1. (Postgres)The cost of evaluating the query:

	QUERY PLAN text
1	Aggregate (cost=2753.62..2753.64 rows=1 width=4) (actual time=41.159..41.160 rows=1 loops=1)
2	-> Seq Scan on batting (cost=0.00..2508.90 rows=97890 width=4) (actual time=0.041..25.023 rows=97890 loops=1)
3	Planning Time: 0.728 ms
4	Execution Time: 41.219 ms

What you did to optimize the query:

create index on batting(h);

The cost re-evaluated after optimization:

	QUERY PLAN text
1	Result (cost=0.39..0.40 rows=1 width=4) (actual time=0.044..0.044 rows=1 loops=1)
2	InitPlan 1 (returns $0)
3	-> Limit (cost=0.29..0.39 rows=1 width=4) (actual time=0.039..0.040 rows=1 loops=1)
4	-> Index Only Scan using one on batting (cost=0.29..8729.40 rows=91632 width=4) (actual time=0.038..0.038 rows=1 loops=1)
5	Index Cond: (h IS NOT NULL)
6	Heap Fetches: 1
7	Planning Time: 0.250 ms
8	Execution Time: 0.084 ms

2. The cost of evaluating the query:

	QUERY PLAN text
1	GroupAggregate (cost=10000010061.01..10000010749.68 rows=143 width=8) (actual time=75.874..97.191 rows=143 loops=1)
2	Group Key: yearid
3	-> Sort (cost=10000010061.01..10000010290.09 rows=91632 width=8) (actual time=75.831..85.985 rows=91477 loops=1)
4	Sort Key: yearid
5	Sort Method: external merge Disk: 1624kB
6	-> Seq Scan on batting (cost=10000000000.00..10000002508.90 rows=91632 width=8) (actual time=0.072..33.812 rows=91477 loops=1)
7	Filter: (h IS NOT NULL)
8	Rows Removed by Filter: 6413
9	Planning Time: 0.174 ms
10	Execution Time: 128.535 ms

What you did to optimize the query:

1. **create index on batting(yearid,h);**

The cost re-evaluated after optimization:

	QUERY PLAN text
1	HashAggregate (cost=4769.86..4771.29 rows=143 width=8) (actual time=59.610..59.629 rows=143 loops=1)
2	Group Key: yearid
3	-> Bitmap Heap Scan on batting (cost=1865.38..4311.70 rows=91632 width=8) (actual time=16.250..33.841 rows=91477 loops=1)
4	Recheck Cond: (h IS NOT NULL)
5	Heap Blocks: exact=1530
6	-> Bitmap Index Scan on batting_yearid_h_idx (cost=0.00..1842.47 rows=91632 width=0) (actual time=15.938..15.939 rows=91477 loo...
7	Index Cond: (h IS NOT NULL)
8	Planning Time: 0.243 ms
9	Execution Time: 59.775 ms

For this problem, we tried different ways on Postgres. Such as,

1. Create index on batting(yearid);
2. Create index on batting(h);
3. Create index on batting(yearid,h);

None of above improve the cost of the query. Since Postgres is knowing that create an index will not improve the result. Therefore, it will keep working on it without the created index. However, for MySQL, when we create an index, it will work with the created index. Therefore, when we create index on MySQL it improved the cost time.

3. The cost of evaluating the query:

	QUERY PLAN text
1	Hash Join (cost=1197.86..8413.48 rows=97890 width=140) (actual time=14.431..99.746 rows=97890 loops=1)
2	Hash Cond: ((batting.masterid)::text = (master.masterid)::text)
3	-> Index Only Scan using batting_pkey on batting (cost=0.42..6959.02 rows=97890 width=9) (actual time=0.022.....
4	Heap Fetches: 97890
5	-> Hash (cost=968.00..968.00 rows=18355 width=140) (actual time=14.308..14.308 rows=18355 loops=1)
6	Buckets: 32768 Batches: 1 Memory Usage: 3402kB
7	-> Index Scan using master_pkey on master (cost=0.29..968.00 rows=18355 width=140) (actual time=0.006..6...
8	Planning Time: 0.407 ms
9	Execution Time: 105.566 ms

What you did to optimize the query:

(postgres)

1. create index num_two on master(masterid);
2. **create index on batting(masterid);**
3. create unique index one1 on batting(masterid);
4. create index one on batting using btree (masterid);

(MySQL)

1.create index num_one using btree on batting(masterid);

The cost re-evaluated after optimization:

	QUERY PLAN text
1	Hash Join (cost=1197.86..5979.67 rows=97890 width=140) (actual time=8.515..74.821 rows=97890 loops=1)
2	Hash Cond: ((batting.masterid)::text = (master.masterid)::text)
3	-> Index Only Scan using batting_masterid_idx on batting (cost=0.42..4525.22 rows=97890 width=9) (actual time=0.013..21.331 rows=97890 loops=1)
4	Heap Fetches: 97890
5	-> Hash (cost=968.00..968.00 rows=18355 width=140) (actual time=8.482..8.482 rows=18355 loops=1)
6	Buckets: 32768 Batches: 1 Memory Usage: 3402kB
7	-> Index Scan using master_pkey on master (cost=0.29..968.00 rows=18355 width=140) (actual time=0.003..4.322 rows=18355 loops=1)
8	Planning Time: 0.463 ms
9	Execution Time: 80.296 ms

For this question, the query did not get any improve on total cost after create index. The reason why is because in Postgres when we create an index for this query it will still use sequential scan instead of created index. When I turned off the sequential scan, the query cost went really high. However, the actual time after create index improved.

4. The cost of evaluating the query:

	QUERY PLAN text
1	Hash Join (cost=1555.45..2278.25 rows=8096 width=140) (actual time=32.459..44.057 rows=8810 loops=1)
2	Hash Cond: ((m.masterid)::text = (pitching.masterid)::text)
3	-> Seq Scan on master m (cost=0.00..584.55 rows=18355 width=140) (actual time=0.019..2.175 rows=18355 loops=1)
4	-> Hash (cost=1454.25..1454.25 rows=8096 width=9) (actual time=32.280..32.281 rows=8810 loops=1)
5	Buckets: 16384 (originally 8192) Batches: 1 (originally 1) Memory Usage: 489kB
6	-> HashAggregate (cost=1373.29..1454.25 rows=8096 width=9) (actual time=23.025..28.128 rows=8810 loops=1)
7	Group Key: (pitching.masterid)::text
8	-> Seq Scan on pitching (cost=0.00..1266.83 rows=42583 width=9) (actual time=0.020..11.681 rows=42583 loops=1)
9	Planning Time: 0.545 ms
10	Execution Time: 45.164 ms

What you did to optimize the query:

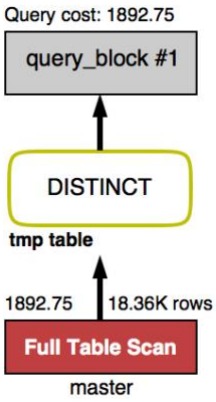
(postgres)

1. **CREATE INDEX ON pitching(masterid);**
   CREATE INDEX ON master(masterid);

	QUERY PLAN text
1	Result (cost=0.63..585.18 rows=18355 width=140) (actual time=0.113..13.850 rows=18355 loops=1)
2	One-Time Filter: $1
3	InitPlan 1 (returns $1)
4	-> Nested Loop (cost=0.29..14542.60 rows=42583 width=0) (actual time=0.101..0.101 rows=1 loops=1)
5	-> Seq Scan on pitching (cost=0.00..1266.83 rows=42583 width=9) (actual time=0.050..0.050 rows=1 loops=1)
6	-> Index Only Scan using master_pkey on master ma (cost=0.29..0.31 rows=1 width=9) (actual time=0.045..0.045 rows=1 loops=1)
7	Index Cond: (masterid = (pitching.masterid)::text)
8	Heap Fetches: 0
9	-> Seq Scan on master m (cost=0.63..585.18 rows=18355 width=140) (actual time=0.005..3.070 rows=18355 loops=1)
10	Planning Time: 0.623 ms
11	Execution Time: 15.162 ms

For this query, basically same problem as the previous query. We tried to create index to optimize the query on Postgres, but same as the previous query. We only can improve the actual time but not the total cost after create query. Also, since we want to rewrite the where exists statement in MySQL, we tried to use limit 1 which means once we find the match it will stop searching/scan in the table. But this did not help.

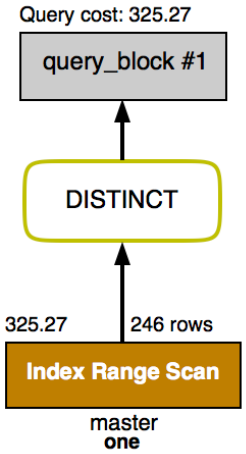
5. The cost of evaluating the query:



What you did to optimize the query:

1. **create index one on master(birthState,birthCountry);**

The cost re-evaluated after optimization:



6. The cost of evaluating the query:

	QUERY PLAN text
1	Seq Scan on master (cost=0.00..584.55 rows=18355 width=7) (actual time=0.022..6.462 rows=18355 loops=1)
2	Planning Time: 0.065 ms
3	Execution Time: 7.719 ms

What you did to optimize the query:

1. **Create index on master(birthState,birthCountry);**

The cost re-evaluates after optimization:

	QUERY PLAN text
1	Index Only Scan using master_birthstate_birthcountry_idx on master (cost=0.29..491.61 rows=18355 width=7) (actual time=0.113..3.713 rows=18355 loops=1)
2	Heap Fetches: 0
3	Planning Time: 0.355 ms
4	Execution Time: 4.780 ms

For this query, the total cost decrease about 100 and actual time decrease 50% after create an index.

7. The cost of evaluating the query:

	QUERY PLAN text
1	HashAggregate (cost=4381.88..5388.38 rows=100650 width=4) (actual time=66.190..66.620 rows=251 loops=1)
2	Group Key: batting.h
3	-> Append (cost=0.00..4130.25 rows=100650 width=4) (actual time=0.011..45.188 rows=100651 loops=1)
4	-> Seq Scan on batting (cost=0.00..2508.90 rows=97890 width=4) (actual time=0.011..32.918 rows=97890 loops=1)
5	-> Seq Scan on teams (cost=0.00..111.60 rows=2760 width=4) (actual time=0.034..0.665 rows=2761 loops=1)
6	Planning Time: 0.066 ms
7	Execution Time: 66.792 ms

What you did to optimize the query:

Since the union on hit from batting with games from teams output is the same as select distinct hits from batting. Therefore, we change the query to use distinct option.

1. CREATE INDEX num_one ON batting(h);
   SELECT DISTINCT h
   FROM batting;
2. **SELECT DISTINCT ***
   from (SELECT h
   FROM batting
   UNION all
   SELECT g
   FROM teams) as a;

The cost re-evaluates after optimization:

	QUERY PLAN text
1	HashAggregate (cost=3375.38..3377.38 rows=200 width=4) (actual time=53.277..53.307 rows=251 loops=1)
2	Group Key: batting.h
3	-> Append (cost=0.00..3123.75 rows=100650 width=4) (actual time=0.012..35.028 rows=100651 loops=1)
4	-> Seq Scan on batting (cost=0.00..2508.90 rows=97890 width=4) (actual time=0.012..23.607 rows=97890 loops=1)
5	-> Seq Scan on teams (cost=0.00..111.60 rows=2760 width=4) (actual time=0.012..0.510 rows=2761 loops=1)
6	Planning Time: 0.240 ms
7	Execution Time: 53.352 ms

8. The cost of evaluating the query:

	QUERY PLAN text
1	Unique (cost=3593.05..3593.06 rows=1 width=12) (actual time=59.546..59.658 rows=193 loops=1)
2	-> Sort (cost=3593.05..3593.06 rows=1 width=12) (actual time=59.545..59.583 rows=483 loops=1)
3	Sort Key: master.namelast, master.namefirst
4	Sort Method: quicksort Memory: 48kB
5	-> Nested Loop (cost=120.82..3593.04 rows=1 width=12) (actual time=1.972..59.065 rows=483 loops=1)
6	-> Nested Loop (cost=120.53..3592.72 rows=1 width=27) (actual time=1.950..56.247 rows=483 loops=1)
7	-> Nested Loop (cost=120.25..3592.41 rows=1 width=42) (actual time=1.643..53.295 rows=770 loops=1)
8	Join Filter: ((b.masterid)::text = (b_2.masterid)::text)
9	-> Nested Loop (cost=119.83..3591.87 rows=1 width=22) (actual time=1.623..50.204 rows=804 loops=1)
10	-> Nested Loop (cost=119.55..3587.55 rows=14 width=33) (actual time=1.588..45.336 rows=1257 loops=1)
11	-> Hash Join (cost=119.13..3399.08 rows=354 width=13) (actual time=1.536..33.790 rows=1479 loops=1)
12	Hash Cond: (((b.teamid)::text = (t.teamid)::text) AND (b.yearid = t.yearid) AND ((b.lgid)::text = (t.lgid)::text))
13	-> Seq Scan on batting b (cost=0.00..2508.90 rows=97890 width=20) (actual time=0.015..13.811 rows=97890 loops=1)
14	-> Hash (cost=118.50..118.50 rows=36 width=10) (actual time=1.298..1.298 rows=36 loops=1)
15	Buckets: 1024 Batches: 1 Memory Usage: 10kB
16	-> Seq Scan on teams t (cost=0.00..118.50 rows=36 width=10) (actual time=0.784..1.261 rows=36 loops=1)
17	Filter: ((name)::text ~~ '%Montreal Expos%'::text)
18	Rows Removed by Filter: 2725
19	-> Index Scan using batting_pkey on batting b_1 (cost=0.42..0.52 rows=1 width=20) (actual time=0.007..0.007 rows=1 loops=1479)
20	Index Cond: (((masterid)::text = (b.masterid)::text) AND (yearid = (b.yearid + 1)))

What you did to optimize the query:

1. **create index on batting(teamid,yearid,lgid);**
   create index on teams(lgid);

The cost re-evaluates after optimization:

	QUERY PLAN text
1	Unique (cost=581.26..581.27 rows=1 width=12) (actual time=26.143..26.268 rows=193 loops=1)
2	-> Sort (cost=581.26..581.27 rows=1 width=12) (actual time=26.135..26.160 rows=483 loops=1)
3	Sort Key: master.namelast, master.namefirst
4	Sort Method: quicksort Memory: 48kB
5	-> Nested Loop (cost=424.55..581.25 rows=1 width=12) (actual time=6.787..25.525 rows=483 loops=1)
6	-> Nested Loop (cost=424.26..580.93 rows=1 width=27) (actual time=6.736..22.239 rows=483 loops=1)
7	-> Nested Loop (cost=423.98..580.62 rows=1 width=42) (actual time=6.702..19.362 rows=770 loops=1)
8	Join Filter: ((b.masterid)::text = (b_2.masterid)::text)
9	-> Nested Loop (cost=423.56..580.08 rows=1 width=22) (actual time=6.677..16.353 rows=804 loops=1)
10	Join Filter: (((t_1.teamid)::text = (b_1.teamid)::text) AND (t_1.yearid = b_1.yearid) AND ((t_1.lgid)::text = (b_1.lgid)::text))
11	Rows Removed by Join Filter: 415
12	-> Hash Join (cost=423.14..545.53 rows=64 width=23) (actual time=6.623..7.647 rows=1433 loops=1)
13	Hash Cond: (t_1.yearid = (b.yearid + 1))
14	-> Seq Scan on teams t_1 (cost=0.00..118.51 rows=36 width=10) (actual time=0.828..1.468 rows=36 loops=1)
15	Filter: ((name)::text ~~ '%Montreal Expos%'::text)
16	Rows Removed by Filter: 2725
17	-> Hash (cost=418.71..418.71 rows=354 width=13) (actual time=5.693..5.694 rows=1479 loops=1)
18	Buckets: 2048 (originally 1024) Batches: 1 (originally 1) Memory Usage: 83kB
19	-> Nested Loop (cost=0.42..418.71 rows=354 width=13) (actual time=0.856..5.069 rows=1479 loops=1)
20	-> Seq Scan on teams t (cost=0.00..118.51 rows=36 width=10) (actual time=0.808..1.633 rows=36 loops=1)

9. The cost of evaluating the query:

	QUERY PLAN text
1	Unique (cost=5324.74..5324.76 rows=2 width=36) (actual time=251.984..252.009 rows=44 loops=1)
2	-> Sort (cost=5324.74..5324.74 rows=2 width=36) (actual time=251.983..251.989 rows=75 loops=1)
3	Sort Key: jeter.masterid, jetert.masterid, jetertt.masterid
4	Sort Method: quicksort Memory: 30kB
5	-> Gather (cost=1677.59..5324.73 rows=2 width=36) (actual time=68.616..253.142 rows=75 loops=1)
6	Workers Planned: 1
7	Workers Launched: 1
8	-> Nested Loop (cost=677.59..4324.53 rows=1 width=36) (actual time=97.239..237.471 rows=38 loops=2)
9	Join Filter: ((jetertt.masterid)::text <> (jeter.masterid)::text)
10	-> Nested Loop (cost=677.17..4322.80 rows=1 width=38) (actual time=97.217..237.105 rows=14 loops=2)
11	Join Filter: ((jeter.teamid)::text <> (jeterty.teamid)::text)
12	Rows Removed by Join Filter: 4
13	-> Nested Loop (cost=676.76..2847.18 rows=2 width=22) (actual time=27.705..32.151 rows=27 loops=2)
14	-> Hash Join (cost=676.34..2845.25 rows=3 width=20) (actual time=27.633..31.593 rows=10 loops=2)
15	Hash Cond: ((jeter.masterid)::text = (m.masterid)::text)
16	-> Parallel Seq Scan on appearances jeter (cost=0.00..2017.33 rows=57733 width=20) (actual time=0.015..12.720 rows=49073 loops=2)
17	-> Hash (cost=676.33..676.33 rows=1 width=9) (actual time=8.770..8.770 rows=1 loops=2)
18	Buckets: 1024 Batches: 1 Memory Usage: 9kB
19	-> Seq Scan on master m (cost=0.00..676.33 rows=1 width=9) (actual time=4.582..8.752 rows=1 loops=2)
20	Filter: (((namelast)::text = 'Ieter'::text) AND ((namefirst)::text = 'Derek'::text))

What you did to optimize the query:

1. **create index on appearances(masterID, teamID,lgID, G_1b, G_2b, G_3b);**
   create index on master(nameLast, nameFirst);

The cost re-evaluates after optimization:

	QUERY PLAN text
1	Unique (cost=53.26..53.28 rows=2 width=36) (actual time=5.918..5.940 rows=44 loops=1)
2	-> Sort (cost=53.26..53.27 rows=2 width=36) (actual time=5.917..5.921 rows=75 loops=1)
3	Sort Key: jeter.masterid, jetert.masterid, jetertt.masterid
4	Sort Method: quicksort Memory: 30kB
5	-> Nested Loop (cost=6.01..53.25 rows=2 width=36) (actual time=0.972..5.610 rows=75 loops=1)
6	Join Filter: ((jetertt.masterid)::text <> (jeter.masterid)::text)
7	-> Nested Loop (cost=5.59..49.79 rows=2 width=38) (actual time=0.905..4.198 rows=27 loops=1)
8	Join Filter: ((jeter.teamid)::text <> (jeterty.teamid)::text)
9	Rows Removed by Join Filter: 8
10	-> Nested Loop (cost=5.18..42.69 rows=4 width=22) (actual time=0.397..1.805 rows=54 loops=1)
11	-> Nested Loop (cost=4.76..39.47 rows=5 width=20) (actual time=0.357..0.584 rows=19 loops=1)
12	-> Index Scan using num_two on master m (cost=0.29..8.31 rows=1 width=9) (actual time=0.226..0.226 rows=1 loops=1)
13	Index Cond: (((namelast)::text = 'Jeter'::text) AND ((namefirst)::text = 'Derek'::text))
14	-> Bitmap Heap Scan on appearances jeter (cost=4.47..31.10 rows=7 width=20) (actual time=0.107..0.305 rows=19 loops=1)
15	Recheck Cond: ((masterid)::text = (m.masterid)::text)
16	Heap Blocks: exact=19
17	-> Bitmap Index Scan on num_one (cost=0.00..4.47 rows=7 width=0) (actual time=0.084..0.084 rows=19 loops=1)
18	Index Cond: ((masterid)::text = (m.masterid)::text)
19	-> Index Scan using appearances_pkey on appearances jetert (cost=0.42..0.63 rows=1 width=20) (actual time=0.034..0.057 rows=3 loops=19)
20	Index Cond: ((yearid = jeter.yearid) AND ((teamid)::text = (jeter.teamid)::text))

10. The cost of evaluating the query:

	QUERY PLAN text
1	Seq Scan on teams t (cost=0.00..106657.66 rows=14 width=25) (actual time=0.180..35.689 rows=154 loops=1)
2	Filter: (w = (SubPlan 1))
3	Rows Removed by Filter: 2607
4	SubPlan 1
5	-> Aggregate (cost=38.58..38.59 rows=1 width=4) (actual time=0.011..0.011 rows=1 loops=2761)
6	-> Index Scan using teams_pkey on teams y (cost=0.28..38.53 rows=19 width=4) (actual time=0.003..0.007 rows=22 loops=2761)
7	Index Cond: (t.yearid = yearid)
8	Planning Time: 0.861 ms
9	Execution Time: 35.839 ms

What you did to optimize the query:

1. **create index on teams(yearid,w,name);**

The cost re-evaluates after optimization:

	QUERY PLAN text
1	Seq Scan on teams t (cost=0.00..6484.38 rows=14 width=25) (actual time=0.260..19.312 rows=154 loops=1)
2	Filter: (w = (SubPlan 2))
3	Rows Removed by Filter: 2607
4	SubPlan 2
5	-> Result (cost=2.30..2.31 rows=1 width=4) (actual time=0.006..0.006 rows=1 loops=2761)
6	InitPlan 1 (returns $1)
7	-> Limit (cost=0.28..2.30 rows=1 width=4) (actual time=0.005..0.005 rows=1 loops=2761)
8	-> Index Only Scan Backward using teams_yearid_w_name_idx on teams y (cost=0.28..38.58 rows=19 width=4) (actual time=0....
9	Index Cond: ((yearid = t.yearid) AND (w IS NOT NULL))
10	Heap Fetches: 2761
11	Planning Time: 0.966 ms
12	Execution Time: 19.411 ms