**RELATÓRIO DE ANALISE**

**TUNNING E PERFORMANCE.**

**CONSULTA ANALISADA – QUERY3.SQL**

|  |
| --- |
|  |

Execução inicial em 1 secs 349 msec para 28 linhas afetadas;

Analisando o plano de execução:

|  |
| --- |
|  |

**LAUDO -** Essa consulta pede os dados de um intervalo mais específico, no caso, a data de ontem no filtro. Além disso temos várias operações distintas como:

1. Quantos bets usuário fez (COUNT(\*));
2. O total apostado (SUM(b.amount));
3. O valor médio de bets(AVG(b.amount));
4. Considerar apenas usuários com 5 ou mais apostas (HAVING COUNT(\*) >= 5), do modo que está codificado, é aplicada só depois da agregação total, ou seja, é feito em todos os dados;
5. Os 20 maiores apostadores (ORDER BY total\_wagered DESC LIMIT 20) e com isso, somente após a agregação, o PostgreSQL irá ordenar todos os usuários pelo total apostado para pegar os top 20;

Identificamos um **PARALLEL SEQ SCAN (TABLE SCAN)**, ou seja, a tabela está sendo lida por inteiro sem nenhum tipo de índice.

**SUGESTÃO DE MELHORIAS**

Com essas informações podemos sugerir a criação de um índice composto.

**CREATE INDEX IDX\_BETS\_DAY\_USER ON BETS(PLACED\_AT, USER\_ID);**

Há uma sugestão também de mudarmos a consulta para a melhoria do item 4 e 5 do laudo.

|  |
| --- |
|  |

Deste modo, o filtro do HAVING roda antes da agregação principal, reduzindo o volume de reads

E com o indice abaixo, forçamos a organização dos maiores apostadores.

**CREATE INDEX IDX\_BETS\_DAY\_USER\_AMOUNT ON BETS(PLACED\_AT, USER\_ID, AMOUNT);**

Com isso a ordenação se torna orgânica.

**APÓS A APLICAÇÃO DAS SUGESTÕES**

Com a aplicação das sugestões, o plano de execução trouxe:

|  |
| --- |
|  |

* Filtragem eficiente com índice composto (**IDX\_BETS\_DAY\_USER\_AMOUNT**) reduzindo o volume de linhas processadas antes do agrupamento, eliminando milhões de apostas fora do dia de interesse e evitando leituras desnecessárias da tabela.
* Renderização de **INDEX ONLY SCAN**, que faz com que o banco responda apenas com o índice, sem precisar acessar o **HEAP (HEAP FETCHES: 0)**, e assim, acelerando ainda mais a leitura.
* A redução de cerca de **50%** no tempo total mostra que a combinação de índices, filtragem adequada, e uso de paralelismo foi bastante efetiva.

Concluímos que as melhorias aplicadas reduziram significativamente o tempo de execução, otimizando o uso de índices e filtragem prévia, além de melhorar o aproveitamento dos recursos do banco (índices, paralelismo e batches).