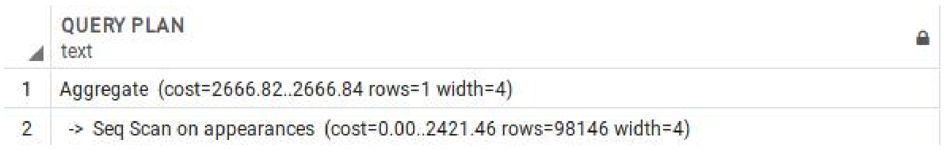
Query Optimization in Postgres

Philip Nelson and Scott Glaittli

19th November 2019

1. Aggregate optimization

```
SELECT max(G_p)
FROM appearances;
```

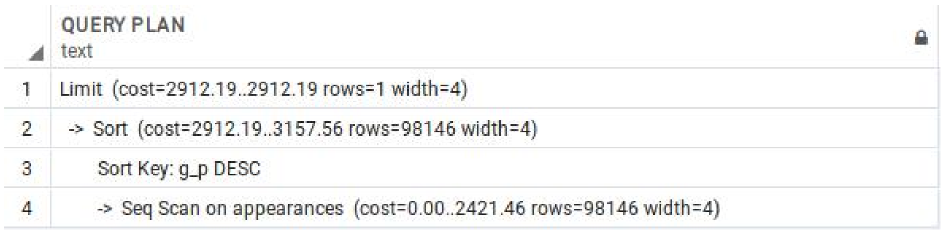


```
QUERY PLAN
text
1  Aggregate  (cost=2666.82..2666.84 rows=1 width=4)
2    -> Seq Scan on appearances  (cost=0.00..2421.46 rows=98146 width=4)
```

The initial cost was 2666.84.

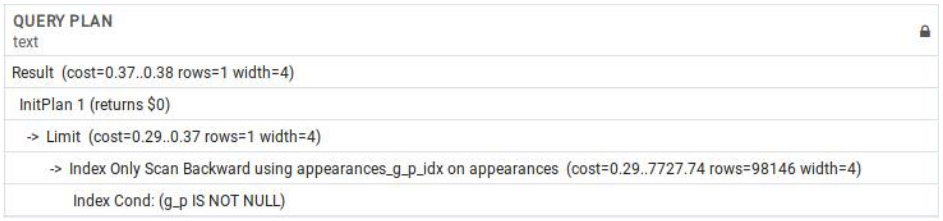
My first attempt to improve this query was to write it as a sorted selection of the `G_p` column then only display one row.

```
SELECT G_p
FROM appearances
ORDER BY G_p DESC
LIMIT 1;
```



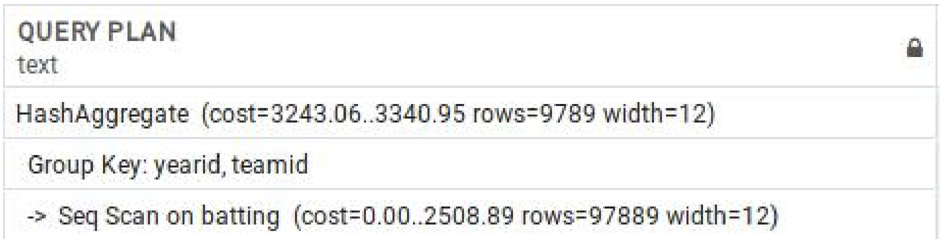
```
QUERY PLAN
text
1  Limit  (cost=2912.19..2912.19 rows=1 width=4)
2    -> Sort  (cost=2912.19..3157.56 rows=98146 width=4)
3         Sort Key: g_p DESC
4         -> Seq Scan on appearances  (cost=0.00..2421.46 rows=98146 width=4)
```

The cost was, 2912.19, which was 9.2% higher than the original. Next I created an index on the `G_p` column and ran the original query again which resulted in the query plan below. This time the cost was 0.38 which is a 99.98% decrease!


```
QUERY PLAN
text
Result  (cost=0.37..0.38 rows=1 width=4)
 InitPlan 1 (returns $0)
  -> Limit  (cost=0.29..0.37 rows=1 width=4)
     -> Index Only Scan Backward using appearances_g_p_idx on appearances  (cost=0.29..7727.74 rows=98146 width=4)
        Index Cond: (g_p IS NOT NULL)
```

2. Group by optimization

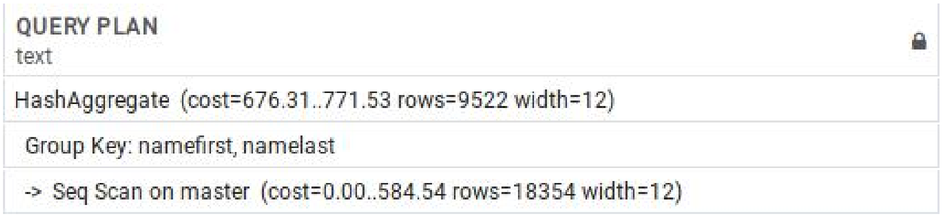
```
SELECT yearID, teamID, max(HR)
FROM batting
GROUP BY yearID, teamID;
```


```
QUERY PLAN
text
HashAggregate  (cost=3243.06..3340.95 rows=9789 width=12)
 Group Key: yearid, teamid
 -> Seq Scan on batting  (cost=0.00..2508.89 rows=97889 width=12)
```

The cost of the original query is 3340.95. I don't think this query can be optimized. I tried creating multiple different index types, changing the order of the aggregation and preselecting the rows affected and I never saw a decrease in total cost. It makes sense that the whole table has to be scanned to find the maximum HR values, and them associate them with a year and team id. All steps in the query execution plan are necessary.

3. Distinct

```
SELECT distinct nameFirst, nameLast
FROM master
```


```
QUERY PLAN
text
HashAggregate  (cost=676.31..771.53 rows=9522 width=12)
 Group Key: namefirst, namelast
 -> Seq Scan on master  (cost=0.00..584.54 rows=18354 width=12)
```

The cost of the original query is 771.53. First I created an index on `master(nameFisrt, nameLast)` which did not alter the query plan. This makes sense because this index would not provide any advantage over a linear scan on the master table. I couldn't think of simpler way to rewrite the query.

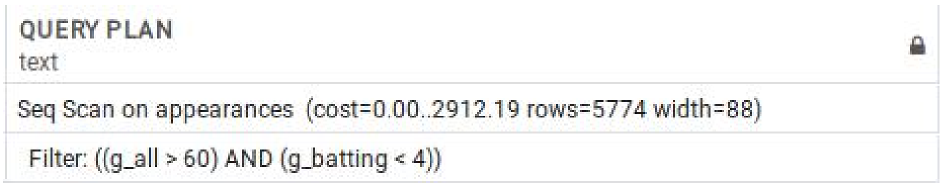
4. Selection

```
SELECT *
FROM appearances
WHERE g_all > 60
```

The original const of the query was 2666.82. I created an index on `appearances(g_all)` which reduced the cost to 2434.80. Then I created a second index on `appearances(g_all) where g_all > 60` which further reduced the cost to 2363.48. I couldn't find a way to rewrite the query that produces a lower cost that the original plan.

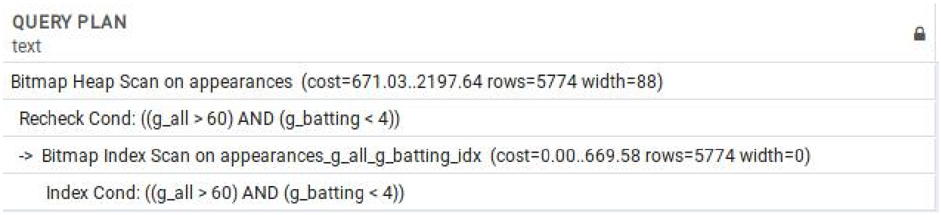
5. Selection conjunction

```
SELECT *
FROM appearances
WHERE g_all > 60 AND g_batting < 4
```



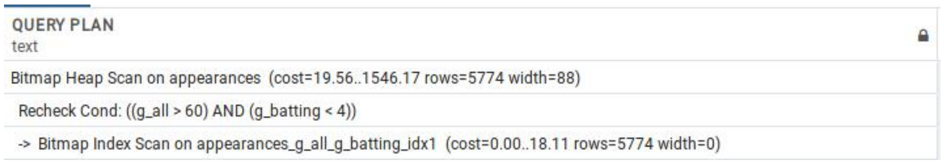
```
QUERY PLAN
text
Seq Scan on appearances  (cost=0.00..2912.19 rows=5774 width=88)
  Filter: ((g_all > 60) AND (g_batting < 4))
```

The cost of the original query is 2912.19. Creating an index on `appearances(g_all, g_batting)` produced the following query plan and reduced the cost by 24.5% to 2197.64.



```
QUERY PLAN
text
Bitmap Heap Scan on appearances  (cost=671.03..2197.64 rows=5774 width=88)
  Recheck Cond: ((g_all > 60) AND (g_batting < 4))
  -> Bitmap Index Scan on appearances_g_all_g_batting_idx  (cost=0.00..669.58 rows=5774 width=0)
        Index Cond: ((g_all > 60) AND (g_batting < 4))
```

I was able to decrease the cost by 46.9% from the original cost to 1546.17 by creating an index on `appearances(g_all, g_batting) where g_all > 60 AND g_batting < 4`. This is a very specific index to create so I would not consider this unless this exact query was going to be run frequently.



```
QUERY PLAN
text
Bitmap Heap Scan on appearances  (cost=19.56..1546.17 rows=5774 width=88)
  Recheck Cond: ((g_all > 60) AND (g_batting < 4))
  -> Bitmap Index Scan on appearances_g_all_g_batting_idx1  (cost=0.00..18.11 rows=5774 width=0)
```

6. Selection disjunction

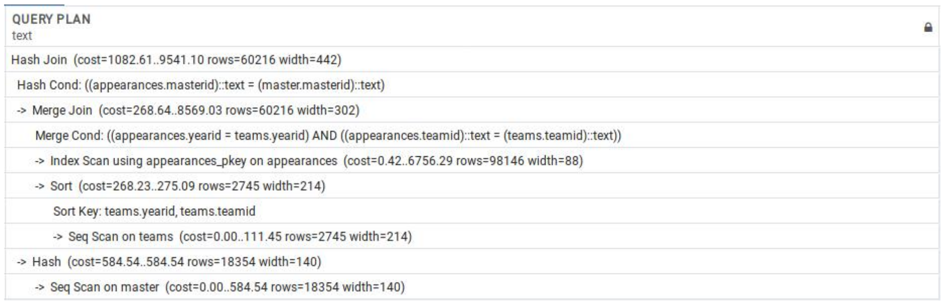
```
SELECT *
```

```
FROM appearances
WHERE g_all > 60 OR g_batting < 4
```

I tried creating an index on `appearances(g_all)` and `appearances(g_batting)` but postgres didn't use it. I tried reordering the conditionals with no change. I tried selecting `g_all > 60` and `g_batting < 4` separately, then selecting the common rows between them, but that produced query plan with a giant cost, greater than 12000. The only index that I found to make a difference was one on `appearances(g_all, g_batting) where g_all > 60 OR g_batting < 4` which produced a query plan that cost 2815.58.

7. Equi-join

```
SELECT *
FROM master, appearances, teams
WHERE master.masterId = appearances.masterId
  AND teams.teamId = appearances.teamId
  AND teams.yearId = appearances.yearId
```



```
QUERY PLAN
text
Hash Join  (cost=1082.61..9541.10 rows=60216 width=442)
 Hash Cond: ((appearances.masterid)::text = (master.masterid)::text)
 -> Merge Join  (cost=268.64..8569.03 rows=60216 width=302)
      Merge Cond: ((appearances.yearid = teams.yearid) AND ((appearances.teamid)::text = (teams.teamid)::text))
      -> Index Scan using appearances_pkey on appearances  (cost=0.42..6756.29 rows=98146 width=88)
      -> Sort  (cost=268.23..275.09 rows=2745 width=214)
           Sort Key: teams.yearid, teams.teamid
           -> Seq Scan on teams  (cost=0.00..111.45 rows=2745 width=214)
 -> Hash  (cost=584.54..584.54 rows=18354 width=140)
      -> Seq Scan on master  (cost=0.00..584.54 rows=18354 width=140)
```

The cost of the original query is 9541.10. My attempt to rewrite the query with the join syntax below resulted in the same exact query plan which confirms to me that `JOIN` and `WHERE` clauses have the same effect on query planning.

```
SELECT *
FROM master JOIN appearances ON master.masterId = appearances.masterId
JOIN teams
on teams.teamId = appearances.teamId
AND teams.yearId = appearances.yearId
```

Creating indices on `teams(yearid)`, `teams(teamid)`, `appearances(yearid)`, and `appearances(teamid)` didn't help to produce a different query plan either. I was a little bit surprised by this because I thought having indices to these columns would help improve the join.

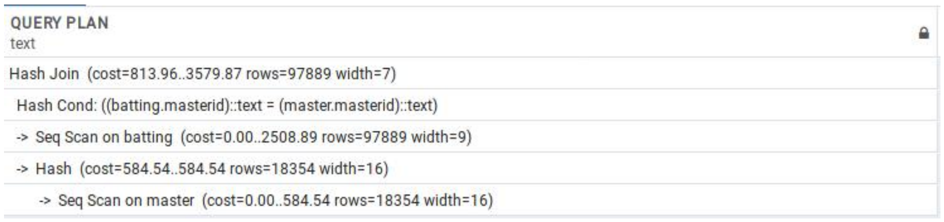
8. Theta-join

```
SELECT a.teamid, a.yearid, b.yearid, c.yearid
FROM teams a, teams b, teams c
WHERE a.teamid = b.teamid
  AND b.teamid = c.teamid
  AND a.w > b.w
  AND b.w > c.w
```

I started by creating an index on `teams (teamid)`. This reduced the number of operations in the query execution plan from 7, to 5, and reduced the total cost of the query from 84559.3 to 21843.4 for a cost difference of 62715.87!

9. Semi-join

```
SELECT nameLast
FROM master, batting
WHERE master.masterid = batting.masterid
```



```
QUERY PLAN
text
Hash Join  (cost=813.96..3579.87 rows=97889 width=7)
  Hash Cond: ((batting.masterid)::text = (master.masterid)::text)
  -> Seq Scan on batting  (cost=0.00..2508.89 rows=97889 width=9)
  -> Hash  (cost=584.54..584.54 rows=18354 width=16)
        -> Seq Scan on master  (cost=0.00..584.54 rows=18354 width=16)
```

The cost of the original query is 3579.87. Once again, rewriting the query with the `JOIN` syntax and adding indices to `batting(masterid)` did not change the original query plan.

10. Putting it all together

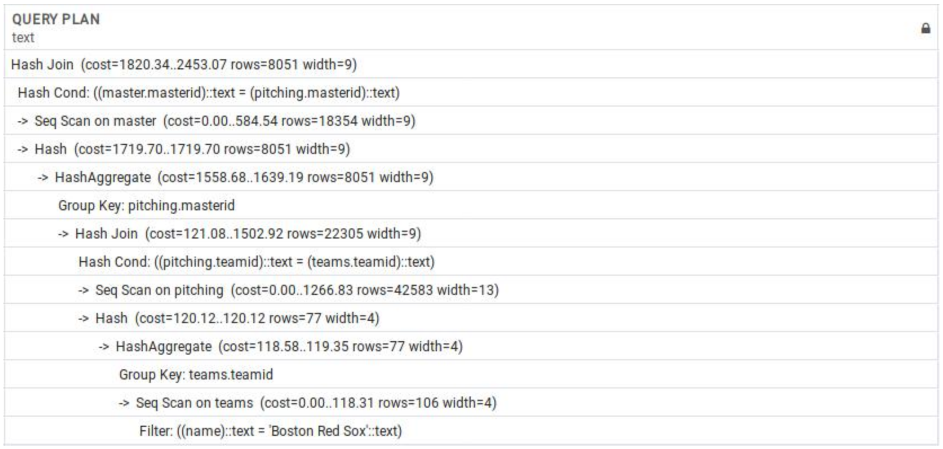
```
SELECT masterid, yearId, HR
FROM batting B
WHERE HR = (SELECT MAX(HR)
            FROM batting B2
            WHERE B.yearID = B2.yearID
           )
```

I created an index on `batting(yearid)`. The query execution plan was more complex, the sequential scan is replaced with a bitmap heap scan and a bitmap index scan for a total cost of about 13. Because of this, the aggregation step gets a cost of 1247.5 which is more than a 50% reduction from the non indexed query which had an aggregation cost of 2755.3.

11. Putting it all together two

```
SELECT masterid
FROM master
WHERE masterID IN
  (SELECT DISTINCT masterID
   FROM pitching
   WHERE teamID IN
```

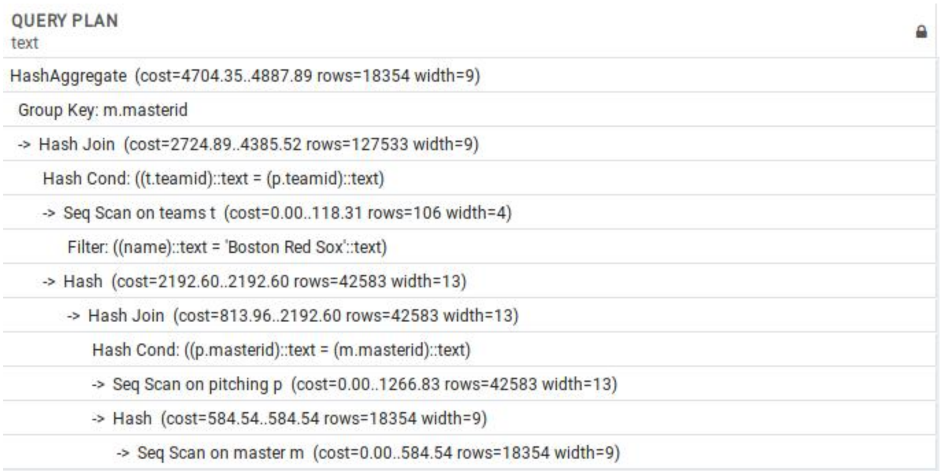
```
     (SELECT DISTINCT teamID
      FROM teams
      WHERE name = 'Boston Red Sox'))
```



```
QUERY PLAN
text
Hash Join  (cost=1820.34..2453.07 rows=8051 width=9)
  Hash Cond: ((master.masterid)::text = (pitching.masterid)::text)
  -> Seq Scan on master  (cost=0.00..584.54 rows=18354 width=9)
  -> Hash  (cost=1719.70..1719.70 rows=8051 width=9)
      -> HashAggregate  (cost=1558.68..1639.19 rows=8051 width=9)
          Group Key: pitching.masterid
          -> Hash Join  (cost=121.08..1502.92 rows=22305 width=9)
              Hash Cond: ((pitching.teamid)::text = (teams.teamid)::text)
              -> Seq Scan on pitching  (cost=0.00..1266.83 rows=42583 width=13)
              -> Hash  (cost=120.12..120.12 rows=77 width=4)
                  -> HashAggregate  (cost=118.58..119.35 rows=77 width=4)
                      Group Key: teams.teamid
                      -> Seq Scan on teams  (cost=0.00..118.31 rows=106 width=4)
                          Filter: ((name)::text = 'Boston Red Sox'::text)
```

The cost of the original query is 2453.07. Rewriting the query as a series of joins made the query cost increase to 4887.89.

```
select distinct m.masterid
from master m join pitching p
  on m.masterid = p.masterid
join teams t
  on p.teamid = t.teamid
where t.name = 'Boston Red Sox'
```

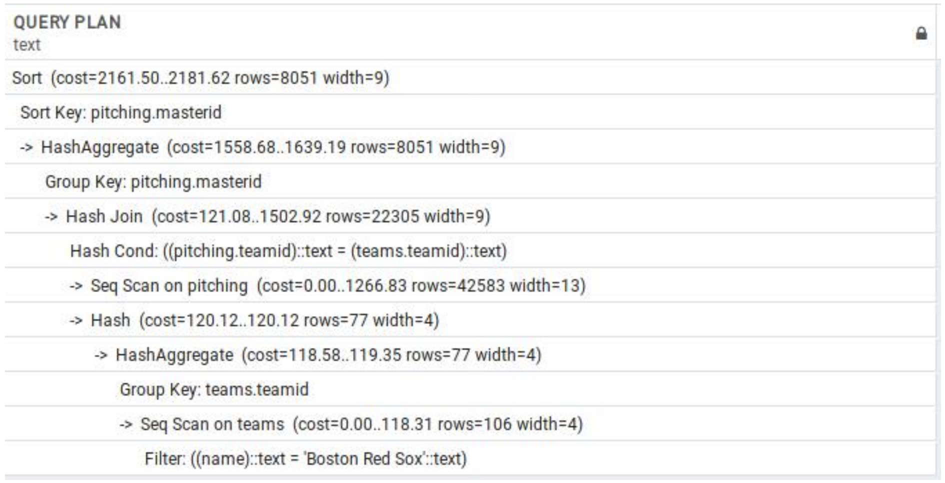


```
QUERY PLAN
text
HashAggregate  (cost=4704.35..4887.89 rows=18354 width=9)
  Group Key: m.masterid
  -> Hash Join  (cost=2724.89..4385.52 rows=127533 width=9)
      Hash Cond: ((t.teamid)::text = (p.teamid)::text)
      -> Seq Scan on teams t  (cost=0.00..118.31 rows=106 width=4)
          Filter: ((name)::text = 'Boston Red Sox'::text)
      -> Hash  (cost=2192.60..2192.60 rows=42583 width=13)
          -> Hash Join  (cost=813.96..2192.60 rows=42583 width=13)
              Hash Cond: ((p.masterid)::text = (m.masterid)::text)
              -> Seq Scan on pitching p  (cost=0.00..1266.83 rows=42583 width=13)
              -> Hash  (cost=584.54..584.54 rows=18354 width=9)
                  -> Seq Scan on master m  (cost=0.00..584.54 rows=18354 width=9)
```

Then I realized that the query was doing the same work twice in the outermost queries so I rewrote it as follows and got an 11.1% decrease to 2181.62.

```
SELECT DISTINCT masterID
```

```
  FROM pitching
  WHERE teamID IN
    (SELECT DISTINCT teamID
     FROM teams
     WHERE name = 'Boston Red Sox')
 order by 1 asc
```



```
QUERY PLAN
text
Sort  (cost=2161.50..2181.62 rows=8051 width=9)
  Sort Key: pitching.masterid
  ->  HashAggregate  (cost=1558.68..1639.19 rows=8051 width=9)
        Group Key: pitching.masterid
        ->  Hash Join  (cost=121.08..1502.92 rows=22305 width=9)
              Hash Cond: ((pitching.teamid)::text = (teams.teamid)::text)
              ->  Seq Scan on pitching  (cost=0.00..1266.83 rows=42583 width=13)
              ->  Hash  (cost=120.12..120.12 rows=77 width=4)
                    ->  HashAggregate  (cost=118.58..119.35 rows=77 width=4)
                          Group Key: teams.teamid
                          ->  Seq Scan on teams  (cost=0.00..118.31 rows=106 width=4)
                                Filter: ((name)::text = 'Boston Red Sox'::text)
```