QUERY 3 – INTERVENÇÃO NO BANCO DE DADOS

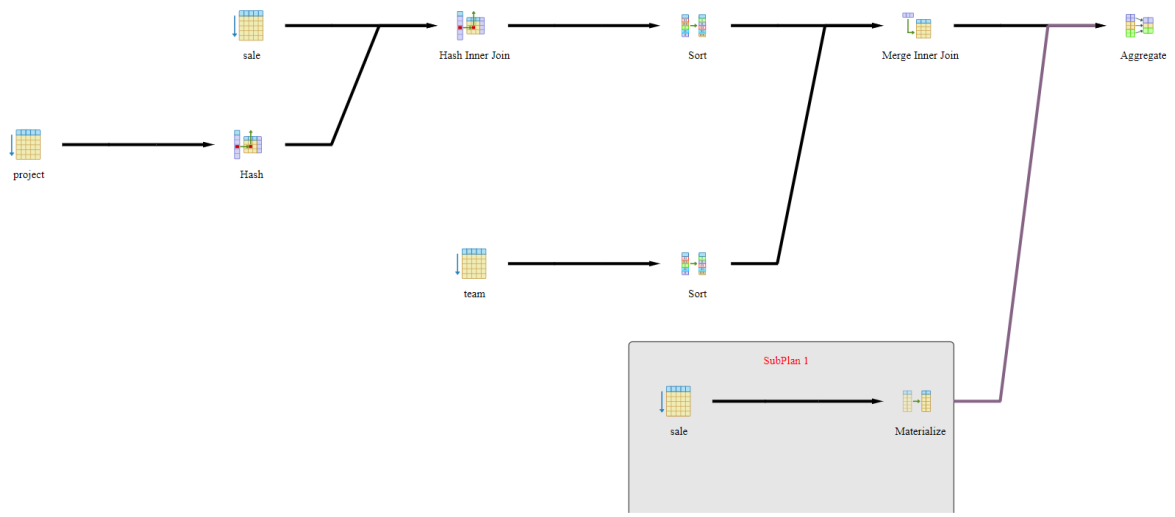
```sql
/* depois: Query 3 seleciona o time que obteve a maior venda de projeto;
*   Decidimos manter com o objetivo principal da consulta, e, apenas adequamos ao novo modelo, substituindo o
* "ORDER BY" pelo "GROUP BY" e também adicionando o HAVING MAX
*
*/
SELECT T.name_team, P.name_project, MAX(S.amount_sale_project)
FROM team as T
NATURAL JOIN sale as S
NATURAL JOIN project as P
WHERE T.idt_project = P.idt_project AND P.idt_project = S.idt_project
GROUP BY T.name_team, P.name_project
HAVING MAX(S.amount_sale_project)>=ALL(
    SELECT amount_sale_project
    FROM sale
    );
```

Consulta

Plano de Consulta:

	QUERY PLAN text
1	HashAggregate (cost=375.53..101397.53 rows=3390 width=96)
2	Group Key: t.name_team, p.name_project
3	Filter: (SubPlan 1)
4	-> Merge Join (cost=200.38..307.73 rows=6780 width=96)
5	Merge Cond: (s.idt_project = t.idt_project)
6	-> Sort (cost=117.01..119.83 rows=1130 width=72)
7	Sort Key: s.idt_project
8	-> Hash Join (cost=35.42..59.70 rows=1130 width=72)
9	Hash Cond: (s.idt_project = p.idt_project)
10	-> Seq Scan on sale s (cost=0.00..21.30 rows=1130 width=36)
11	-> Hash (cost=21.30..21.30 rows=1130 width=36)
12	-> Seq Scan on project p (cost=0.00..21.30 rows=1130 width=36)
13	-> Sort (cost=83.37..86.37 rows=1200 width=36)
14	Sort Key: t.idt_project
15	-> Seq Scan on team t (cost=0.00..22.00 rows=1200 width=36)
16	SubPlan 1
17	-> Materialize (cost=0.00..26.95 rows=1130 width=32)
18	-> Seq Scan on sale (cost=0.00..21.30 rows=1130 width=32)
19	JIT:
20	Functions: 29
21	Options: Inlining false, Optimization false, Expressions true, Deforming true

Árvore de Consulta:



QUERY 3: CRIAÇÃO DE ÍNDICES

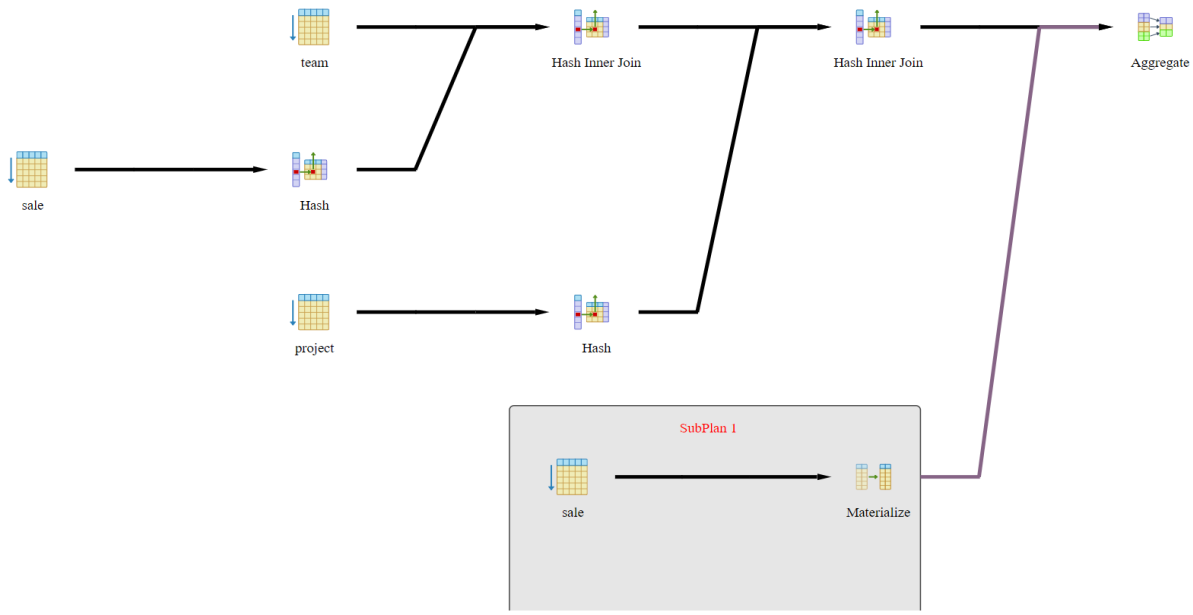
```
CREATE INDEX indiceTerceiraQuery
 ON sale (amount_sale_project);

 CREATE INDEX indiceTerceiraQueryProject
 ON project (idt_project);
```

Relatório do plano de consulta a partir do comando explain do Postgress:

```
1  HashAggregate  (cost=29.40..46.09 rows=15 width=96)
2    Group Key: t.name_team, p.name_project
3    Filter: (SubPlan 1)
4    ->  Hash Join  (cost=2.23..29.10 rows=30 width=96)
5          Hash Cond: (t.idt_project = p.idt_project)
6          ->  Hash Join  (cost=1.11..27.91 rows=30 width=72)
7                Hash Cond: (t.idt_project = s.idt_project)
8                ->  Seq Scan on team t  (cost=0.00..22.00 rows=1200 width=36)
9                ->  Hash  (cost=1.05..1.05 rows=5 width=36)
10                     ->  Seq Scan on sale s  (cost=0.00..1.05 rows=5 width=36)
11         ->  Hash  (cost=1.05..1.05 rows=5 width=36)
12               ->  Seq Scan on project p  (cost=0.00..1.05 rows=5 width=36)
13   SubPlan 1
14     ->  Materialize  (cost=0.00..1.07 rows=5 width=32)
15           ->  Seq Scan on sale  (cost=0.00..1.05 rows=5 width=32)
```

Árvore do plano de consulta a partir do comando explain do Postgress:



Comparação antes e depois da alteração

A consulta realizada seleciona o nome de um time, nome do projeto ligado a esse time e o seu valor de venda, retornando então o time com maior valor de venda de seu projeto. Analisando o relatório antes da implementação dos index vemos através do comando explain a progressão da consulta, inicialmente temos um grande custo no HashAggregate seguido de vários passos diferenciados em Merges, Sorts e Hashes que apesar de ter valores menores que a primeira linha ainda assim acumulam seu valor, deixando dúvidas sobre a efetividade dessa consulta em ambientes que tenham muitos times e valores de vendas, por mais que não tenha o maior valor de custo entre as queries.

Entretanto, ainda é necessária acrescentar índices, que resultarão em grandes diferenças nos valores discutidos, além de potencialmente diminuir os números de passos e dar margem a execuções menos custosas, por isso foram criados dois índices, “indiceTerceiraQuery” que “indexa” o atributo amount_sale_project na table sale,“indiceTerceiraQueryProject” que “indexa” o atributo idt_project na table project, nos levando ao segundo resultado, resultado este que diminui em grande quantidade o custo e usa quase que em sua totalidade hashes de baixo custo, diminuindo o número de passos, mais uma vez indicando a importância desta análise.

Por fim, conclui-se que desta maneira não haverá problemas sérios em eventuais casos de consultas em tabelas com informações que se alongam por muito mais do que o analisado.