

# SGBD : BASES DE DONNÉES AVANCÉES [M3106C]

## TD N<sup>o</sup>2 - OPTIMISATION DE REQUÊTES SOUS POSTGRESQL

### OBJECTIFS

- Utilisation de EXPLAIN
- Notions de *tuning* des requêtes
- Méthodes d'accès

### CORRIGÉS

Exercice I :

#### Question 1.1.

```
# EXPLAIN SELECT dy_company, date(dy_timestamp) FROM diary;
                        QUERY PLAN
```

```
-----
Seq Scan on diary  (cost=0.00..22.50 rows=1000 width=12)
(1 ligne)
```

#### Question 1.2.

```
# select reltuples, relpages
   from pg_class where relname='diary';
 reltuples | relpages
-----+-----
```

```
      1000 |         10
(1 ligne)
```

```
# show cpu_tuple_cost;
cpu_tuple_cost
```

```
-----
      0.01
(1 ligne)
```

```
# show cpu_operator_cost;
cpu_operator_cost
```

```
-----
```

---

*Date:* 20 mars 2014.

Hocine ABIR - IUT Villetaneuse .

```
0.0025
(1 ligne)
```

```
# select 10+1000*0.01+1000*0.0025;
?column?
```

```
-----
22.5000
(1 ligne)
```

### Question 1.3.

```
# EXPLAIN SELECT dy_company, date(dy_timestamp)
      FROM diary WHERE dy_id>=900;
      QUERY PLAN
```

```
-----
Seq Scan on diary  (cost=0.00..23.33 rows=333 width=12)
  Filter: (dy_id >= 900)
(2 lignes)
```

```
# EXPLAIN SELECT dy_company, date(dy_timestamp)
      FROM diary WHERE dy_id>=10;
      QUERY PLAN
```

```
-----
Seq Scan on diary  (cost=0.00..23.33 rows=333 width=12)
  Filter: (dy_id >= 10)
(2 lignes)
```

Commentaire :

- Les estimations de ces deux requêtes sont identiques !
- La première a un faible taux de sélectivité ( $\geq 900$ ) mais l'accès séquentiel **Seq Scan** est préféré à l'accès par l'index **Index Scan** !
- le nombre de tuples estimé pour le résultat est identique : bizarre

Pour les deux plans, l'optimiseur a estimé le même nombre de tuple pour le résultat !! (333) . Ceci est dû au fait qu'il n'a pas de statistiques sur la table **diary** (et les autres).

Pour remédier à ce problème :

```
-----
# VACUUM ANALYZE diary;
VACUUM
```

### Question 1.4.

```
EXPLAIN SELECT dy_company, date(dy_timestamp)
FROM diary WHERE dy_id>=900;
```

QUERY PLAN

```
-----
Index Scan using diary_pkey on diary \
      (cost=0.00..5.60 rows=100 width=12)
  Index Cond: (dy_id >= 900)
(2 lignes)
```

```
abir=> EXPLAIN SELECT dy_company, date(dy_timestamp)
FROM diary WHERE dy_id>=10;
```

QUERY PLAN

```
-----
Seq Scan on diary (cost=0.00..24.98 rows=991 width=12)
  Filter: (dy_id >= 10)
(2 lignes)
```

- Le nombre de tuples estimés dans chaque requête est plus réaliste.
- L'accès par index Index Scan pour la sélectivite faible est honoré.

### Question 1.5.

```
# set enable_indexscan to off;
SET
```

```
# EXPLAIN SELECT dy_company, date(dy_timestamp)
FROM diary WHERE dy_id>=900;
```

QUERY PLAN

```
-----
Seq Scan on diary (cost=0.00..22.75 rows=100 width=12)
  Filter: (dy_id >= 900)
(2 lignes)
```

le temps a plus que triplé : l'index diary\_pkey est bénéfique.

### Question 1.6.

*"Donner pour chaque compagnie, les noms des produits et les quantités commandées (si ces dernières sont) supérieures à 60? "*

### Question 1.7.

```
# 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;

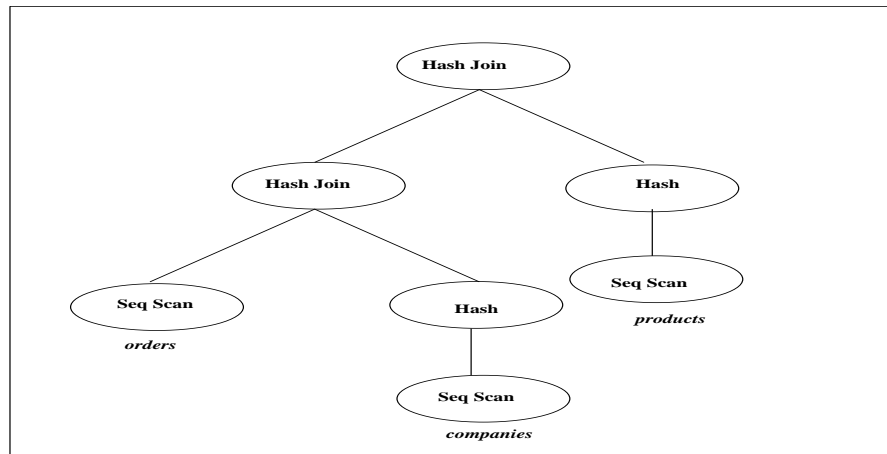
```

QUERY PLAN

```

-----
Hash Join (cost=4.38..36.16 rows=376 width=52)
  Hash Cond: (("outer".ord_product)::text = ("inner".pr_code)::text)
  -> Hash Join (cost=3.25..29.39 rows=376 width=39)
    Hash Cond: ("outer".ord_company = "inner".co_id)
    -> Seq Scan on orders (cost=0.00..20.50 rows=376 width=17)
        Filter: (ord_qty > 60)
    -> Hash (cost=3.00..3.00 rows=100 width=30)
        -> Seq Scan on companies (cost=0.00..3.00 rows=100 width=30)
  -> Hash (cost=1.10..1.10 rows=10 width=31)
    -> Seq Scan on products (cost=0.00..1.10 rows=10 width=31)
(10 lignes)

```



```

# set enable_hashjoin to off;
SET
# 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

```

-----
Merge Join (cost=65.94..72.08 rows=376 width=52)
  Merge Cond: ("outer".co_id = "inner".ord_company)
  -> Sort (cost=6.32..6.57 rows=100 width=30)
    Sort Key: companies.co_id
  -> Seq Scan on companies (cost=0.00..3.00 rows=100 width=30)

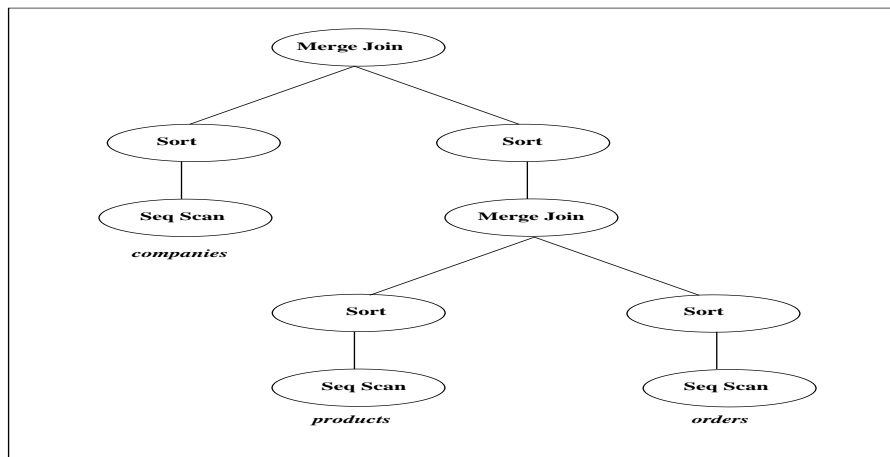
```

```

-> Sort (cost=59.62..60.56 rows=376 width=30)
    Sort Key: orders.ord_company
  -> Merge Join (cost=37.85..43.54 rows=376 width=30)
      Merge Cond: ("outer"."?column3?" = "inner"."?column4?")
    -> Sort (cost=1.27..1.29 rows=10 width=31)
        Sort Key: (products.pr_code)::text
    -> Seq Scan on products (cost=0.00..1.10 rows=10 width=31)
  -> Sort (cost=36.58..37.52 rows=376 width=17)
      Sort Key: (orders.ord_product)::text
    -> Seq Scan on orders (cost=0.00..20.50 rows=376 width=17)
        Filter: (ord_qty > 60)

```

(16 lignes)



```

# set enable_mergejoin to off;
SET
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;

```

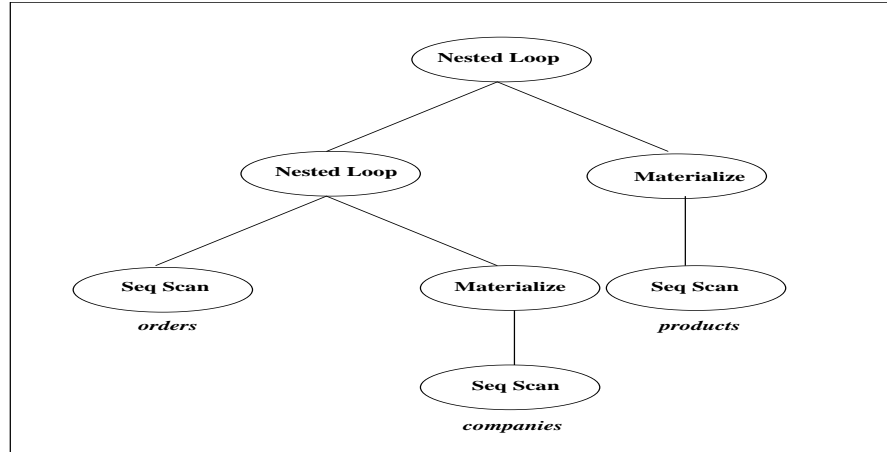
QUERY PLAN

```

-----
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=31)  
 -> Materialize (cost=1.11..1.21 rows=10 width=31)  
 -> Seq Scan on products (cost=0.00..1.10 rows=10 width=31)  
 (10 lignes)



### Question 1.8.

```
# explain SELECT co_name,pr_desc,ord_qty
FROM orders join products on (pr_code=ord_product ) ,
companies
where co_id=ord_company
AND ord_qty>60;
```

#### QUERY PLAN

```
-----
Nested Loop (cost=4.21..955.31 rows=376 width=52)
  Join Filter: ("inner".co_id = "outer".ord_company)
  -> Nested Loop (cost=1.11..106.21 rows=376 width=30)
    Join Filter: (("inner".pr_code)::text = ("outer".ord_product)::text)
    -> Seq Scan on orders (cost=0.00..20.50 rows=376 width=17)
      Filter: (ord_qty > 60)
    -> Materialize (cost=1.11..1.21 rows=10 width=31)
      -> Seq Scan on products (cost=0.00..1.10 rows=10 width=31)
  -> Materialize (cost=3.10..4.10 rows=100 width=30)
    -> Seq Scan on companies (cost=0.00..3.00 rows=100 width=30)
(10 rows)
```

## Exercice II :

### Question 2.1.

```
select fou_nom, pie_nom from
  fournisseur join piece on fou_id=fou_id_fk;
```

**Question 2.2.**

```
select fou_nom, pie_nom
  from piece join fournisseur on fou_id=fou_id_fk;
```

**Question 2.3.**

```
select fou_nom, pie_nom
  from fournisseur join piece on fou_id=fou_id_fk;
fou_nom | pie_nom
-----+-----
Peugeot | Huile
Renault | Pneu
(2 rows)
```

**Question 2.4.**

```
select fou_nom, pie_nom
  from piece join fournisseur on fou_id=fou_id_fk;
fou_nom | pie_nom
-----+-----
Renault | Pneu
Peugeot | Huile
(2 rows)
```

**Question 2.5.**

C1 : rows=1000 <---> C2 : 2 rows

**Question 2.6.**

```
Select * from piece;
 pie_id | pie_nom | pie_stock | pie_prix | fou_id_fk
-----+-----+-----+-----+-----
      1 | Pneu    |          11 |         59 |         20
      2 | Huile   |          20 |         18 |         10
      3 | Pneu Michelin |          10 |         90 |         20
      4 | Pneu Michelin |          10 |         90 |         20
      5 | Pneu Keber |          10 |         90 |         10
      6 | Pneu Keber |          10 |         90 |         10
(6 rows)
```

**Question 2.7.**

```
# select reltuples from pg_class
  where relname='piece';
 reltuples
-----
```

2

(1 row)

Les statistiques ne sont pas mises à jour !

```
# vacuum piece;
VACUUM
```

```
# select reltuples from pg_class
  where relname='piece';
 reltuples
-----
```

6

(1 row)