ABD - TP 06

Auteurs

- Anne-Sophie Saint-Omer
- Thomas Bernard

Description

L'objectif de ce TP est d'analyser les plans d'exécutions de requêtes SQL dans Postgresql.

Analyse

Explication des premières instructions :

1. `select * from T`

```
explain select * from T ;

Seq Scan on t  (cost=0.00..3.00 rows=100 width=102)
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
0	**3**	100	102

Le `SELECT *` retourne toutes les lignes présentes dans la table. Le nombre de lignes de la table t étant de 100, rows = 100. Les champs a et b sont des VARCHARS respectivement de taille 3 et 97. En tout on a donc une taille de 100 octets, la largeur moyenne estimée étant de 102 octets.

2. `select * from T where a = '4'`

```
explain select * from T where a = '4' ;

Seq Scan on t  (cost=0.00..3.25 rows=1 width=102)
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
0	**3.25**	1	102

Le `SELECT *` retourne une seule ligne. On cherche en effet dans le champ a, qui correspond à la clé primaire (elle est donc unique), la valeur 4.
Les champs a et b sont des VARCHARS respectivement de taille 3 et 97. En tout, on a donc une taille de 100 octets, la

largeur moyenne estimée étant de 102 octets.
Le coût est plus élevé que pour la requête précédente à cause de la condition `WHERE`.

3. `select T.A from T where T.A > '50'`

```
explain select T.A from T where T.A > '50';

Seq Scan on t  (cost=0.00..3.25 rows=54 width=4)
Filter: (a > '50'::bpchar)
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
0	**3.25**	54	4

Cette requête retourne 54 lignes, de l'ID 51 à 99 mais également les ID 6 à 9.
Le champs a est un VARCHAR de taille 3, la largeur estimée est de 4 octets.
Même chose que la requête précédente : le coût est plus élevé que pour la première requête à cause de la condition `WHERE`.

4. `select T.A, T.B from T where T.A > '50'`

```
explain  select T.A, T.B from T where T.A > '50';

Seq Scan on t  (cost=0.00..3.25 rows=54 width=102)
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
0	**3.25**	54	102

Cette requête retourne 54 lignes, de l'ID 51 à 99 mais également les ID 6 à 9.
Le champs a est un VARCHAR de taille 3, la largeur estimée est de 4.
Même chose que précédemment : VARCHAR(3) + VARCHAR(97)

Jointure :

Examen du plan d'execution du calcul de la jointure entre T et TT :

```
select tt.a, t.a, tt.b
from t join tt on t.a = tt.t ;

Hash Join  (cost=4.25..16.05 rows=267 width=103)
        Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=103)
                ->  Hash  (cost=3.00..3.00 rows=100 width=4)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=4)
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
4.25	**16.05**	267	103

Le coût de lancement n'est plus 0 comme dans les requêtes précédentes, la jointure augmente ce coût.
Chaque SeqScan correspond à une table : il y a 300 lignes dans tt et 100 lignes dans t.

On visualise deux sous-requêtes executées sur chaque table :
`Seq Scan on tt (cost=0.00..8.00 rows=300 width=103)` Récupération des enregistrements de la table tt.
`Seq Scan on t (cost=0.00..3.00 rows=100 width=4)` Récupération des enregistrements de la table t.

Il n'y a pas de jointure, on exécute une requête sur une table dans ces lignes, le coût de lancement est donc estimé est de 0.
La jointure n'a lieu qu'à ce moment : `Hash Join (cost=4.25..16.05 rows=267 width=103)`

Comparaison avec les plans d'exécutions des requêtes suivantes :

1. **`select tt.a, tt.t, tt.b from t join tt on t.a = tt.t`**

```
explain select tt.a, tt.t, tt.b from t join tt on t.a = tt.t ;

Hash Join  (cost=4.25..16.05 rows=267 width=103)
        Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=103)
                ->  Hash  (cost=3.00..3.00 rows=100 width=4)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=4)
```

Jointure sur un élément différent mais apparement identique.
Les coûts estimés sont les mêmes qu'à la requête précédente.

```
explain analyze select tt.a, tt.t, tt.b from t join tt on t.a = tt.t ;

Hash Join  (cost=4.25..16.05 rows=267 width=103) (actual time=0.381..1.474 rows=267 loops=1)
        Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=103) (actual time=0.022..0.412 rows=300
                ->  Hash  (cost=3.00..3.00 rows=100 width=4) (actual time=0.292..0.292 rows=100 loops=1)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=4) (actual time=0.022..0.152 rows
Total runtime: 1.827 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
4.25	**16.05**	267	103

Le mot clé `analyze` retourne `actual time` qui correspond au temps réel d'exécution pour chaque noeud du plan.
`rows` représente le nombre de lignes pour ce noeud.
`loops` correspond au nombre total d'exécution du noeud. Dans cette requête, chaque noeud n'a été executé qu'une seule fois.
Chaque noeud récupère les valeurs des noeud précédents, on obtient donc le temps d'exécution total de la requête en analysant la première ligne. Cette première ligne est différente du temps total précisé à la dernière ligne. Le "total runtime" de la dernière ligne inclut le temps de lancement et d'arrêt de l'exécuteur.

2. **`select tt.a, tt.t, tt.b from tt where tt.t in (select a from t)`**

```
explain select tt.a, tt.t, tt.b from tt where tt.t in (select a from t) ;

Hash Semi Join  (cost=4.25..16.01 rows=267 width=103)
        Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=103)
                ->  Hash  (cost=3.00..3.00 rows=100 width=4)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=4
```

Les coûts sont similaires à la requête précédente malgré la sous-requête.

On visualise deux sous-requêtes executées sur chaque table :
`Seq Scan on tt (cost=0.00..8.00 rows=300 width=103)` Récupération des enregistrements de la table tt.
`Seq Scan on t (cost=0.00..3.00 rows=100 width=4)` Récupération des enregistrements de la table t.

Il n'y a pas de jointure, on exécute une requête sur une table dans ces lignes, le coût de lancement est donc estimé est de 0.
La jointure n'a lieu qu'à ce moment :

```
explain analyze select tt.a, tt.t, tt.b from tt where tt.t in (select a from t) ;

Hash Semi Join  (cost=4.25..16.01 rows=267 width=103) (actual time=0.385..1.463 rows=267 loops=1)
        Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=103) (actual time=0.023..0.402 rows=300
                ->  Hash  (cost=3.00..3.00 rows=100 width=4) (actual time=0.289..0.289 rows=100 loops=1)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=4) (actual time=0.023..0.150 rows
Total runtime: 1.806 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
4.25	**16.01**	267	103

3. `select tt.a, tt.t, tt.b from tt where tt.t is not null`

```
explain select tt.a, tt.t, tt.b from tt where tt.t is not null ;

Seq Scan on tt  (cost=0.00..8.00 rows=267 width=103)
        Filter: (t IS NOT NULL)
```

Le plan d'exécution de cette requête est différent des précédentes. Un filtre est appliqué.
Le coût est différent pour le même résultat. Ce coût semble avantageux.

```
explain analyze select tt.a, tt.t, tt.b from tt where tt.t is not null ;

Seq Scan on tt  (cost=0.00..8.00 rows=267 width=103) (actual time=0.029..0.413 rows=267 loops=1)
        Filter: (t IS NOT NULL)
Total runtime: 0.744 ms
```

On gagne effectivement plus d'1 ms. Cette requête est donc plus efficace que les autres.

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
0.0	**8.00**	267	103

Calcul d'une jointure entre les colonnes T.A et T.B :

1. **`select T.A, T.B from T join TT on T.A = TT.T`**

```
explain select T.A, T.B from T join TT on T.A = TT.T ;

Hash Join  (cost=4.25..16.05 rows=267 width=102)
        Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4)
                ->  Hash  (cost=3.00..3.00 rows=100 width=102)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=102)
```

```
explain analyze select T.A, T.B from T join TT on T.A = TT.T ;

Hash Join  (cost=4.25..16.05 rows=267 width=102) (actual time=0.233..0.833 rows=267 loops=1)
        Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4) (actual time=0.016..0.237 rows=300 lo
                ->  Hash  (cost=3.00..3.00 rows=100 width=102) (actual time=0.174..0.174 rows=100 loops=1)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=102) (actual time=0.012..0.081 ro
Total runtime: 1.008 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
4.25	**16.05**	267	103

2. **`select distinct T.A, T.B from T join TT on T.A = TT.T`**

```
explain select distinct T.A, T.B from T join TT on T.A = TT.T ;

HashAggregate  (cost=17.38..18.38 rows=100 width=102)
        ->  Hash Join  (cost=4.25..16.05 rows=267 width=102)
                Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4)
                ->  Hash  (cost=3.00..3.00 rows=100 width=102)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=102)
```

Nous pouvons en déduire que le distinct impacte fortement le coût estimé du lancement.

```
explain analyze  select distinct T.A, T.B from T join TT on T.A = TT.T ;

HashAggregate  (cost=17.38..18.38 rows=100 width=102) (actual time=2.055..2.156 rows=89 loops=1)
        ->  Hash Join  (cost=4.25..16.05 rows=267 width=102) (actual time=0.372..1.521 rows=267 loops=1)
                Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4) (actual time=0.028..0.416 rows=300 lo
                ->  Hash  (cost=3.00..3.00 rows=100 width=102) (actual time=0.309..0.309 rows=100 loops=1)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=102) (actual time=0.019..0.145 ro
Total runtime: 2.392 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
17.38	**18.38**	100	102

3. `select T.A, T.B from T where T.A in (select TT.T from TT)`

```
explain select T.A, T.B from T where T.A in (select TT.T from TT)

Hash Semi Join  (cost=11.75..16.11 rows=89 width=102)
        Hash Cond: (t.a = tt.t)
        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=102)
        ->  Hash  (cost=8.00..8.00 rows=300 width=4)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4)
```

La sous-requête impacte le coût estimé de lancement, il est cependant inférieur au coût que peut représenter le distinct.

```
explain analyze select T.A, T.B from T where T.A in (select TT.T from TT)

Hash Semi Join  (cost=11.75..16.11 rows=89 width=102) (actual time=0.990..1.354 rows=89 loops=1)
        Hash Cond: (t.a = tt.t)
        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=102) (actual time=0.023..0.146 rows=100 loops=1)
        ->  Hash  (cost=8.00..8.00 rows=300 width=4) (actual time=0.901..0.901 rows=267 loops=1)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4) (actual time=0.026..0.458 rows=300 lo
Total runtime: 1.506 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
11.75	**16.11**	89	102

4. `select T.A, T.B from T where 3 = (select count(*) from TT where TT.T = T.A)`

```
explain select T.A, T.B from T where 3 = (select count(*) from TT where TT.T = T.A)

Seq Scan on t  (cost=0.00..880.25 rows=1 width=102)
        Filter: (3 = (SubPlan 1))
        SubPlan 1
                ->  Aggregate  (cost=8.76..8.77 rows=1 width=0)
                        ->  Seq Scan on tt  (cost=0.00..8.75 rows=3 width=0)
                                Filter: (t = $0)
```

Le coût total estimé est de 880.25 !
Le count(*) en est responsable.

```
explain analyze select T.A, T.B from T where 3 = (select count(*) from TT where TT.T = T.A)

Seq Scan on t  (cost=0.00..880.25 rows=1 width=102) (actual time=0.231..9.358 rows=89 loops=1)
        Filter: (3 = (SubPlan 1))
        SubPlan 1
                ->  Aggregate  (cost=8.76..8.77 rows=1 width=0) (actual time=0.088..0.089 rows=1 loops=100)
                        ->  Seq Scan on tt  (cost=0.00..8.75 rows=3 width=0) (actual time=0.044..0.080 rows=
```

```
                                Filter: (t = $0)
Total runtime: 9.638 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
0	**880.25**	1	102

Calcul d'une jointure entre les colonnes T.A et T.B avec index sur TT(T) :

1. `select T.A, T.B from T join TT on T.A = TT.T`

```
explain select T.A, T.B from T join TT on T.A = TT.T ;

Hash Join  (cost=4.25..16.05 rows=267 width=102)
        Hash Cond: (tt.t = t.a)
        ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4)
        ->  Hash  (cost=3.00..3.00 rows=100 width=102)
                ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=102)
```

L'index ne change rien au coût.

```
explain analyze select T.A, T.B from T join TT on T.A = TT.T ;

Hash Join  (cost=4.25..16.05 rows=267 width=102) (actual time=0.428..1.521 rows=267 loops=1)
        Hash Cond: (tt.t = t.a)
        ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4) (actual time=0.028..0.411 rows=300 loops=1)
        ->  Hash  (cost=3.00..3.00 rows=100 width=102) (actual time=0.341..0.341 rows=100 loops=1)
                ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=102) (actual time=0.020..0.167 rows=100 l
Total runtime: 1.854 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
4.25	**16.05**	267	103

Le temps d'exécution total est beaucoup plus long. 1.008 ms sans l'index, 1.854 ms avec l'index.

2. `select distinct T.A, T.B from T join TT on T.A = TT.T`

```
explain select distinct T.A, T.B from T join TT on T.A = TT.T ;

HashAggregate  (cost=17.38..18.38 rows=100 width=102)
        ->  Hash Join  (cost=4.25..16.05 rows=267 width=102)
                Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4)
                ->  Hash  (cost=3.00..3.00 rows=100 width=102)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=102)
```

L'index ne change rien au coût.

```
explain analyze  select distinct T.A, T.B from T join TT on T.A = TT.T ;

HashAggregate  (cost=17.38..18.38 rows=100 width=102) (actual time=2.086..2.188 rows=89 loops=1)
        ->  Hash Join  (cost=4.25..16.05 rows=267 width=102) (actual time=0.387..1.544 rows=267 loops=1)
                Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4) (actual time=0.024..0.446 rows=300 lo
                ->  Hash  (cost=3.00..3.00 rows=100 width=102) (actual time=0.331..0.331 rows=100 loops=1)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=102) (actual time=0.019..0.159 ro
Total runtime: 2.417 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
17.38	**18.38**	100	102

L'index augmente le le temps d'exécution total d'environ 0,100 ms.

3. `select T.A, T.B from T where T.A in (select TT.T from TT)`

```
explain select T.A, T.B from T where T.A in (select TT.T from TT);

Hash Semi Join  (cost=11.75..16.11 rows=89 width=102)
        Hash Cond: (t.a = tt.t)
        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=102)
        ->  Hash  (cost=8.00..8.00 rows=300 width=4)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4)
```

```
explain analyze select T.A, T.B from T where T.A in (select TT.T from TT)

Hash Semi Join  (cost=11.75..16.11 rows=89 width=102) (actual time=0.934..1.296 rows=89 loops=1)
        Hash Cond: (t.a = tt.t)
        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=102) (actual time=0.022..0.144 rows=100 loops=1)
        ->  Hash  (cost=8.00..8.00 rows=300 width=4) (actual time=0.843..0.843 rows=267 loops=1)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4) (actual time=0.022..0.432 rows=300 lo
Total runtime: 1.447 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
11.75	**16.11**	89	102

On gagne 0,100 sur le temps d'exécution total.

4. `select T.A, T.B from T where 3 = (select count(*) from TT where TT.T = T.A)`

```
explain select T.A, T.B from T where 3 = (select count(*) from TT where TT.T = T.A)

Seq Scan on t  (cost=0.00..880.25 rows=1 width=102)
        Filter: (3 = (SubPlan 1))
        SubPlan 1
                ->  Aggregate  (cost=8.76..8.77 rows=1 width=0)
                        ->  Seq Scan on tt  (cost=0.00..8.75 rows=3 width=0)
                                Filter: (t = $0)
```

```
explain analyze select T.A, T.B from T where 3 = (select count(*) from TT where TT.T = T.A)

Seq Scan on t  (cost=0.00..880.25 rows=1 width=102) (actual time=0.242..8.346 rows=89 loops=1)
        Filter: (3 = (SubPlan 1))
        SubPlan 1
                ->  Aggregate  (cost=8.76..8.77 rows=1 width=0) (actual time=0.078..0.079 rows=1 loops=100)
                        ->  Seq Scan on tt  (cost=0.00..8.75 rows=3 width=0) (actual time=0.037..0.071 rows=
                                Filter: (t = $0)
Total runtime: 8.559 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
0	**880.25**	1	102

On gagne 1 ms grâce à l'index.

En conclusion l'index apporte un gain de temps dans des requêtes contenant des sous-requêtes mais fait perdre du temps lors d'une simple jointure entre deux tables.

Comparaisons de requêtes avec GROUP BY :

Comparaisons des deux premières requêtes :

1. `select a, count(*) from t group by a`

```
explain select a, count(*) from t group by a ;

HashAggregate  (cost=3.50..4.75 rows=100 width=4)
        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=4)
```

```
explain analyze select a, count(*) from t group by a ;

HashAggregate  (cost=3.50..4.75 rows=100 width=4) (actual time=0.328..0.442 rows=100 loops=1)
        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=4) (actual time=0.021..0.145 rows=100 loops=1)
Total runtime: 0.743 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
3.5	**4.75**	100	4

2. `select b, count(*) from t group by b`

```
explain select b, count(*) from t group by b;

HashAggregate  (cost=3.50..4.75 rows=100 width=98)
        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=98)
```

```
explain analyze select b, count(*) from t group by b;

HashAggregate  (cost=3.50..4.75 rows=100 width=98) (actual time=0.363..0.478 rows=100 loops=1)
        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=98) (actual time=0.022..0.148 rows=100 loops=1)
Total runtime: 0.756 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
3.5	**4.75**	100	4

Le temps d'éxécution total, le coût estimé du lancement et le coût total estimé des deux requêtes sotn similaires.

Comparaisons des deux dernières requêtes :

1. `select t.a, count(*) from t join tt on tt.t = t.a group by t.a`

```
explain select t.a, count(*) from t join tt on tt.t = t.a group by t.a ;

HashAggregate  (cost=17.38..18.63 rows=100 width=4)
        ->  Hash Join  (cost=4.25..16.05 rows=267 width=4)
                Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4)
                ->  Hash  (cost=3.00..3.00 rows=100 width=4)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=4)
```

```
explain analyze select t.a, count(*) from t join tt on tt.t = t.a group by t.a ;

HashAggregate  (cost=17.38..18.63 rows=100 width=4) (actual time=1.890..1.993 rows=89 loops=1)
        ->  Hash Join  (cost=4.25..16.05 rows=267 width=4) (actual time=0.358..1.484 rows=267 loops=1)
                Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4) (actual time=0.018..0.409 rows=300 lo
                ->  Hash  (cost=3.00..3.00 rows=100 width=4) (actual time=0.297..0.297 rows=100 loops=1)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=4) (actual time=0.020..0.148 rows
Total runtime: 2.272 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
17.38	**18.63**	100	4

2. `select t.b, count(*) from t join tt on tt.t = t.a group by t.b`

```
explain select t.b, count(*) from t join tt on tt.t = t.a group by t.b ;

HashAggregate  (cost=17.38..18.63 rows=100 width=98)
        ->  Hash Join  (cost=4.25..16.05 rows=267 width=98)
                Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4)
                ->  Hash  (cost=3.00..3.00 rows=100 width=102)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=102
```

```
explain analyze select t.b, count(*) from t join tt on tt.t = t.a group by t.b ;

HashAggregate  (cost=17.38..18.63 rows=100 width=98) (actual time=2.105..2.207 rows=89 loops=1)
        ->  Hash Join  (cost=4.25..16.05 rows=267 width=98) (actual time=0.399..1.526 rows=267 loops=1)
                Hash Cond: (tt.t = t.a)
                ->  Seq Scan on tt  (cost=0.00..8.00 rows=300 width=4) (actual time=0.026..0.422 rows=300 lo
                ->  Hash  (cost=3.00..3.00 rows=100 width=102) (actual time=0.334..0.334 rows=100 loops=1)
                        ->  Seq Scan on t  (cost=0.00..3.00 rows=100 width=102) (actual time=0.018..0.143 ro
Total runtime: 2.474 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
17.38	**18.63**	100	4

Le temps d'éxécution total, le coût estimé du lancement et le coût total estimé des deux requêtes sotn similaires.

Comparaisons de requêtes avec UNION :

1. `select * from t where a='5 ' or a='6 '`

```
explain select * from t where a='5 ' or a='6 ' ;

Seq Scan on t  (cost=0.00..3.50 rows=2 width=102)
        Filter: ((a = '5 '::bpchar) OR (a = '6 '::bpchar))
```

```
explain analyze select * from t where a='5 ' or a='6 ' ;

Seq Scan on t  (cost=0.00..3.50 rows=2 width=102) (actual time=0.057..0.106 rows=2 loops=1)
        Filter: ((a = '5 '::bpchar) OR (a = '6 '::bpchar))
Total runtime: 0.169 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
0	**3.50**	2	102

2. `select * from t where a='5 ' union select * from t where a='6 '`

```
explain select * from t where a='5 '
union
select * from t where a='6 '

Unique  (cost=6.53..6.54 rows=2 width=102)
        ->  Sort  (cost=6.53..6.53 rows=2 width=102)
                Sort Key: tp6.t.a, tp6.t.b
                ->  Append  (cost=0.00..6.52 rows=2 width=102)
                        ->  Seq Scan on t  (cost=0.00..3.25 rows=1 width=102)
                                Filter: (a = '5 '::bpchar)
                        ->  Seq Scan on t  (cost=0.00..3.25 rows=1 width=102)
                                Filter: (a = '6 '::bpchar)
```

```
explain analyze select * from t where a='5 '
union
select * from t where a='6 '

Unique  (cost=6.53..6.54 rows=2 width=102) (actual time=0.234..0.242 rows=2 loops=1)
        ->  Sort  (cost=6.53..6.53 rows=2 width=102) (actual time=0.231..0.233 rows=2 loops=1)
                Sort Key: tp6.t.a, tp6.t.b
                Sort Method:  quicksort  Memory: 25kB
                ->  Append  (cost=0.00..6.52 rows=2 width=102) (actual time=0.048..0.124 rows=2 loops=1)
                        ->  Seq Scan on t  (cost=0.00..3.25 rows=1 width=102) (actual time=0.042..0.081 rows
                                Filter: (a = '5 '::bpchar)
                        ->  Seq Scan on t  (cost=0.00..3.25 rows=1 width=102) (actual time=0.013..0.032 rows
                                Filter: (a = '6 '::bpchar)
Total runtime: 0.312 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
6.53	**6.54**	2	102

3. `select * from t where a='5 ' union all select * from t where a='6 '`

```
explain select * from t where a='5 '
union all
select * from t where a='6 '

Append  (cost=0.00..6.52 rows=2 width=102)
        ->  Seq Scan on t  (cost=0.00..3.25 rows=1 width=102)
                Filter: (a = '5 '::bpchar)
        ->  Seq Scan on t  (cost=0.00..3.25 rows=1 width=102)
                Filter: (a = '6 '::bpchar)
```

```
explain analyze select * from t where a='5 '
union all
select * from t where a='6 '

Append  (cost=0.00..6.52 rows=2 width=102) (actual time=0.044..0.110 rows=2 loops=1)
        ->  Seq Scan on t  (cost=0.00..3.25 rows=1 width=102) (actual time=0.038..0.069 rows=1 loops=1)
                Filter: (a = '5 '::bpchar)
        ->  Seq Scan on t  (cost=0.00..3.25 rows=1 width=102) (actual time=0.011..0.030 rows=1 loops=1)
                Filter: (a = '6 '::bpchar)
Total runtime: 0.165 ms
```

Coût estimé du lancement	Coût total estimé	Nombre de lignes estimé	Largeur moyenne estimée
0	**6.52**	2	102

Ces trois requêtes ont un temps d'exécution total très faibles. Le mot clé union fait perdre 0,200 ms au temps d'exécution. Les mots clés union all permette un temps d'exécution égal voir inférieur à une requête similaire sans les mots clés “union” ou “union all”