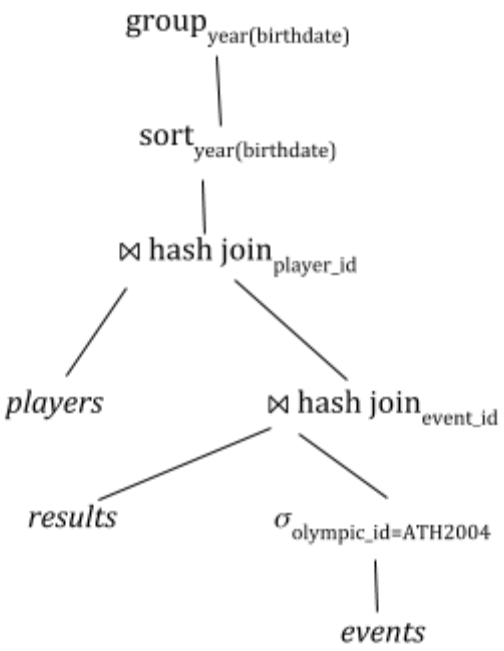
Answer to Question 1

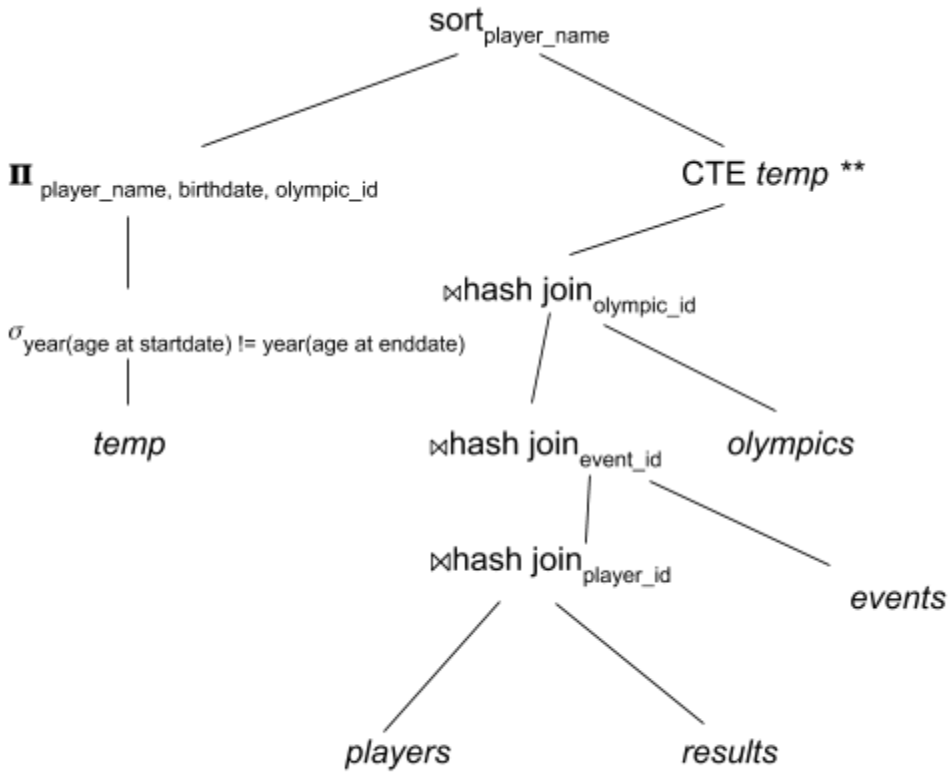
First, you need to remove the space between Chris and % to capture names like Christine. The second issue is including the name check in the join clause. The left outer join results in all records in p being included regardless of whether or not they have a corresponding value in results. By moving the name check to a where clause, it gets all the records from results and then filters out with Chris%.

```
select p.player_id, p.name, count(r.medal)
from players p
left outer join results r on (p.player_id = r.player_id and r.medal = 'GOLD')
where p.name like 'Chris%'
group by p.player_id, p.name
order by p.player_id;
```

Answer to Question 2



Answer to Question 3



Everything in the subtree marked by ** denotes operators responsible for creating temp. The rest of the tree pertains to the final answer given by the query.

Answer to Question 4

Both aggregate functions correctly determined there should only be one final row. The differences occur within the nested loop in the query plan. First, lets compare the seq scan on s. Both filter based on the condition T.C = x where x is the integer. In query 1, x is 1, and there is only 1 row in s where c = 1. So the query plan is correct. Then in query 2, x is 48, and there are 40 rows in s where c = 48. So the query plan is correct again. And then, we can look at the seq scan on r, which correctly gets all 100000 rows from r in both queries. Then the hash join that combines r.b = s.b. For query 1, there are no rows where s.b = r.b and s.c = 1, so query plan correctly identifies that there are 0 rows and never executes the seq scan on t because there is no need to. For query 2, there are 7647 rows in r that correspond to a row in s that has s.c = 48. So query two correctly identifies that, and then executes the seq scan on t. There is only one row in t where c = 48, so that row then corresponds to all the rows

found by the hash join on s and r and the query plan correctly identifies the 7647 rows found.

Generally the query plan approximated the sizes of intermediate results perfectly, although whether that would hold for larger datasets I am unsure. Differences in cardinalities for the two queries come from the T.C value because there is only one value in s that s.c = t.c = 1, and forty where s.c = t.c = 48. And then when joining with r, there aren't values in r that correspond to s.b for s.c = 1, while there are many for s.c = 48.