Планы выполнения запросов на таблицы с 10000 записями

Первый запрос

Запрос: SELECT name FROM movies WHERE year > 2020 OR (year = 2020 AND month > 1) OR (year = 2020 AND month = 1 AND day > 1);  
  
План выполнения запроса: Seq Scan on movies (cost=0.00..136.47 rows=448 width=516) (actual time=0.023..47.256 rows=4364 loops=1)

Filter: ((year > 2020) OR ((year = 2020) AND (month > 1)) OR ((year = 2020) AND (month = 1) AND (day > 1)))

Rows Removed by Filter: 5636

Planning Time: 0.061 ms

Execution Time: 89.187 ms  
  
Запрос на данный момент не имеет ни каких индексов  
  
Потом я добавил индекс на поля year, mounth, day и вот что изменилось

Bitmap Heap Scan on movies (cost=70.88..303.99 rows=4000 width=10) (actual time=5.755..54.656 rows=4364 loops=1)

Recheck Cond: ((year > 2020) OR (year = 2020) OR (month = 1))

Filter: ((year > 2020) OR ((year = 2020) AND (month > 1)) OR ((year = 2020) AND (month = 1) AND (day > 1)))

Rows Removed by Filter: 468

Heap Blocks: exact=64

-> BitmapOr (cost=70.88..70.88 rows=5204 width=0) (actual time=5.711..5.768 rows=0 loops=1)

-> Bitmap Index Scan on idx\_movies\_year (cost=0.00..40.80 rows=3269 width=0) (actual time=3.701..3.710 rows=3269 loops=1)

Index Cond: (year > 2020)

-> Bitmap Index Scan on idx\_movies\_year (cost=0.00..16.52 rows=1098 width=0) (actual time=0.038..0.046 rows=1098 loops=1)

Index Cond: (year = 2020)

-> Bitmap Index Scan on idx\_movies\_month (cost=0.00..10.56 rows=837 width=0) (actual time=1.939..1.947 rows=837 loops=1)

Index Cond: (month = 1)

Planning Time: 0.467 ms

Execution Time: 96.470 ms  
  
Вывод:

Seq Scan на Bitmap Heap Scan: Перед добавлением индексов база данных выполняла последовательный поиск по таблице movies, что означает, что она просматривала каждую строку, чтобы найти те, которые соответствуют условиям фильтрации. После добавления индексов она выполняет Bitmap Heap Scan, который обычно более эффективен, поскольку использует битовый индекс для поиска соответствующих строк.

Уменьшение количества строк, исключенных фильтром: До добавления индексов фильтр удалял 5636 строк, но после добавления индексов удаляется только 468 строк. Это указывает на то, что индекс помогает более эффективно предварительно фильтровать строки.

Использование Bitmap Index Scan: После добавления индексов запрос использует Bitmap Index Scan по столбцам year и month. Это означает, что он использует индекс для быстрого определения интересующих строк для запроса.

Время выполнения: Несмотря на то, что Bitmap Heap Scan кажется более сложным, фактическое время выполнения не увеличилось значительно (89.187 мс до индексов и 96.470 мс после).

Время планирования: Время планирования слегка увеличилось с 0.061 мс до 0.467 мс, что ожидаемо, поскольку планировщик запросов теперь должен учитывать индексы.

Второй запрос

Запрос: SELECT COUNT(\*) FROM movies;

План выполнения запроса со статистикой: Aggregate (cost=228.00..228.01 rows=1 width=8) (actual time=203.495..203.528 rows=1 loops=1)

-> Seq Scan on movies (cost=0.00..203.00 rows=10000 width=0) (actual time=0.018..101.779 rows=10000 loops=1)

Planning Time: 0.107 ms

Execution Time: 203.599 ms

Учитывая, что время выполнения составляет около 203 мс, что является приемлемым для больших таблиц я решил не добавлять ни каких оптимизаций кроме как сделать count(id) по первичному ключу он своего рода уже являеться индексом пол умолчанию то я получил не большой выигрыш по оптимизации как видно ниже:  
  
Aggregate (cost=228.00..228.01 rows=1 width=8) (actual time=198.859..198.914 rows=1 loops=1)

-> Seq Scan on movies (cost=0.00..203.00 rows=10000 width=4) (actual time=0.016..96.941 rows=10000 loops=1)

Planning Time: 0.057 ms

Execution Time: 198.976 ms

Третий запрос

Запрос: SELECT name, duration FROM movies WHERE duration >= 120;

План выполнения запроса со статистикой: Seq Scan on movies (cost=0.00..228.00 rows=8388 width=14) (actual time=0.015..91.176 rows=8349 loops=1)

Filter: (duration >= 120)

Rows Removed by Filter: 1651

Planning Time: 0.103 ms

Execution Time: 169.312 ms

На данный момент таблица не каких индексов не имеет.

В теории по моему мнению надо создать индекс на столбец duration, чтобы ускорить выборку по условию фильтрации. Это особенно эффективно, если выборка включает в себя меньшинство записей из таблицы.  
  
Но после его добавления и обновления статистики я ни какого прироста не получил как видно ниже а на оборот по каким-то параметрам получил более дорогие значения, что говорит о том что в данном случае последовательное чтение считаеться более экономным и не смотря на то что пристувствует индекс.

Seq Scan on movies (cost=0.00..228.00 rows=8388 width=14) (actual time=0.026..91.603 rows=8349 loops=1)

Filter: (duration >= 120)

Rows Removed by Filter: 1651

Planning Time: 0.378 ms

Execution Time: 172.951 ms

Четвертый запрос

Зарос: SELECT at.type\_name, AVG(m.duration) AS average\_duration

FROM movies m

JOIN attribute\_values av ON m.movie\_id = av.movie\_id

JOIN attributes a ON av.attribute\_id = a.attribute\_id

JOIN attribute\_types at ON a.type\_id = at.type\_id

GROUP BY at.type\_name;

План выполнения запроса с подробной статистикой: HashAggregate (cost=1007.31..1044.81 rows=3000 width=41) (actual time=1363.523..1400.117 rows=2898 loops=1)

Group Key: at.type\_name

Batches: 1 Memory Usage: 625kB

-> Hash Join (cost=706.50..957.31 rows=10000 width=13) (actual time=503.444..1236.938 rows=10000 loops=1)

Hash Cond: (a.type\_id = at.type\_id)

-> Hash Join (cost=622.00..846.52 rows=10000 width=8) (actual time=436.080..952.859 rows=10000 loops=1)

Hash Cond: (av.attribute\_id = a.attribute\_id)

-> Hash Join (cost=328.00..526.26 rows=10000 width=8) (actual time=224.359..534.412 rows=10000 loops=1)

Hash Cond: (av.movie\_id = m.movie\_id)

-> Seq Scan on attribute\_values av (cost=0.00..172.00 rows=10000 width=8) (actual time=0.022..98.116 rows=10000 loops=1)

-> Hash (cost=203.00..203.00 rows=10000 width=8) (actual time=224.194..224.219 rows=10000 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 519kB

-> Seq Scan on movies m (cost=0.00..203.00 rows=10000 width=8) (actual time=0.015..109.930 rows=10000 loops=1)

-> Hash (cost=169.00..169.00 rows=10000 width=8) (actual time=211.484..211.509 rows=10000 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 519kB

-> Seq Scan on attributes a (cost=0.00..169.00 rows=10000 width=8) (actual time=0.018..101.081 rows=10000 loops=1)

-> Hash (cost=47.00..47.00 rows=3000 width=13) (actual time=67.304..67.328 rows=3000 loops=1)

Buckets: 4096 Batches: 1 Memory Usage: 173kB

-> Seq Scan on attribute\_types at (cost=0.00..47.00 rows=3000 width=13) (actual time=0.019..32.013 rows=3000 loops=1)

Planning Time: 0.549 ms

Execution Time: 1427.855 ms

Из представленного плана выполнения запроса видно, что запрос занимает значительное время, прежде всего, из-за множественных операций соединения (Hash Join) и последующей агрегации (HashAggregate)

Я решил добавить индексы по которым идут JOIN и на столбец который аггрегируеться но как видно по статистике ниже не которые значения так и остались а не которые выросли еще больше что говорит о том что в данном случае добавление индексов не было эффективным а наоборот затраты ресурсов еще больше выросли

CREATE INDEX ON attribute\_values(movie\_id);

CREATE INDEX ON attributes(attribute\_id);

CREATE INDEX ON attribute\_types(type\_id);

CREATE INDEX idx\_attribute\_types\_type\_name ON attribute\_types(type\_name);  
  
План выполнения со статистикой после добавления индексов:  
  
HashAggregate (cost=1007.31..1044.81 rows=3000 width=41) (actual time=1398.673..1433.658 rows=2898 loops=1)

Group Key: at.type\_name

Batches: 1 Memory Usage: 625kB

-> Hash Join (cost=706.50..957.31 rows=10000 width=13) (actual time=542.003..1280.621 rows=10000 loops=1)

Hash Cond: (a.type\_id = at.type\_id)

-> Hash Join (cost=622.00..846.52 rows=10000 width=8) (actual time=481.183..996.230 rows=10000 loops=1)

Hash Cond: (av.attribute\_id = a.attribute\_id)

-> Hash Join (cost=328.00..526.26 rows=10000 width=8) (actual time=269.074..578.769 rows=10000 loops=1)

Hash Cond: (av.movie\_id = m.movie\_id)

-> Seq Scan on attribute\_values av (cost=0.00..172.00 rows=10000 width=8) (actual time=0.019..98.089 rows=10000 loops=1)

-> Hash (cost=203.00..203.00 rows=10000 width=8) (actual time=269.013..269.038 rows=10000 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 519kB

-> Seq Scan on movies m (cost=0.00..203.00 rows=10000 width=8) (actual time=0.014..114.405 rows=10000 loops=1)

-> Hash (cost=169.00..169.00 rows=10000 width=8) (actual time=212.066..212.090 rows=10000 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 519kB

-> Seq Scan on attributes a (cost=0.00..169.00 rows=10000 width=8) (actual time=0.022..98.904 rows=10000 loops=1)

-> Hash (cost=47.00..47.00 rows=3000 width=13) (actual time=60.747..60.771 rows=3000 loops=1)

Buckets: 4096 Batches: 1 Memory Usage: 173kB

-> Seq Scan on attribute\_types at (cost=0.00..47.00 rows=3000 width=13) (actual time=0.019..29.476 rows=3000 loops=1)

Planning Time: 0.803 ms

Execution Time: 1460.204 ms

Пятый запрос

Запрос: SELECT m.name, COUNT(av.attribute\_id) AS attribute\_count

FROM movies m

JOIN attribute\_values av ON m.movie\_id = av.movie\_id

GROUP BY m.name;

План выполнения запроса с детальной статистикой: HashAggregate (cost=576.26..676.26 rows=10000 width=18) (actual time=657.831..726.076 rows=6317 loops=1)

Group Key: m.name

Batches: 1 Memory Usage: 913kB

-> Hash Join (cost=328.00..526.26 rows=10000 width=14) (actual time=219.883..539.471 rows=10000 loops=1)

Hash Cond: (av.movie\_id = m.movie\_id)

-> Seq Scan on attribute\_values av (cost=0.00..172.00 rows=10000 width=8) (actual time=0.018..102.828 rows=10000 loops=1)

-> Hash (cost=203.00..203.00 rows=10000 width=14) (actual time=219.831..219.856 rows=10000 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 597kB

-> Seq Scan on movies m (cost=0.00..203.00 rows=10000 width=14) (actual time=0.009..105.344 rows=10000 loops=1)

Planning Time: 0.326 ms

Execution Time: 783.114 ms

Выше план выполнения запроса и статистика без оптимизаций в плане добавления индексов.  
  
Ниже я добавлю индексы на столбцы, которые участвуют в соединении и агрегации. Учитывая, что выполняеться GROUP BY m.name, индекс на movies(name) может помочь, как и индекс на attribute\_values(movie\_id) для ускорения соединения.

Я добавил эти индексы и обновил статистику таблиц movies и attribute\_values и получил следующую вывод плана выполнения запроса и подробной статистики которую можно увидеть ниже:  
  
HashAggregate (cost=576.26..676.26 rows=10000 width=18) (actual time=641.054..712.200 rows=6317 loops=1)

Group Key: m.name

Batches: 1 Memory Usage: 913kB

-> Hash Join (cost=328.00..526.26 rows=10000 width=14) (actual time=209.615..524.830 rows=10000 loops=1)

Hash Cond: (av.movie\_id = m.movie\_id)

-> Seq Scan on attribute\_values av (cost=0.00..172.00 rows=10000 width=8) (actual time=0.013..98.139 rows=10000 loops=1)

-> Hash (cost=203.00..203.00 rows=10000 width=14) (actual time=209.564..209.754 rows=10000 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 597kB

-> Seq Scan on movies m (cost=0.00..203.00 rows=10000 width=14) (actual time=0.013..100.534 rows=10000 loops=1)

Planning Time: 0.435 ms

Execution Time: 769.483 ms

Исходя из этого результата можно увидеть небольшой прирост единиц только в actual time и не смотря на добавление индексов планировщие по прежнему делает выбор в сторону последовательного чтения так как по его мнению это менее ресурсозатратнее чем рандомное чтение. Вывод следующий что добовление индексов фактически не решил проблему производительности только произошли несколько микро оптимизаций.

Шестой запрос

Запрос: SELECT m.name, a.name AS attribute\_name

FROM movies m

JOIN attribute\_values av ON m.movie\_id = av.movie\_id

JOIN attributes a ON av.attribute\_id = a.attribute\_id

WHERE a.name = 'Оскар';

План выполнения запроса с подробной статистикой: Hash Join (cost=609.50..826.15 rows=7000 width=22) (actual time=377.731..812.769 rows=7000 loops=1)

Hash Cond: (av.movie\_id = m.movie\_id)

-> Hash Join (cost=281.50..479.76 rows=7000 width=16) (actual time=154.437..439.490 rows=7000 loops=1)

Hash Cond: (av.attribute\_id = a.attribute\_id)

-> Seq Scan on attribute\_values av (cost=0.00..172.00 rows=10000 width=8) (actual time=0.021..101.593 rows=10000 loops=1)

-> Hash (cost=194.00..194.00 rows=7000 width=16) (actual time=154.370..154.397 rows=7000 loops=1)

Buckets: 8192 Batches: 1 Memory Usage: 393kB

-> Seq Scan on attributes a (cost=0.00..194.00 rows=7000 width=16) (actual time=0.018..76.286 rows=7000 loops=1)

Filter: ((name)::text = 'Оскар'::text)

Rows Removed by Filter: 3000

-> Hash (cost=203.00..203.00 rows=10000 width=14) (actual time=223.255..223.283 rows=10000 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 597kB

-> Seq Scan on movies m (cost=0.00..203.00 rows=10000 width=14) (actual time=0.017..106.586 rows=10000 loops=1)

Planning Time: 0.447 ms

Execution Time: 885.873 ms

Статистика выше была выполненнна на таблице которая не имеет ни каких критериев оптимизации.

Мои действия по оптимизации:  
  
- Добавить индексы для ускорения соединений и фильтрации:

То есть создать индексы на столбцы movie\_id и attribute\_id в таблице attribute\_values, а также на столбец name в таблице attributes для ускорения соединений и фильтрации.

- Предположим что я часто выполняю запросы с группировкой или сортировкой по столбцу name в таблице movies, и по этому индекс на этом столбце может улучшить производительность.

Результат Плана выполнения запроса со статистикой выше перечисленных действий ниже:  
  
Hash Join (cost=609.50..826.15 rows=7000 width=22) (actual time=362.614..784.250 rows=7000 loops=1)

Hash Cond: (av.movie\_id = m.movie\_id)

-> Hash Join (cost=281.50..479.76 rows=7000 width=16) (actual time=155.675..432.185 rows=7000 loops=1)

Hash Cond: (av.attribute\_id = a.attribute\_id)

-> Seq Scan on attribute\_values av (cost=0.00..172.00 rows=10000 width=8) (actual time=0.032..96.928 rows=10000 loops=1)

-> Hash (cost=194.00..194.00 rows=7000 width=16) (actual time=155.574..155.599 rows=7000 loops=1)

Buckets: 8192 Batches: 1 Memory Usage: 393kB

-> Seq Scan on attributes a (cost=0.00..194.00 rows=7000 width=16) (actual time=0.034..79.495 rows=7000 loops=1)

Filter: ((name)::text = 'Оскар'::text)

Rows Removed by Filter: 3000

-> Hash (cost=203.00..203.00 rows=10000 width=14) (actual time=206.777..206.800 rows=10000 loops=1)

Buckets: 16384 Batches: 1 Memory Usage: 597kB

-> Seq Scan on movies m (cost=0.00..203.00 rows=10000 width=14) (actual time=0.022..99.070 rows=10000 loops=1)

Planning Time: 1.020 ms

Execution Time: 852.315 ms

Итог следующий: Как видно в сравнении с двумя планами до оптимизации и после добавления всех вышеперечисленных мною шагов произошла только микро оптимизация в actual time, при этом выбор планировщика по прежнему склонялся в сторону последовательного чтения так как он посчитал это менее ресурсо-затратным, что говорит о том что добавление индексов дал не значительный прирост оптимизации.

Планы выполнения запросов на таблицы с 10000000 строками

Первый запрос

Запрос: SELECT name FROM movies WHERE year > 2020 OR (year = 2020 AND month > 1) OR (year = 2020 AND month = 1 AND day > 1);

План выполнения запроса: Seq Scan on movies (cost=0.00..362580.18 rows=3739394 width=13) (actual time=21.174..50557.733 rows=3986699 loops=1)

Filter: ((year > 2020) OR ((year = 2020) AND (month > 1)) OR ((year = 2020) AND (month = 1) AND (day > 1)))

Rows Removed by Filter: 6013301

Planning Time: 0.113 ms

JIT:

Functions: 4

" Options: Inlining false, Optimization false, Expressions true, Deforming true"

" Timing: Generation 0.348 ms, Inlining 0.000 ms, Optimization 1.055 ms, Emission 20.031 ms, Total 21.435 ms"

Execution Time: 94180.854 ms

Это план выполнения запроса на таблице которая не имеет ни каких индексов в данный момент.

Наиболее эффективным способом оптимизации запроса может быть создание индекса по столбцу release\_date. Это позволит СУБД быстрее находить записи, соответствующие критериям фильтрации. Создадим индекс и проверим как изменился план выполнения запроса

Результат после плана оптимизации добавлкния индекса: Seq Scan on movies (cost=0.00..228092.76 rows=4029013 width=17) (actual time=17.380..42779.480 rows=3983959 loops=1)

Filter: (release\_date > '2020-01-01'::date)

Rows Removed by Filter: 6016041

Planning Time: 0.798 ms

JIT:

Functions: 4

" Options: Inlining false, Optimization false, Expressions true, Deforming true"

" Timing: Generation 5.219 ms, Inlining 0.000 ms, Optimization 2.490 ms, Emission 14.741 ms, Total 22.451 ms"

Execution Time: 81741.532 ms

Как мы видим добавление индекса не помогло так как некоторые характеристики просели по производительности. И не смотря на добавление индекса все равно произошло последовательное чтение.

Следующий шаг который я решил попробовать это сделать партицирование по диапазонам дат так как в таблице находиться 10000000 записей и фильтр идет с определенной даты то оно имеет место быть результат плана выполнения запроса ниже:

Append (cost=0.00..110864.91 rows=3983710 width=17)

-> Seq Scan on movies\_2020 movies\_1 (cost=0.00..22874.50 rows=1000103 width=17)

Filter: (release\_date > '2020-01-01'::date)

-> Seq Scan on movies\_2021 movies\_2 (cost=0.00..22812.25 rows=1000001 width=17)

Filter: (release\_date > '2020-01-01'::date)

-> Seq Scan on movies\_2022 movies\_3 (cost=0.00..22811.25 rows=1000001 width=17)

Filter: (release\_date > '2020-01-01'::date)

-> Seq Scan on movies\_2023 movies\_4 (cost=0.00..22436.74 rows=983562 width=17)

Filter: (release\_date > '2020-01-01'::date)

-> Seq Scan on movies\_2024 movies\_5 (cost=0.00..11.62 rows=43 width=520)

Filter: (release\_date > '2020-01-01'::date)

JIT:

Functions: 20

" Options: Inlining false, Optimization false, Expressions true, Deforming true"

Из этой статистики можно сделать вывод что в данном примере партицирование было уместным в плане оптимизации так как стоимость выполнения зароса уменьшилось в два раза.

Второй запрос

Запрос: SELECT COUNT(\*) FROM movies;

План выполнения запроса: Finalize Aggregate (cost=137324.61..137324.62 rows=1 width=8) (actual time=65386.577..65393.781 rows=1 loops=1)

-> Gather (cost=137324.39..137324.60 rows=2 width=8) (actual time=65364.887..65393.513 rows=3 loops=1)

Workers Planned: 2

Workers Launched: 2

-> Partial Aggregate (cost=136324.39..136324.40 rows=1 width=8) (actual time=65259.768..65259.800 rows=1 loops=3)

-> Parallel Index Only Scan using idx\_movies\_release\_date on movies (cost=0.43..125907.66 rows=4166692 width=0) (actual time=1.087..33012.948 rows=3333333 loops=3)

Heap Fetches: 0

Planning Time: 0.200 ms

JIT:

Functions: 8

" Options: Inlining false, Optimization false, Expressions true, Deforming true"

" Timing: Generation 2.302 ms, Inlining 0.000 ms, Optimization 0.749 ms, Emission 29.715 ms, Total 32.766 ms"

Execution Time: 65395.747 ms

В плане запроса уже используется Parallel Index Only Scan, что хорошо, поскольку это обычно быстрее, чем полное сканирование таблицы. И в случае с простым запросом COUNT(\*) вариантов для оптимизации не так много.

Третий запрос

Запрос: SELECT name, duration FROM movies WHERE duration >= 120;

План выполнения запроса: Seq Scan on movies (cost=0.00..228092.04 rows=8388566 width=17) (actual time=10.106..87905.727 rows=8359852 loops=1)

Filter: (duration >= 120)

Rows Removed by Filter: 1640148

Planning Time: 0.194 ms

JIT:

Functions: 4

" Options: Inlining false, Optimization false, Expressions true, Deforming true"

" Timing: Generation 1.379 ms, Inlining 0.000 ms, Optimization 1.765 ms, Emission 8.267 ms, Total 11.411 ms"

Execution Time: 168023.847 ms

Создание индекса по столбцу duration: Создание индекса по столбцу duration позволит СУБД использовать индексное сканирование для фильтрации строк, что значительно ускорит выполнение запроса. Результат статистики с добавленным индексом ниже:

Seq Scan on movies (cost=0.00..228092.16 rows=8366529 width=17) (actual time=5.420..88277.388 rows=8359852 loops=1)

Filter: (duration >= 120)

Rows Removed by Filter: 1640148

Planning Time: 0.619 ms

JIT:

Functions: 4

" Options: Inlining false, Optimization false, Expressions true, Deforming true"

" Timing: Generation 2.454 ms, Inlining 0.000 ms, Optimization 1.168 ms, Emission 4.182 ms, Total 7.803 ms"

Execution Time: 168923.074 ms

К сожалению как показано в результате выполнения плана добавление индекса не оправдало ожидание так как произошла только одна микро оптимизация - общее время выполнение первой строки.

Четвертый запрос

Запрос: EXPLAIN SELECT at.type\_name, AVG(m.duration) AS average\_duration

FROM movies m

JOIN attribute\_values av ON m.movie\_id = av.movie\_id

JOIN attributes a ON av.attribute\_id = a.attribute\_id

JOIN attribute\_types at ON a.type\_id = at.type\_id

GROUP BY at.type\_name;

План выполнения запроса без подробной статистики для выявления узких мест: Finalize GroupAggregate (cost=657274.57..658042.11 rows=3000 width=41)

Group Key: at.type\_name

-> Gather Merge (cost=657274.57..657974.61 rows=6000 width=41)

Workers Planned: 2

-> Sort (cost=656274.54..656282.04 rows=3000 width=41)

Sort Key: at.type\_name

-> Partial HashAggregate (cost=656071.28..656101.28 rows=3000 width=41)

Group Key: at.type\_name

-> Hash Join (cost=391656.19..635238.14 rows=4166629 width=13)

Hash Cond: (a.type\_id = at.type\_id)

-> Parallel Hash Join (cost=391571.69..624200.78 rows=4166629 width=8)

Hash Cond: (av.attribute\_id = a.attribute\_id)

-> Parallel Hash Join (cost=213119.12..385980.81 rows=4166629 width=8)

Hash Cond: (av.movie\_id = m.movie\_id)

-> Parallel Seq Scan on attribute\_values av (cost=0.00..113095.29 rows=4166629 width=8)

-> Parallel Hash (cost=144758.72..144758.72 rows=4166672 width=8)

-> Parallel Seq Scan on movies m (cost=0.00..144758.72 rows=4166672 width=8)

-> Parallel Hash (cost=110087.03..110087.03 rows=4167003 width=8)

-> Parallel Seq Scan on attributes a (cost=0.00..110087.03 rows=4167003 width=8)

-> Hash (cost=47.00..47.00 rows=3000 width=13)

-> Seq Scan on attribute\_types at (cost=0.00..47.00 rows=3000 width=13)

JIT:

Functions: 31

" Options: Inlining true, Optimization true, Expressions true, Deforming true"

Этот запрос, который выполняет агрегацию для вычисления средней продолжительности фильмов по типам атрибутов, занимает значительное время исходя из плана выполнения выше

После того как я создал индексы на ключевых столбцах, используемых в операциях соединения (JOIN) и фильтрации. И при этом выполнил команду ANALYZE для обновления статистики всех затрагиваемых таблиц. Я получил следующий план

Finalize GroupAggregate (cost=657251.72..658019.26 rows=3000 width=41)

Group Key: at.type\_name

-> Gather Merge (cost=657251.72..657951.76 rows=6000 width=41)

Workers Planned: 2

-> Sort (cost=656251.69..656259.19 rows=3000 width=41)

Sort Key: at.type\_name

-> Partial HashAggregate (cost=656048.43..656078.43 rows=3000 width=41)

Group Key: at.type\_name

-> Hash Join (cost=391637.32..635215.28 rows=4166630 width=13)

Hash Cond: (a.type\_id = at.type\_id)

-> Parallel Hash Join (cost=391552.82..624177.92 rows=4166630 width=8)

Hash Cond: (av.attribute\_id = a.attribute\_id)

-> Parallel Hash Join (cost=213117.71..385978.42 rows=4166630 width=8)

Hash Cond: (av.movie\_id = m.movie\_id)

-> Parallel Seq Scan on attribute\_values av (cost=0.00..113095.30 rows=4166630 width=8)

-> Parallel Hash (cost=144758.54..144758.54 rows=4166654 width=8)

-> Parallel Seq Scan on movies m (cost=0.00..144758.54 rows=4166654 width=8)

-> Parallel Hash (cost=110080.60..110080.60 rows=4166360 width=8)

-> Parallel Seq Scan on attributes a (cost=0.00..110080.60 rows=4166360 width=8)

-> Hash (cost=47.00..47.00 rows=3000 width=13)

-> Seq Scan on attribute\_types at (cost=0.00..47.00 rows=3000 width=13)

JIT:

Functions: 31

" Options: Inlining true, Optimization true, Expressions true, Deforming true"

Как мы видим результат статистики созданной после добавления индексов и обновления статистики всех затрагиваемых таблиц дал не большую микро оптимизацию на стоимость последней строки и все.

Пятый запрос

Запрос: EXPLAIN SELECT m.name, COUNT(av.attribute\_id) AS attribute\_count

FROM movies m

JOIN attribute\_values av ON m.movie\_id = av.movie\_id

GROUP BY m.name;

План выполнения без оптимизации: HashAggregate (cost=1439826.88..1657012.47 rows=9999912 width=21)

Group Key: m.name

Planned Partitions: 256

-> Hash Join (cost=386685.30..721083.20 rows=9999912 width=17)

Hash Cond: (av.movie\_id = m.movie\_id)

-> Seq Scan on attribute\_values av (cost=0.00..171428.12 rows=9999912 width=8)

-> Hash (cost=203091.69..203091.69 rows=9999969 width=17)

-> Seq Scan on movies m (cost=0.00..203091.69 rows=9999969 width=17)

JIT:

Functions: 12

" Options: Inlining true, Optimization true, Expressions true, Deforming true"

Анализируя план выполнения моего запроса, видно, что основное время занимает операция соединения (Hash Join) и последовательные сканирования (Seq Scan) таблиц movies и attribute\_values. Надо попробовать создать индексы на столбцы, используемые в операциях соединения. В данном случае, это movie\_id в таблицах movies и attribute\_values. Эти индексы позволят СУБД использовать более эффективные методы соединения, такие как Index Join, вместо Hash Join, что может существенно ускорить выполнение запроса.

Вот результат добавления индексов и обновление статистики:  
  
HashAggregate (cost=1439840.86..1657029.35 rows=10000027 width=21)

Group Key: m.name

Planned Partitions: 256

-> Hash Join (cost=386686.61..721086.40 rows=10000062 width=17)

Hash Cond: (av.movie\_id = m.movie\_id)

-> Seq Scan on attribute\_values av (cost=0.00..171429.62 rows=10000062 width=8)

-> Hash (cost=203092.27..203092.27 rows=10000027 width=17)

-> Seq Scan on movies m (cost=0.00..203092.27 rows=10000027 width=17)

JIT:

Functions: 12

" Options: Inlining true, Optimization true, Expressions true, Deforming true"

Как мы видим оптимизации вообще не произошло в данном случае мои предположения не сработали так как оказалось что postgres видит противоположные методы выполнения запроса более производительные и менее затратны.

Шестой запрос

Запрос: EXPLAIN SELECT m.name, a.name AS attribute\_name

FROM movies m

JOIN attribute\_values av ON m.movie\_id = av.movie\_id

JOIN attributes a ON av.attribute\_id = a.attribute\_id

WHERE a.name = 'Оскар';

План выполнения запроса без оптимизации: Hash Join (cost=708784.98..1184808.15 rows=7010044 width=26)

Hash Cond: (av.movie\_id = m.movie\_id)

-> Hash Join (cost=322098.38..638976.17 rows=7010044 width=17)

Hash Cond: (av.attribute\_id = a.attribute\_id)

-> Seq Scan on attribute\_values av (cost=0.00..171429.62 rows=10000062 width=8)

-> Hash (cost=193407.81..193407.81 rows=7009485 width=17)

-> Seq Scan on attributes a (cost=0.00..193407.81 rows=7009485 width=17)

Filter: ((name)::text = 'Оскар'::text)

-> Hash (cost=203092.27..203092.27 rows=10000027 width=17)

-> Seq Scan on movies m (cost=0.00..203092.27 rows=10000027 width=17)

JIT:

Functions: 20

" Options: Inlining true, Optimization true, Expressions true, Deforming true"

План показывает, что используются последовательные сканирования (Seq Scan) и хеш-соединения (Hash Join), что может быть неэффективно для больших таблиц. В таком случае попробую создать индексирование на следующих столбцах которые используються для соединения (JOIN). В данном случае это movie\_id в таблице attribute\_values и attribute\_id в таблице attributes.

Также полезно создать индекс на столбец name в таблице attributes, особенно если фильтрация по этому столбцу является общей операцией.

Ниже результат плана выполнения запроса с шагами которые я перечислил выше и с обновлением статистики всех затрагиваемых таблиц: Hash Join (cost=708442.95..1184078.41 rows=6990712 width=26)

Hash Cond: (av.movie\_id = m.movie\_id)

-> Hash Join (cost=321753.79..638519.62 rows=6990712 width=17)

Hash Cond: (av.attribute\_id = a.attribute\_id)

-> Seq Scan on attribute\_values av (cost=0.00..171429.65 rows=10000065 width=8)

-> Hash (cost=193413.15..193413.15 rows=6990451 width=17)

-> Seq Scan on attributes a (cost=0.00..193413.15 rows=6990451 width=17)

Filter: ((name)::text = 'Оскар'::text)

-> Hash (cost=203092.96..203092.96 rows=10000096 width=17)

-> Seq Scan on movies m (cost=0.00..203092.96 rows=10000096 width=17)

JIT:

Functions: 20

" Options: Inlining true, Optimization true, Expressions true, Deforming true"

Как мы видим мои предположения не оказались верными и произошли микро изминения на стоимости и по прежнему используются последовательные сканирования (Seq Scan) и хеш-соединения (Hash Join).

Итог: индексы не привносили каких-то значительных оптимизаций, значительная оптимизация произошла с использованием патриций на огромной таблице и так же эффективным способом оказалось использование материализованных представлений. Это не считая оптимизаций самого сервера таких как увеличение количество воркеров, или грамотного присвоение оценки таким параметрам как seq.reading и rand.reading в соответствии моего железа и др.