 width=95)
   Filter: ((et_prenom)::text = 'xxxx'::text)
(2 rows)
```

Comment estimer la selectivité quand la valeur n'est pas dans la liste MCV ?

```
selectivity = (1 - sum(mcf))/(num_distinct - num_mcv)
            = (1 - (0.0352113 + 0.0211268 + 0.0211268
      + 0.0211268 + 0.0211268 + 0.0211268
      + 0.0211268 + 0.0211268 + 0.0140845
      + 0.0140845)/( (0.788732*142) - 10)
            = (1 - 0.211268)/(102)
            = (1 -  0.211268)/102)=1.14084506593645/112
    = 0.00704225352300065

rows = 142 * 0.00704225352300065
     =1
```

2.7 Maintenance d'une Base de Données

Durant l'activité d'une base de données, un grand nombre de tuples sont ajoutés, ou supprimés, la commande `SQL VACUUM` et `ANALYZE` permettent d'assurer deux fonctions :

1. Mettre à jour les données : supprimer toutes les données désuètes
2. Mettre à jour le profile des données : analyse de l'activité de la base de données pour assister le planner dans la génération de plans d'exécution performants.

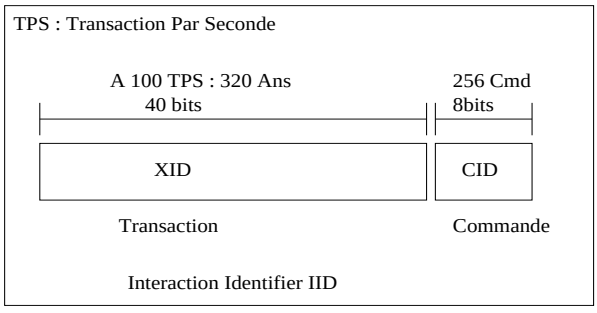
Deux niveaux de maintenance :

2.7.1 Physique

Gestion de l'utilisation de l'espace disque. Cette maintenance est assurée par la commande `SQL VACUUM`.

Version de Données

Identifiant de Transaction XID



2.7.2 Analytique

Pour accroitre les performances et se protéger contre des pertes de données. Cette maintenance est assurée par la commande `SQL VACUUM ANALYZE` ou `SQL ANALYZE`.

pg_stat

pg_class