Bases de données

Polytech Paris-Sud

Apprentis 4ème année

Cours 6 : Prise en main de Postgresql

kn@lri.fr
http://www.lri.fr/~kn

Plan

1 Rappels ✔
2 Stockage ✔
3 Indexation ✔
4 Optimisation des opérateurs ✔
5 Optimisation de requêtes ✔
6 Prise en main de Postgresql
- 6.1 Base d'exemple
- 6.2 EXPLAIN
- 6.3 Exemples d'opérateurs
- 6.4 Démo

On considère la base d'exemple suivante

```
CREATE TABLE PEOPLE (pid INTEGER PRIMARY key,
                     firstname VARCHAR(30),
                     lastname VARCHAR(30));

CREATE TABLE MOVIE (mid INTEGER PRIMARY KEY,
                    title VARCHAR(90) NOT NULL,
                    year INTEGER NOT NULL,
                    runtime INTEGER NOT NULL,
                    rank INTEGER NOT NULL);

CREATE TABLE ROLE (mid INTEGER REFERENCES MOVIE,
                   pid INTEGER REFERENCES PEOPLE,
                   name VARCHAR(70),
                   PRIMARY KEY(mid, pid, name));

CREATE TABLE DIRECTOR (mid INTEGER REFERENCES MOVIE,
                       pid INTEGER REFERENCES PEOPLE,
                       PRIMARY KEY (mid, pid));
```


Plan

1 Rappels ✔
2 Stockage ✔
3 Indexation ✔
4 Optimisation des opérateurs ✔
5 Optimisation de requêtes ✔
6 Prise en main de Postgresql
- 6.1 Base d'exemple ✔
- 6.2 EXPLAIN
- 6.3 Exemples d'opérateurs
- 6.4 Démo

EXPLAIN ANALYSE

On peut demander à Postgresql d'afficher le plan qu'il calcule pour une requête avec les esitmations de coût :

```
EXPLAIN requête;
```

Dans ce cas, la requête n'est pas évaluée. On peut aussi évaluer la requête :

```
EXPLAIN ANALYSE requête;
```

Dans ce cas, la requête est évaluée et les coût réels sont affichés. S'ils divergent des coûts estimés, l'optimiseur s'est trompé, par exemple parce que ses statistiques ne sont pas à jour

EXPLAIN ANALYSE (suite)

On considère:

```
EXPLAIN ANALYSE SELECT * FROM ROLE,PEOPLE WHERE
                    ROLE.pid = PEOPLE.pid;
```

On obtient :

```
Hash Join (cost=312.07..822.95 rows=14535 width=37)
          (actual time=14.799..44.691 rows=14535 loops=1)
  Hash Cond: (role.pid = people.pid)
  -> Seq Scan on role (cost=0.00..238.35 rows=14535 width=20)
                      (actual time=0.019..7.570 rows=14535 loops=1)
  -> Hash (cost=175.92..175.92 rows=10892 width=17)
          (actual time=14.711..14.711 rows=10892 loops=1)
        -> Seq Scan on people (cost=0.00..175.92 rows=10892 width=17)
                              (actual time=0.015..5.944 rows=10892 loops=1)
```


EXPLAIN ANALYSE (suite)

```
Seq Scan on people (cost=0.00..175.92 rows=10892 width=17)
                   (actual time=0.015..5.944 rows=10892 loops=1)
```

- **Le nom de l'opérateur** ≡ nœud de l'arbre dans le plan de requête
- **Estimation du coût** (voir suite)
- **Coût réel** n'apparaît que si on a fait `EXPLAIN ANALYSE` (voir suite)

Estimation des coût

```
(cost=0.00..175.92 rows=10892 width=17)
```

- **Estimation du coût**. Unité : temps que met une lecture de bloc de 8ko (pour être indépendant du hardware). Le premier nombre est le temps estimé pour avoir le premier résultat. Le deuxième le temps estimé pour avoir l'ensemble.
- **Estimation du nombre de lignes** dans le résultat
- **Taille des lignes en octets**

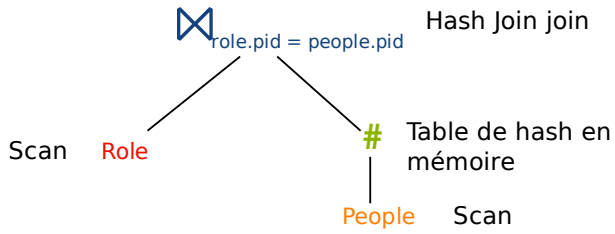
Coût réel

```
(actual time=0.015..5.944 rows=10892 loops=1)
```

- **Coût réel**. Unité : **ms**. Devrait être proportionnel à l'estimation si l'optimiseur ne s'est pas trompé. Le premier nombre est le temps pour avoir le premier résultat. Le deuxième le temps pour avoir l'ensemble.
- **Nombre de lignes** dans le résultat
- **looks=x** l'opérateur a été appelé x fois

Lecture du plan de requête

```
Hash Join (cost=…) (actual time=…)
  Hash Cond: (role.pid = people.pid)
  -> Seq Scan on role (cost=…) (actual time=…)
  -> Hash (cost=…) (actual time=…)
        -> Seq Scan on people (cost=…) (actual time=…)
```



Note: les projections n'apparaissent pas, on peut les voir avec `EXPLAIN ANALYSE VERBOSE`**.**

Plan

1 Rappels ✔
2 Stockage ✔
3 Indexation ✔
4 Optimisation des opérateurs ✔
5 Optimisation de requêtes ✔
6 Prise en main de Postgresql
  6.1 Base d'exemple ✔
  6.2 EXPLAIN ✔
  6.3 Exemples d'opérateurs
  6.4 Démo

Nœuds fréquements rencontrés lors d'un EXPLAIN ANALYSE

Les opérateurs sont déclinés selon les différents algorithmes (jointure, tris, ...)

- `Seq Scan`**: Scan séquentiel**
- `Nested loop`**: Jointure itérative page à page**
- `Merge sort join`**: Jointure par tri fusion**
- `Hash join`**: jointure par hashage (généralisation de la jointure sur index)**
- `Sort`**: Tri (le nœud indique l'algo de tri et la fonction de comparaison)**
- `Index scan`**: scan d'un index (précise l'index et la condition)**
- `Hash`**: génération d'une table de hash à la volée**
- `Bitmap Index scan/Bitmap heap scan`**: génération et utilisation d'un index bitmap à la volée (voire suite)**
- `Materialize`**: écriture de résultats intermédiaires sur le disque**

Retour sur les opératuers « *Bitmap* »

(Cours 5) si l'index est non-groupant, l'utilisation de l'index peut provoquer une séquence de chargement/déchargement de pages ruinant les performances

Solution en deux phases

- Scanner l'index et créer un bitmap de taille N où N est le nombre de pages de la relation. Pour chaque entrée d'index satisfaisant le résultat, mettre le bit correspondant à 1 (phase Bitmap Index Scan)
- Parcourir la relation dans l'ordre du disque, page à page. Si la page est à 1 dans le bitmap, on la charge, sinon on l'ignore. Une fois la page chargée il faut **réévaluer la condition** car on a oublié quels étaients les résultats de l'index (phase Bitmap Heap Scan)

Intéret: Un bitmap est petit (si la relation contient 10000 pages, le bitmap contient 10000 bits ou environs 1250 octets (soit moins d'une page).

Cela permet aussi de répondre efficacement aux requêtes booléenes complexes (i.e. autre qu AND). En effet on calcule un bitmap pour chaque sous-condition, et on fait les opérations entre bitmap bit à bit.

Plan

1 Rappels ✔
2 Stockage ✔
3 Indexation ✔
4 Optimisation des opérateurs ✔
5 Optimisation de requêtes ✔
6 Prise en main de Postgresql
- 6.1 Base d'exemple ✔
- 6.2 EXPLAIN ✔
- 6.3 Exemples d'opérateurs ✔
- 6.4 Démo