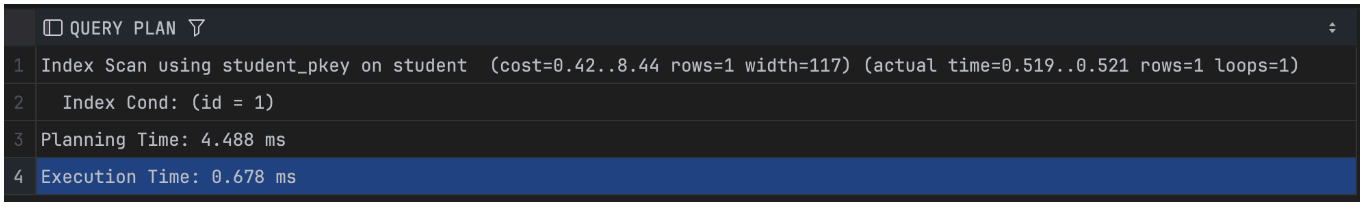
1 Recup les infos d'un etudiant

Avec index

```sql
EXPLAIN ANALYSE
select * from student where id = 1;
```



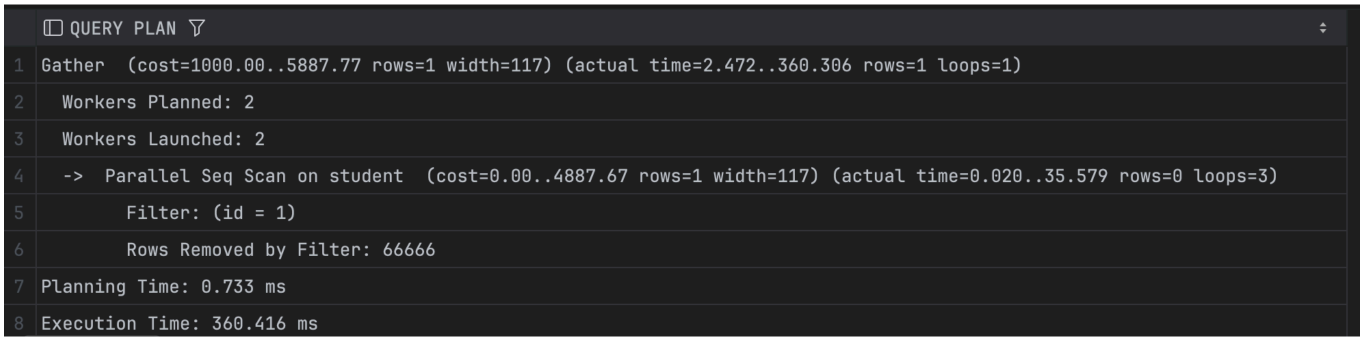
```
QUERY PLAN
Index Scan using student_pkey on student  (cost=0.42..8.44 rows=1 width=117) (actual time=0.519..0.521 rows=1 loops=1)
  Index Cond: (id = 1)
Planning Time: 4.488 ms
Execution Time: 0.678 ms
```

PostgreSQL utilise l'index student_pkey (clé primaire) pour localiser directement l'étudiant avec id = 1. Le temps d'exécution est de 0.678 ms, ce qui est rapide. L'index évite un scan séquentiel de la table.

Sans index

```sql
set enable_indexscan = off;
set enable_bitmapscan = off;

EXPLAIN ANALYSE
select * from student where id = 1;
```



```
QUERY PLAN
Gather  (cost=1000.00..5887.77 rows=1 width=117) (actual time=2.472..360.306 rows=1 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on student  (cost=0.00..4887.67 rows=1 width=117) (actual time=0.020..35.579 rows=0 loops=3)
        Filter: (id = 1)
        Rows Removed by Filter: 66666
Planning Time: 0.733 ms
Execution Time: 360.416 ms
```

Sans index :

PostgreSQL effectue un scan séquentiel parallèle de la table student. 66 666 lignes sont scannées avant de trouver la bonne. Temps d'exécution : 360.416 ms.

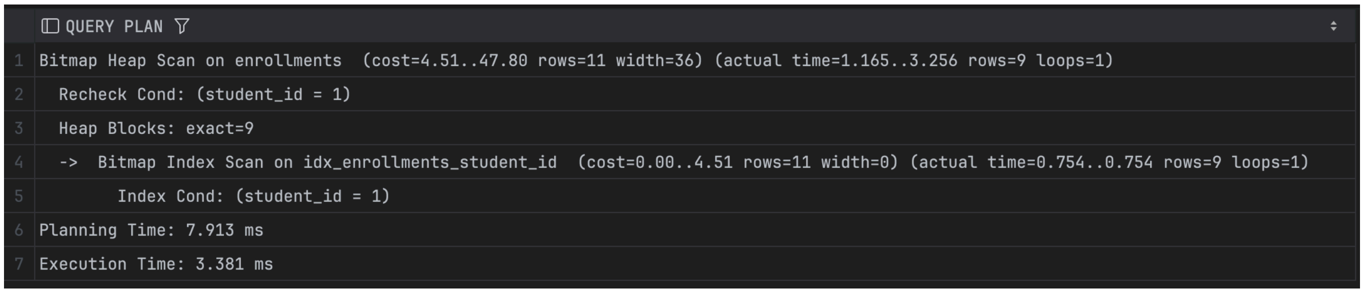
Gain de performance :

360.416 / 0.678 = 531x plus rapide avec l'index.

2 recup toutes les inscriptions d'un etudiant

Avec index

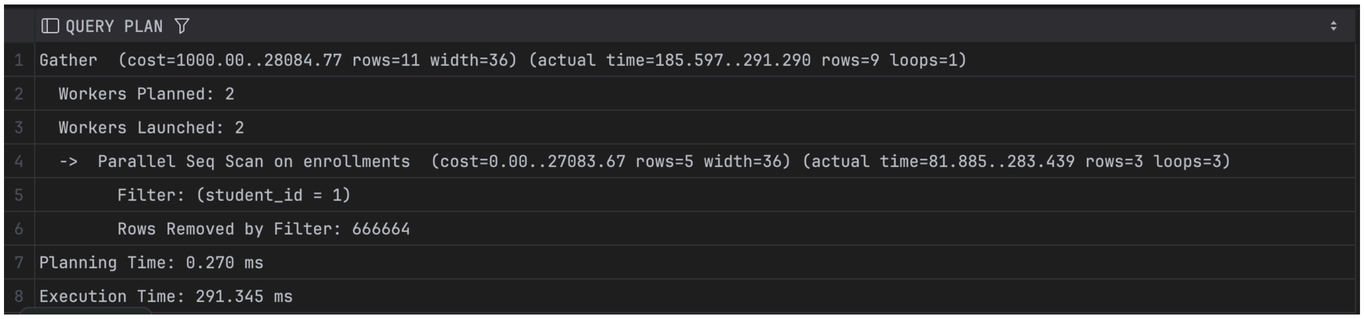
```sql
EXPLAIN ANALYSE
select * from enrollments where student_id = 1;
```



```
Bitmap Heap Scan on enrollments  (cost=4.51..47.80 rows=11 width=36) (actual time=1.165..3.256 rows=9 loops=1)
  Recheck Cond: (student_id = 1)
  Heap Blocks: exact=9
  ->  Bitmap Index Scan on idx_enrollments_student_id  (cost=0.00..4.51 rows=11 width=0) (actual time=0.754..0.754 rows=9 loops=1)
        Index Cond: (student_id = 1)
Planning Time: 7.913 ms
Execution Time: 3.381 ms
```

PostgreSQL utilise l'index idx_enrollments_student_id via un Bitmap Index Scan pour localiser les inscriptions de l'étudiant. Un Bitmap Heap Scan récupère ensuite les 9 lignes correspondantes depuis la table. Temps d'exécution : 3.381 ms.

Sans index



```
Gather  (cost=1000.00..28084.77 rows=11 width=36) (actual time=185.597..291.290 rows=9 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on enrollments  (cost=0.00..27083.67 rows=5 width=36) (actual time=81.885..283.439 rows=3 loops=3)
        Filter: (student_id = 1)
        Rows Removed by Filter: 666664
Planning Time: 0.270 ms
Execution Time: 291.345 ms
```

PostgreSQL effectue un scan séquentiel parallèle de la table enrollments. 666 664 lignes sont scannées avant de trouver les 9 lignes correspondantes. Temps d'exécution : 291.345 ms.

Gain de performance

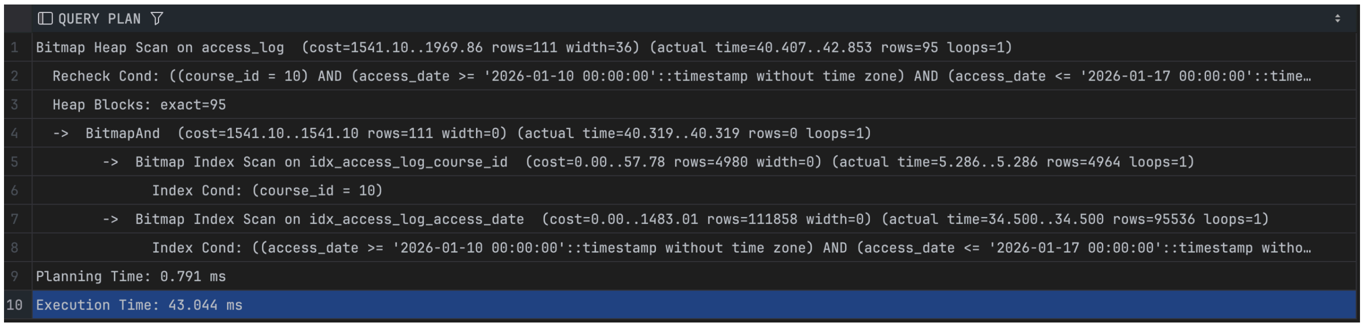
291.345 / 3.381 = 86x plus rapide avec l'index.

3 Recup tous les accès a un cours sur une periode de 7 jours

Avec index

```sql
EXPLAIN ANALYSE
select * from access_log
```

```sql
where course_id = 10
  and access_date between '2026-01-10' and '2026-01-17';
```



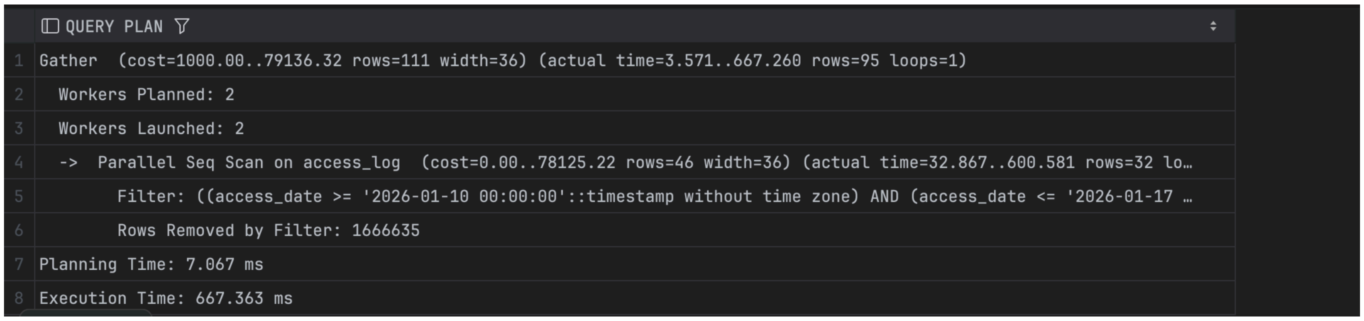
```
QUERY PLAN
Bitmap Heap Scan on access_log  (cost=1541.10..1969.86 rows=111 width=36) (actual time=40.407..42.853 rows=95 loops=1)
  Recheck Cond: ((course_id = 10) AND (access_date >= '2026-01-10 00:00:00'::timestamp without time zone) AND (access_date <= '2026-01-17 00:00:00'::time…
  Heap Blocks: exact=95
  ->  BitmapAnd  (cost=1541.10..1541.10 rows=111 width=0) (actual time=40.319..40.319 rows=0 loops=1)
        ->  Bitmap Index Scan on idx_access_log_course_id  (cost=0.00..57.78 rows=4980 width=0) (actual time=5.286..5.286 rows=4964 loops=1)
              Index Cond: (course_id = 10)
        ->  Bitmap Index Scan on idx_access_log_access_date  (cost=0.00..1483.01 rows=111858 width=0) (actual time=34.500..34.500 rows=95536 loops=1)
              Index Cond: ((access_date >= '2026-01-10 00:00:00'::timestamp without time zone) AND (access_date <= '2026-01-17 00:00:00'::timestamp witho…
Planning Time: 0.791 ms
Execution Time: 43.044 ms
```

PostgreSQL combine deux index via un BitmapAnd : idx_access_log_course_id pour filtrer course_id = 10 et idx_access_log_access_date pour la période. Un Bitmap Heap Scan récupère ensuite les 95 lignes correspondantes. Temps d'exécution : 43.044 ms.

Sans index

```sql
set enable_indexscan = off;
set enable_bitmapscan = off;

EXPLAIN ANALYSE
select * from access_log
where course_id = 10
  and access_date between '2026-01-10' and '2026-01-17';
```



```
QUERY PLAN
Gather  (cost=1000.00..79136.32 rows=111 width=36) (actual time=3.571..667.260 rows=95 loops=1)
  Workers Planned: 2
  Workers Launched: 2
  ->  Parallel Seq Scan on access_log  (cost=0.00..78125.22 rows=46 width=36) (actual time=32.867..600.581 rows=32 lo…
        Filter: ((access_date >= '2026-01-10 00:00:00'::timestamp without time zone) AND (access_date <= '2026-01-17 …
        Rows Removed by Filter: 1666635
Planning Time: 7.067 ms
Execution Time: 667.363 ms
```

PostgreSQL effectue un scan séquentiel parallèle de la table access_log. 1 666 635 lignes sont scannées avant de trouver les 95 lignes correspondantes à la période. Temps d'exécution : 667.363 ms.

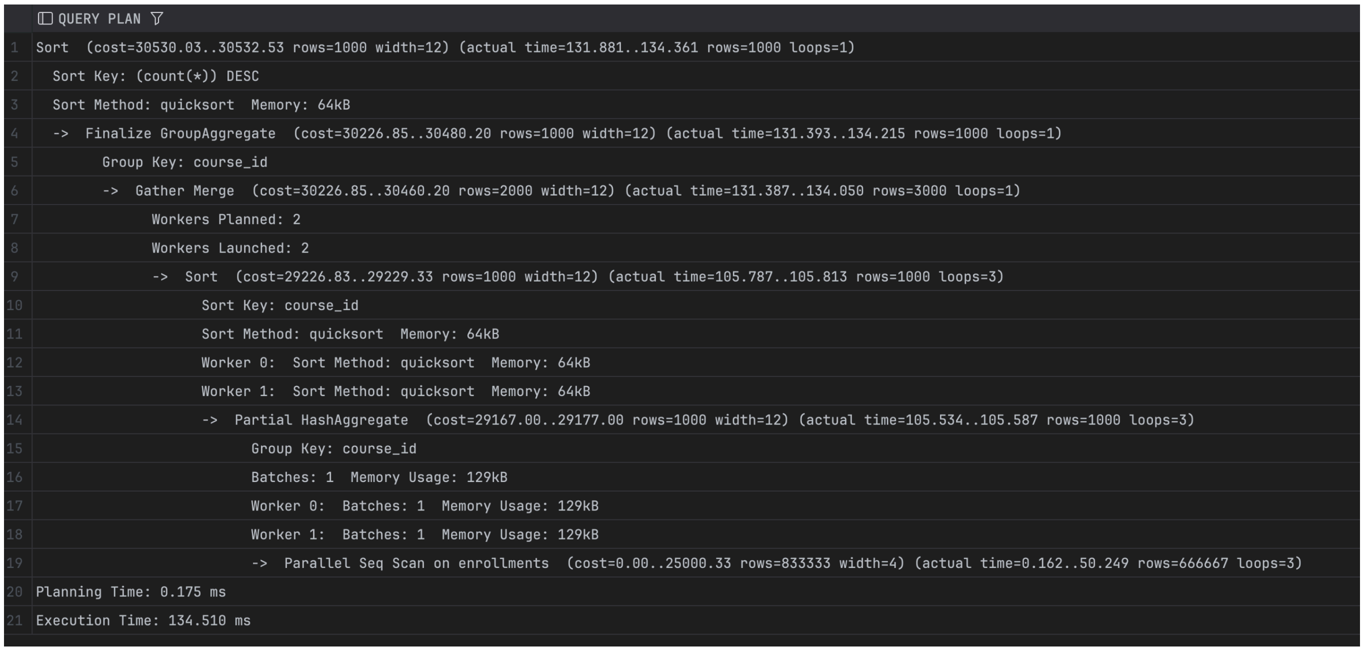
gain de temps

667.363 / 43.044 = 15.5x plus rapide avec les index.

4 Nombre d'isncription par cours

Sans index

```sql
EXPLAIN ANALYSE
select course_id, count(*) as nombre_inscriptions
from enrollments
group by course_id;
order by nombre_inscriptions desc;
```



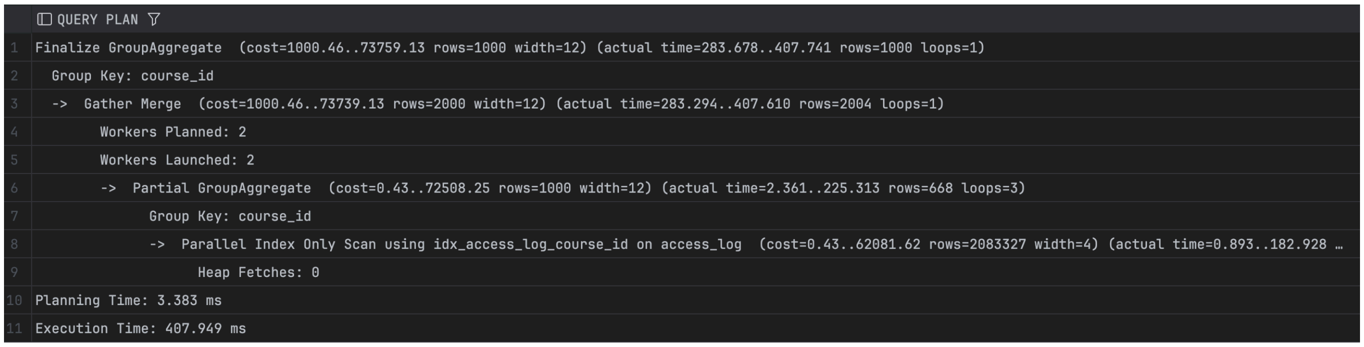
```
QUERY PLAN
Sort  (cost=30530.03..30532.53 rows=1000 width=12) (actual time=131.881..134.361 rows=1000 loops=1)
  Sort Key: (count(*)) DESC
  Sort Method: quicksort  Memory: 64kB
  ->  Finalize GroupAggregate  (cost=30226.85..30480.20 rows=1000 width=12) (actual time=131.393..134.215 rows=1000 loops=1)
        Group Key: course_id
        ->  Gather Merge  (cost=30226.85..30460.20 rows=2000 width=12) (actual time=131.387..134.050 rows=3000 loops=1)
              Workers Planned: 2
              Workers Launched: 2
              ->  Sort  (cost=29226.83..29229.33 rows=1000 width=12) (actual time=105.787..105.813 rows=1000 loops=3)
                    Sort Key: course_id
                    Sort Method: quicksort  Memory: 64kB
                    Worker 0:  Sort Method: quicksort  Memory: 64kB
                    Worker 1:  Sort Method: quicksort  Memory: 64kB
                    ->  Partial HashAggregate  (cost=29167.00..29177.00 rows=1000 width=12) (actual time=105.534..105.587 rows=1000 loops=3)
                          Group Key: course_id
                          Batches: 1  Memory Usage: 129kB
                          Worker 0:  Batches: 1  Memory Usage: 129kB
                          Worker 1:  Batches: 1  Memory Usage: 129kB
                          ->  Parallel Seq Scan on enrollments  (cost=0.00..25000.33 rows=833333 width=4) (actual time=0.162..50.249 rows=666667 loops=3)
Planning Time: 0.175 ms
Execution Time: 134.510 ms
```

PostgreSQL effectue un Parallel Seq Scan sur la table enrollments. Pour un GROUP BY qui agrège toute la table, le planificateur choisit un scan séquentiel parallèle plutôt qu'un index, car il faut lire toutes les lignes. Les données sont agrégées partiellement par course_id en parallèle (Partial HashAggregate), triées par course_id, puis fusionnées (Gather Merge), finalisées (Finalize GroupAggregate), et enfin triées par count(*) décroissant. Temps d'exécution : 134.510 ms.

5 Nombre d'accès par cours

Avec index

```sql
explain analyze
select course_id, count(*) as nombre_accès
from access_log
group by course_id;
order by nombre_accès desc;
```



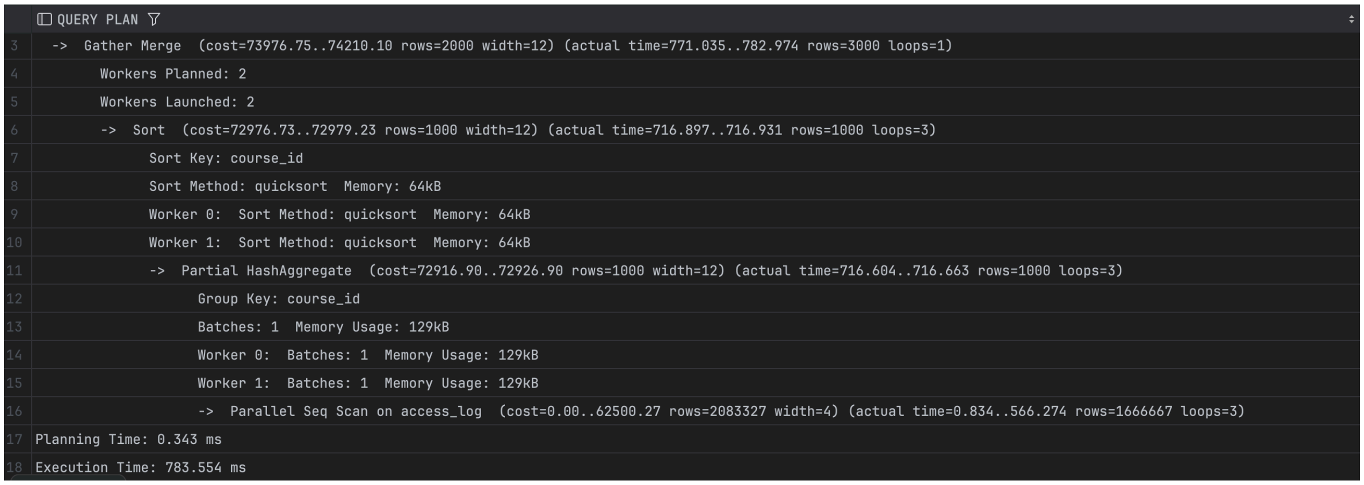
```
Finalize GroupAggregate  (cost=1000.46..73759.13 rows=1000 width=12) (actual time=283.678..407.741 rows=1000 loops=1)
  Group Key: course_id
  ->  Gather Merge  (cost=1000.46..73739.13 rows=2000 width=12) (actual time=283.294..407.610 rows=2004 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        ->  Partial GroupAggregate  (cost=0.43..72508.25 rows=1000 width=12) (actual time=2.361..225.313 rows=668 loops=3)
              Group Key: course_id
              ->  Parallel Index Only Scan using idx_access_log_course_id on access_log  (cost=0.43..62081.62 rows=2083327 width=4) (actual time=0.893..182.928 …
                    Heap Fetches: 0
Planning Time: 3.383 ms
Execution Time: 407.949 ms
```

PostgreSQL utilise l'index idx_access_log_course_id via un Parallel Index Only Scan. C'est un "Index Only Scan" : toutes les données nécessaires sont dans l'index, donc pas besoin d'accéder à la table (Heap Fetches: 0). Les données sont agrégées partiellement par course_id en parallèle, puis fusionnées et finalisées. Temps d'exécution : 407.949 ms.

Sans index

```
set enable_indexscan = off;
set enable_bitmapscan = off;

explain analyze
select course_id, count(*) as nombre_accès
from access_log
group by course_id;
order by nombre_accès desc;
```



```
  ->  Gather Merge  (cost=73976.75..74210.10 rows=2000 width=12) (actual time=771.035..782.974 rows=3000 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        ->  Sort  (cost=72976.73..72979.23 rows=1000 width=12) (actual time=716.897..716.931 rows=1000 loops=3)
              Sort Key: course_id
              Sort Method: quicksort  Memory: 64kB
              Worker 0:  Sort Method: quicksort  Memory: 64kB
              Worker 1:  Sort Method: quicksort  Memory: 64kB
              ->  Partial HashAggregate  (cost=72916.90..72926.90 rows=1000 width=12) (actual time=716.604..716.663 rows=1000 loops=3)
                    Group Key: course_id
                    Batches: 1  Memory Usage: 129kB
                    Worker 0:  Batches: 1  Memory Usage: 129kB
                    Worker 1:  Batches: 1  Memory Usage: 129kB
                    ->  Parallel Seq Scan on access_log  (cost=0.00..62500.27 rows=2083327 width=4) (actual time=0.834..566.274 rows=1666667 loops=3)
Planning Time: 0.343 ms
Execution Time: 783.554 ms
```

PostgreSQL effectue un scan séquentiel parallèle de la table access_log (1 666 667 lignes scannées), puis un Partial HashAggregate pour regrouper par course_id, un tri, et enfin un Gather Merge pour fusionner les résultats. Temps d'exécution : 783.554 ms.

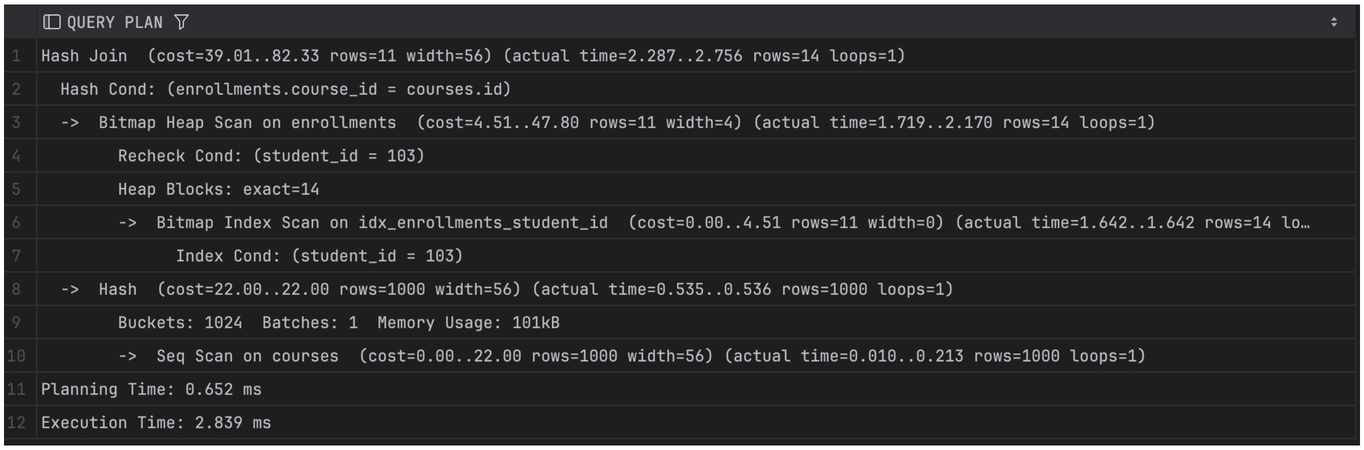
Gain de performance

783.554 / 407.949 = 1.9x plus rapide avec l'index.

6 les cours sur lequel un étudiant est inscrit

Avec index

```
explain analyze
select courses.* from enrollments
join courses on courses.id = enrollments.course_id
where enrollments.student_id = 103;
```



QUERY PLAN

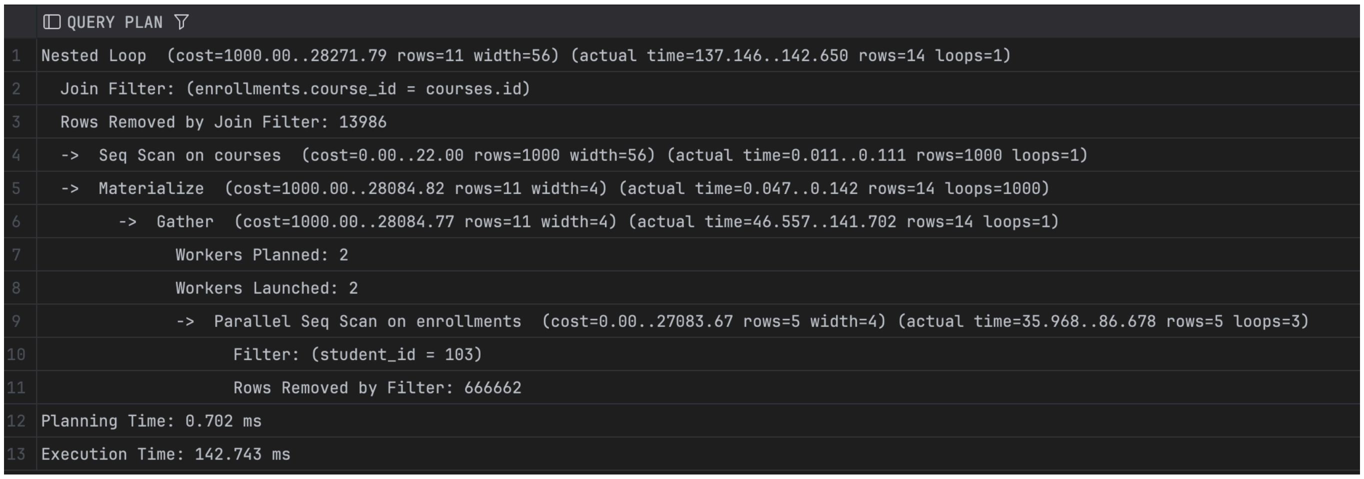
```
Hash Join  (cost=39.01..82.33 rows=11 width=56) (actual time=2.287..2.756 rows=14 loops=1)
  Hash Cond: (enrollments.course_id = courses.id)
  ->  Bitmap Heap Scan on enrollments  (cost=4.51..47.80 rows=11 width=4) (actual time=1.719..2.170 rows=14 loops=1)
        Recheck Cond: (student_id = 103)
        Heap Blocks: exact=14
        ->  Bitmap Index Scan on idx_enrollments_student_id  (cost=0.00..4.51 rows=11 width=0) (actual time=1.642..1.642 rows=14 lo…
              Index Cond: (student_id = 103)
  ->  Hash  (cost=22.00..22.00 rows=1000 width=56) (actual time=0.535..0.536 rows=1000 loops=1)
        Buckets: 1024  Batches: 1  Memory Usage: 101kB
        ->  Seq Scan on courses  (cost=0.00..22.00 rows=1000 width=56) (actual time=0.010..0.213 rows=1000 loops=1)
Planning Time: 0.652 ms
Execution Time: 2.839 ms
```

PostgreSQL utilise un Hash Join pour combiner les tables. Sur enrollments, il utilise l'index idx_enrollments_student_id via un Bitmap Index Scan pour filtrer les inscriptions de l'étudiant, puis un Bitmap Heap Scan pour récupérer les données. Sur courses, un Seq Scan est effectué (normal pour une table de 1000 lignes). Le Hash Join combine ensuite les résultats. Temps d'exécution : 2.839 ms.

Sans index

```
set enable_indexscan = off;
set enable_bitmapscan = off;

explain analyze
select courses.* from enrollments
join courses on courses.id = enrollments.course_id
where enrollments.student_id = 103;
```



```
QUERY PLAN
Nested Loop  (cost=1000.00..28271.79 rows=11 width=56) (actual time=137.146..142.650 rows=14 loops=1)
  Join Filter: (enrollments.course_id = courses.id)
  Rows Removed by Join Filter: 13986
  ->  Seq Scan on courses  (cost=0.00..22.00 rows=1000 width=56) (actual time=0.011..0.111 rows=1000 loops=1)
  ->  Materialize  (cost=1000.00..28084.82 rows=11 width=4) (actual time=0.047..0.142 rows=14 loops=1000)
        ->  Gather  (cost=1000.00..28084.77 rows=11 width=4) (actual time=46.557..141.702 rows=14 loops=1)
              Workers Planned: 2
              Workers Launched: 2
              ->  Parallel Seq Scan on enrollments  (cost=0.00..27083.67 rows=5 width=4) (actual time=35.968..86.678 rows=5 loops=3)
                    Filter: (student_id = 103)
                    Rows Removed by Filter: 666662
Planning Time: 0.702 ms
Execution Time: 142.743 ms
```

PostgreSQL effectue un Nested Loop join. Sur enrollments, un Parallel Seq Scan est utilisé (666 662 lignes scannées avant de trouver les 14 lignes correspondantes). Les résultats filtrés sont matérialisés et réutilisés pour chaque ligne de courses. Temps d'exécution : 142.743 ms.

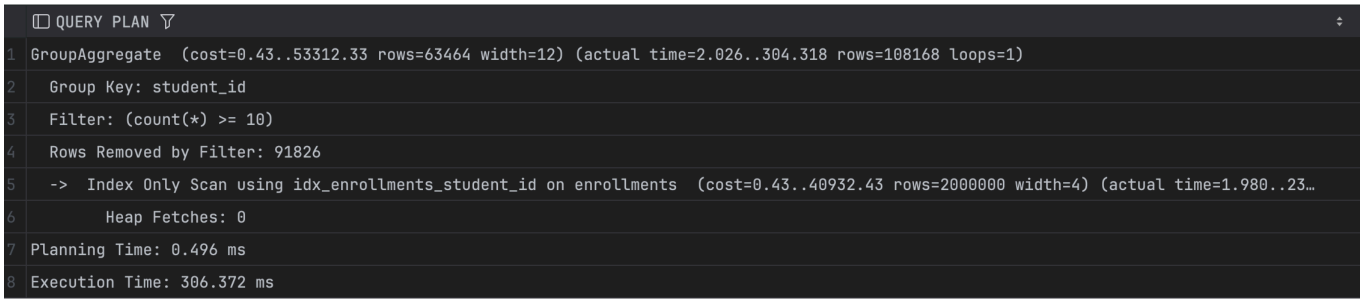
Gain de performance

142.743 / 2.839 = 50x plus rapide avec l'index.

7 les étudiant qui ont au moins 10 inscriptions

Avec Index

```sql
explain analyze
select student_id, count(*) as nombre_inscriptions from enrollments
group by student_id
having count(*) >= 10;
order by nombre_inscriptions desc;
```



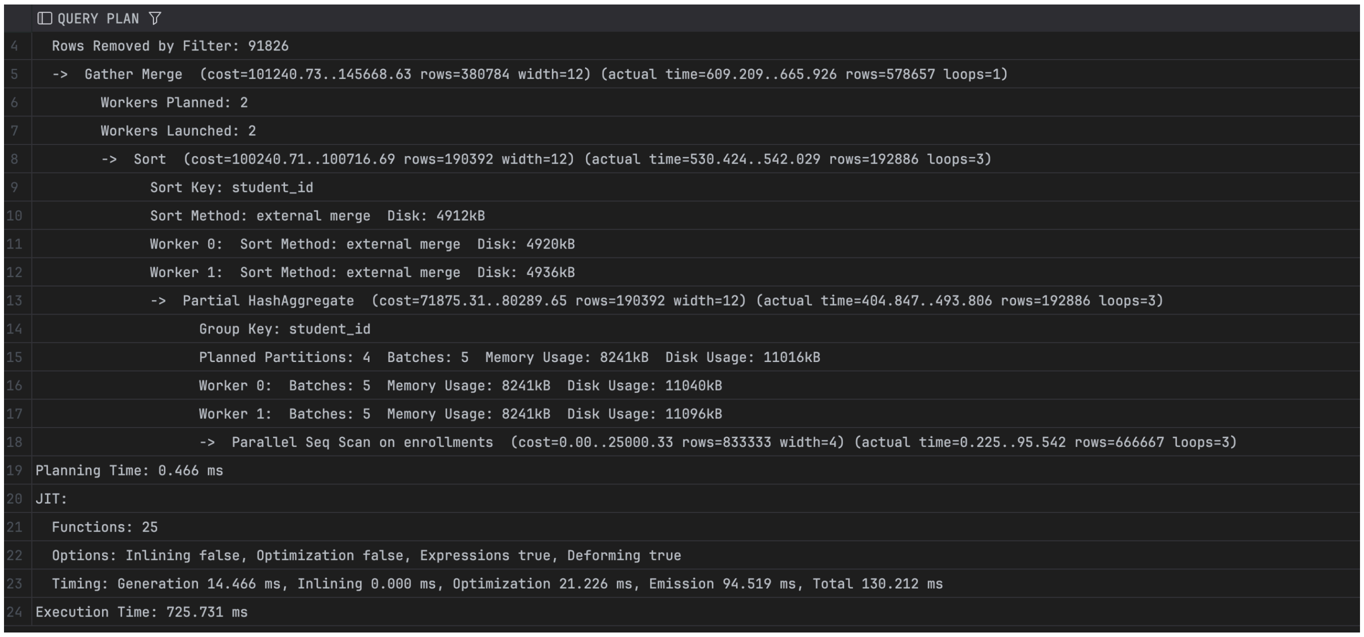
```
QUERY PLAN
GroupAggregate  (cost=0.43..53312.33 rows=63464 width=12) (actual time=2.026..304.318 rows=108168 loops=1)
  Group Key: student_id
  Filter: (count(*) >= 10)
  Rows Removed by Filter: 91826
  ->  Index Only Scan using idx_enrollments_student_id on enrollments  (cost=0.43..40932.43 rows=2000000 width=4) (actual time=1.980..23…
        Heap Fetches: 0
Planning Time: 0.496 ms
Execution Time: 306.372 ms
```

PostgreSQL utilise un Index Only Scan sur l'index idx_enrollments_student_id. Toutes les données nécessaires sont dans l'index, donc pas besoin d'accéder à la table (Heap Fetches: 0). Les données sont agrégées par student_id, puis filtrées avec HAVING count() >= 10. Temps d'exécution : 306.372 ms.

Sans index

```sql
set enable_indexscan = off;
set enable_bitmapscan = off;

explain analyze
select student_id, count(*) as nombre_inscriptions from enrollments
group by student_id
having count(*) >= 10;
order by nombre_inscriptions desc;
```



```
QUERY PLAN
  Rows Removed by Filter: 91826
  ->  Gather Merge  (cost=101240.73..145668.63 rows=380784 width=12) (actual time=609.209..665.926 rows=578657 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        ->  Sort  (cost=100240.71..100716.69 rows=190392 width=12) (actual time=530.424..542.029 rows=192886 loops=3)
              Sort Key: student_id
              Sort Method: external merge  Disk: 4912kB
              Worker 0:  Sort Method: external merge  Disk: 4920kB
              Worker 1:  Sort Method: external merge  Disk: 4936kB
              ->  Partial HashAggregate  (cost=71875.31..80289.65 rows=190392 width=12) (actual time=404.847..493.806 rows=192886 loops=3)
                    Group Key: student_id
                    Planned Partitions: 4  Batches: 5  Memory Usage: 8241kB  Disk Usage: 11016kB
                    Worker 0:  Batches: 5  Memory Usage: 8241kB  Disk Usage: 11040kB
                    Worker 1:  Batches: 5  Memory Usage: 8241kB  Disk Usage: 11096kB
                    ->  Parallel Seq Scan on enrollments  (cost=0.00..25000.33 rows=833333 width=4) (actual time=0.225..95.542 rows=666667 loops=3)
Planning Time: 0.466 ms
JIT:
  Functions: 25
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 14.466 ms, Inlining 0.000 ms, Optimization 21.226 ms, Emission 94.519 ms, Total 130.212 ms
Execution Time: 725.731 ms
```

PostgreSQL effectue un Parallel Seq Scan sur la table enrollments (666 667 lignes scannées par worker). Les données sont agrégées partiellement par student_id en parallèle, puis triées (avec spill sur disque), fusionnées, et filtrées. Temps d'exécution : 725.731 ms.

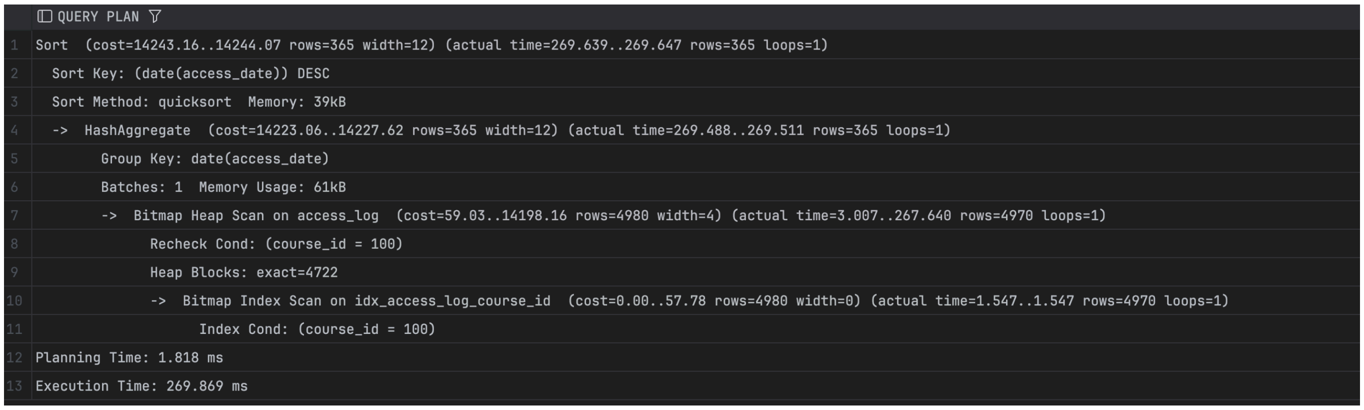
Gain de performance

725.731 / 306.372 = 2.4x plus rapide avec l'index.

8 nombre d'accès par jour sur un cours donné

Avec index

```sql
explain analyze
select date(access_date) as jour, count(*) as nombre_accès
from access_log
where course_id = 100
group by date(access_date)
order by jour desc;
```



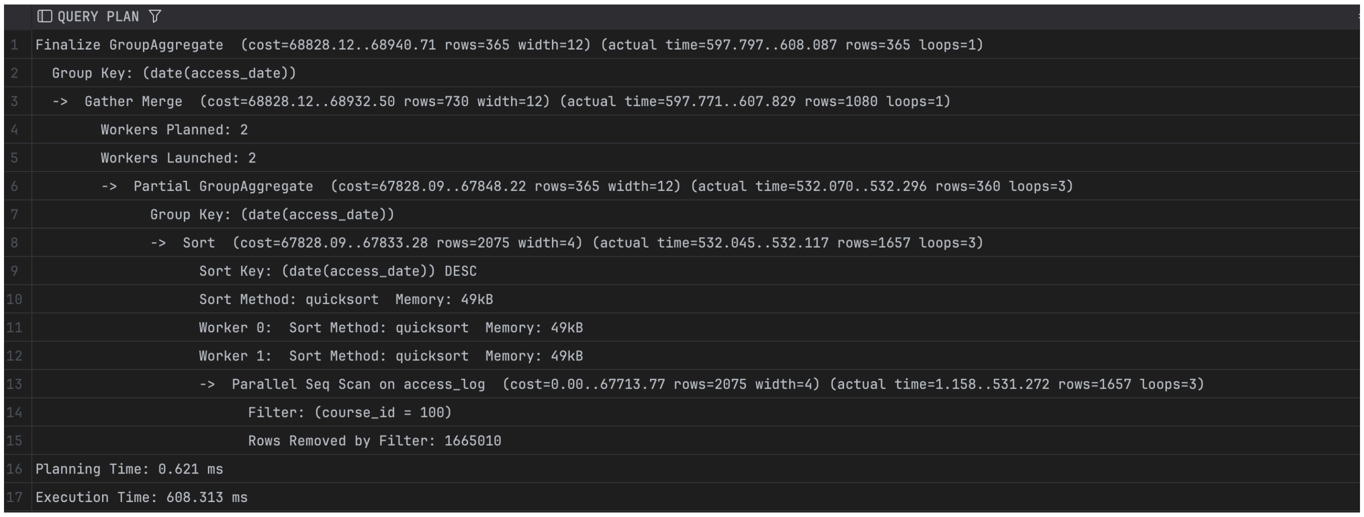
```
QUERY PLAN
Sort  (cost=14243.16..14244.07 rows=365 width=12) (actual time=269.639..269.647 rows=365 loops=1)
  Sort Key: (date(access_date)) DESC
  Sort Method: quicksort  Memory: 39kB
  ->  HashAggregate  (cost=14223.06..14227.62 rows=365 width=12) (actual time=269.488..269.511 rows=365 loops=1)
        Group Key: date(access_date)
        Batches: 1  Memory Usage: 61kB
        ->  Bitmap Heap Scan on access_log  (cost=59.03..14198.16 rows=4980 width=4) (actual time=3.007..267.640 rows=4970 loops=1)
              Recheck Cond: (course_id = 100)
              Heap Blocks: exact=4722
              ->  Bitmap Index Scan on idx_access_log_course_id  (cost=0.00..57.78 rows=4980 width=0) (actual time=1.547..1.547 rows=4970 loops=1)
                    Index Cond: (course_id = 100)
Planning Time: 1.818 ms
Execution Time: 269.869 ms
```

PostgreSQL utilise l'index idx_access_log_course_id via un Bitmap Index Scan pour filtrer les lignes avec course_id = 100, puis un Bitmap Heap Scan pour récupérer les données. Les données sont ensuite agrégées par date(access_date) avec un HashAggregate, puis triées par date décroissante. Le tri se fait en mémoire (39kB, pas de spill sur disque). Temps d'exécution : 269.869 ms.

Sans index

```sql
set enable_indexscan = off;
set enable_bitmapscan = off;

explain analyze
select date(access_date) as jour, count(*) as nombre_accès
from access_log
where course_id = 100
group by date(access_date)
order by jour desc;
```



```
QUERY PLAN
Finalize GroupAggregate  (cost=68828.12..68940.71 rows=365 width=12) (actual time=597.797..608.087 rows=365 loops=1)
  Group Key: (date(access_date))
  ->  Gather Merge  (cost=68828.12..68932.50 rows=730 width=12) (actual time=597.771..607.829 rows=1080 loops=1)
        Workers Planned: 2
        Workers Launched: 2
        ->  Partial GroupAggregate  (cost=67828.09..67848.22 rows=365 width=12) (actual time=532.070..532.296 rows=360 loops=3)
              Group Key: (date(access_date))
              ->  Sort  (cost=67828.09..67833.28 rows=2075 width=4) (actual time=532.045..532.117 rows=1657 loops=3)
                    Sort Key: (date(access_date)) DESC
                    Sort Method: quicksort  Memory: 49kB
                    Worker 0:  Sort Method: quicksort  Memory: 49kB
                    Worker 1:  Sort Method: quicksort  Memory: 49kB
                    ->  Parallel Seq Scan on access_log  (cost=0.00..67713.77 rows=2075 width=4) (actual time=1.158..531.272 rows=1657 loops=3)
                          Filter: (course_id = 100)
                          Rows Removed by Filter: 1665010
Planning Time: 0.621 ms
Execution Time: 608.313 ms
```

PostgreSQL effectue un Parallel Seq Scan sur la table access_log. 1 665 010 lignes sont scannées avant de trouver les 1 657 lignes correspondantes à course_id = 100. Les données sont ensuite triées, agrégées partiellement en parallèle, puis fusionnées et finalisées. Temps d'exécution : 608.313 ms.

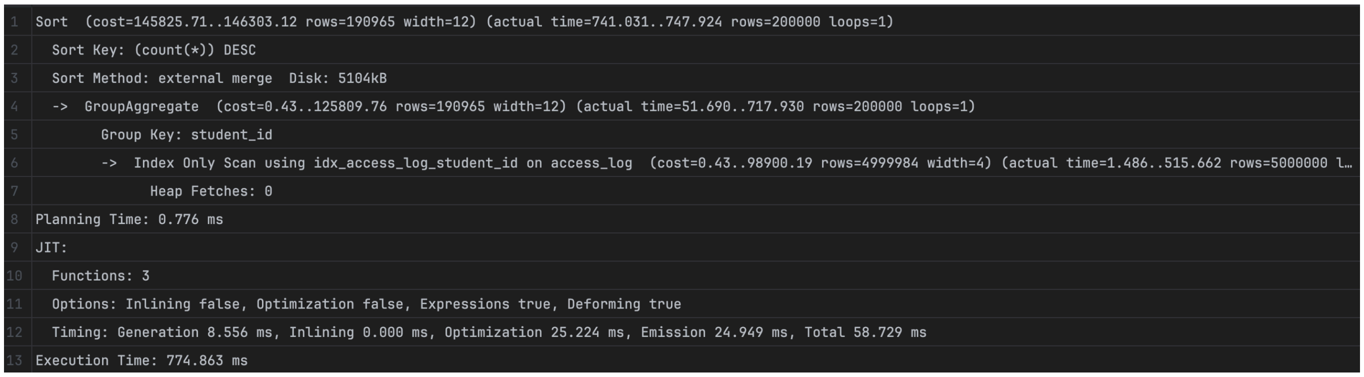
Gain de performance

608.313 / 269.869 = 2.3x plus rapide avec l'index.

9 les 50 étudiant les plus actif

Avec Index

```sql
explain analyze
select student_id, count(*) from access_log
group by student_id
order by count(*) desc
```



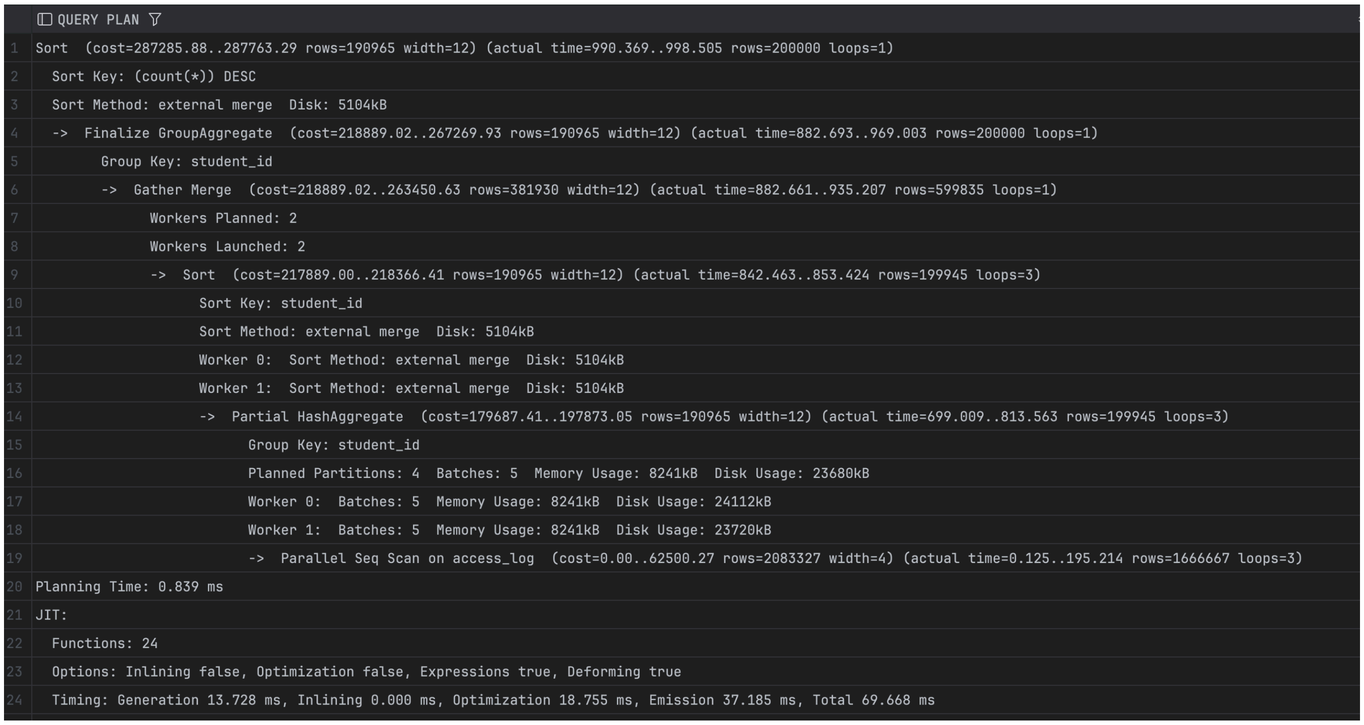
```
Sort  (cost=145825.71..146303.12 rows=190965 width=12) (actual time=741.031..747.924 rows=200000 loops=1)
  Sort Key: (count(*)) DESC
  Sort Method: external merge  Disk: 5104kB
  ->  GroupAggregate  (cost=0.43..125809.76 rows=190965 width=12) (actual time=51.690..717.930 rows=200000 loops=1)
        Group Key: student_id
        ->  Index Only Scan using idx_access_log_student_id on access_log  (cost=0.43..98900.19 rows=4999984 width=4) (actual time=1.486..515.662 rows=5000000 l…
              Heap Fetches: 0
Planning Time: 0.776 ms
JIT:
  Functions: 3
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 8.556 ms, Inlining 0.000 ms, Optimization 25.224 ms, Emission 24.949 ms, Total 58.729 ms
Execution Time: 774.863 ms
```

L'index idx_access_log_student_id est utilisé via un Index Only Scan. Les données sont agrégées par student_id, puis triées. Le tri déborde sur disque (5104kB). Temps : 774.863 ms.

Sans index

```sql
set enable_indexscan = off;
set enable_bitmapscan = off;

explain analyze
select student_id, count(*) from access_log
group by student_id
order by count(*) desc
```



```
Sort  (cost=287285.88..287763.29 rows=190965 width=12) (actual time=990.369..998.505 rows=200000 loops=1)
  Sort Key: (count(*)) DESC
  Sort Method: external merge  Disk: 5104kB
  ->  Finalize GroupAggregate  (cost=218889.02..267269.93 rows=190965 width=12) (actual time=882.693..969.003 rows=200000 loops=1)
        Group Key: student_id
        ->  Gather Merge  (cost=218889.02..263450.63 rows=381930 width=12) (actual time=882.661..935.207 rows=599835 loops=1)
              Workers Planned: 2
              Workers Launched: 2
              ->  Sort  (cost=217889.00..218366.41 rows=190965 width=12) (actual time=842.463..853.424 rows=199945 loops=3)
                    Sort Key: student_id
                    Sort Method: external merge  Disk: 5104kB
                    Worker 0:  Sort Method: external merge  Disk: 5104kB
                    Worker 1:  Sort Method: external merge  Disk: 5104kB
                    ->  Partial HashAggregate  (cost=179687.41..197873.05 rows=190965 width=12) (actual time=699.009..813.563 rows=199945 loops=3)
                          Group Key: student_id
                          Planned Partitions: 4  Batches: 5  Memory Usage: 8241kB  Disk Usage: 23680kB
                          Worker 0:  Batches: 5  Memory Usage: 8241kB  Disk Usage: 24112kB
                          Worker 1:  Batches: 5  Memory Usage: 8241kB  Disk Usage: 23720kB
                          ->  Parallel Seq Scan on access_log  (cost=0.00..62500.27 rows=2083327 width=4) (actual time=0.125..195.214 rows=1666667 loops=3)
Planning Time: 0.839 ms
JIT:
  Functions: 24
  Options: Inlining false, Optimization false, Expressions true, Deforming true
  Timing: Generation 13.728 ms, Inlining 0.000 ms, Optimization 18.755 ms, Emission 37.185 ms, Total 69.668 ms
```

PostgreSQL effectue un Parallel Seq Scan sur la table access_log (1 666 667 lignes scannées par worker). Les données sont agrégées partiellement en parallèle, puis triées. L'agrégation et le tri débordent sur disque (23680kB). Temps : 998.505 ms.

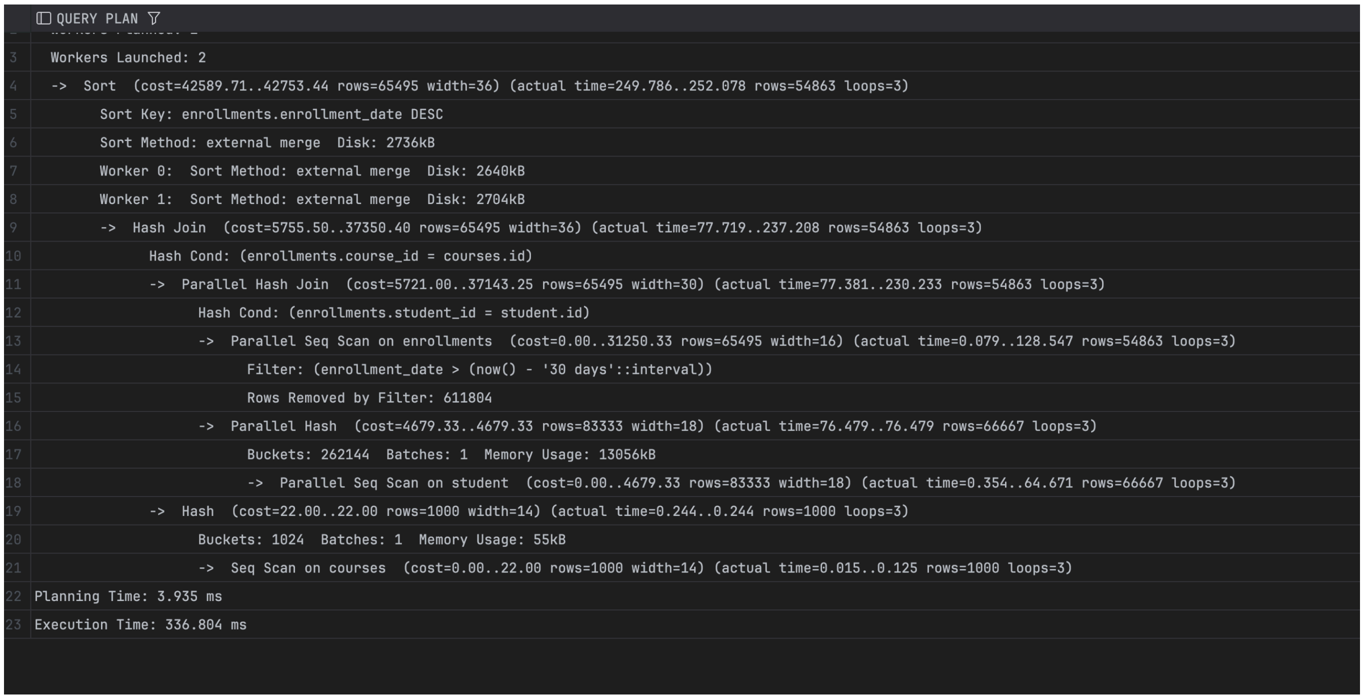
Gain de performance

998.505 / 774.863 = 1.3x plus rapide avec l'index.

10 les 10 inscription avec détail d'un étudaint + cours

Sans index

```
explain analyze
select student.id, student.name, courses.name, enrollments.enrollment_date
from enrollments
join student on student.id = enrollments.student_id
join courses on courses.id = enrollments.course_id
where enrollments.enrollment_date > now() - interval '30 days'
order by enrollments.enrollment_date desc
```



```
  Workers Launched: 2
  ->  Sort  (cost=42589.71..42753.44 rows=65495 width=36) (actual time=249.786..252.078 rows=54863 loops=3)
        Sort Key: enrollments.enrollment_date DESC
        Sort Method: external merge  Disk: 2736kB
        Worker 0:  Sort Method: external merge  Disk: 2640kB
        Worker 1:  Sort Method: external merge  Disk: 2704kB
        ->  Hash Join  (cost=5755.50..37350.40 rows=65495 width=36) (actual time=77.719..237.208 rows=54863 loops=3)
              Hash Cond: (enrollments.course_id = courses.id)
              ->  Parallel Hash Join  (cost=5721.00..37143.25 rows=65495 width=30) (actual time=77.381..230.233 rows=54863 loops=3)
                    Hash Cond: (enrollments.student_id = student.id)
                    ->  Parallel Seq Scan on enrollments  (cost=0.00..31250.33 rows=65495 width=16) (actual time=0.079..128.547 rows=54863 loops=3)
                          Filter: (enrollment_date > (now() - '30 days'::interval))
                          Rows Removed by Filter: 611804
                    ->  Parallel Hash  (cost=4679.33..4679.33 rows=83333 width=18) (actual time=76.479..76.479 rows=66667 loops=3)
                          Buckets: 262144  Batches: 1  Memory Usage: 13056kB
                          ->  Parallel Seq Scan on student  (cost=0.00..4679.33 rows=83333 width=18) (actual time=0.354..64.671 rows=66667 loops=3)
              ->  Hash  (cost=22.00..22.00 rows=1000 width=14) (actual time=0.244..0.244 rows=1000 loops=3)
                    Buckets: 1024  Batches: 1  Memory Usage: 55kB
                    ->  Seq Scan on courses  (cost=0.00..22.00 rows=1000 width=14) (actual time=0.015..0.125 rows=1000 loops=3)
Planning Time: 3.935 ms
Execution Time: 336.804 ms
```

PostgreSQL effectue des Parallel Seq Scan sur enrollments (611 804 lignes filtrées) et student. Les tables sont jointes via des Hash Join parallèles, puis triées. Le tri déborde sur disque (2736kB). Les index ne sont pas utilisés car le planificateur estime qu'un scan séquentiel parallèle est plus efficace pour cette requête. Temps : 336.804 ms.