FATEC SÃO JOSÉ DOS CAMPOS Nome: Leonardo Adler da Silva

Otimização de Sistemas de Banco de Dados

Laboratório 04

Análise de Desempenho: Prepared Statements vs Concatenação

Objetivo do Laboratório

Reproduzir em laboratório a teoria apresentada em sala de aula quanto ao *parse* de comandos e custo gerado no servidor.

O laboratório consiste na implementação de um código-fonte acessando o banco de dados via um driver capaz de processar **prepared statements** (exemplo: Java e JDBC). O aluno deve elaborar duas versões do mesmo código, alternando o modo de acesso aos dados conforme explicado em aula (*hardcoded* e *softcoded*).

O processamento consiste em executar 5**.000 vezes ou mais** um mesmo SELECT numa tabela criada pelo aluno. Para cada modo de acesso, o aluno deve computar o tempo gasto para processar o total dos comandos. Ao final, deve-se montar um gráfico comparativo entre as duas versões implementadas e demonstrar a diferença entre elas.

Análise dos Resultados

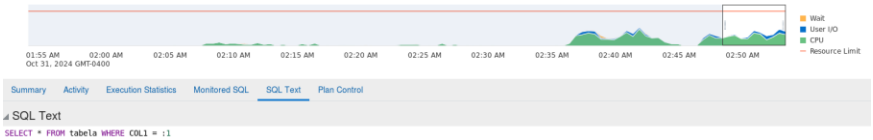
Os testes realizados mostraram um desempenho superior ao utilizar **Prepared Statements** em relação ao método de concatenação de strings. Os tempos de execução das consultas foram os seguintes:

- **Tempo de Execução com Hardcoded Query (Concatenação)**: **213573 ms**
- **Tempo de Execução com Prepared Statement**: **187738 ms**

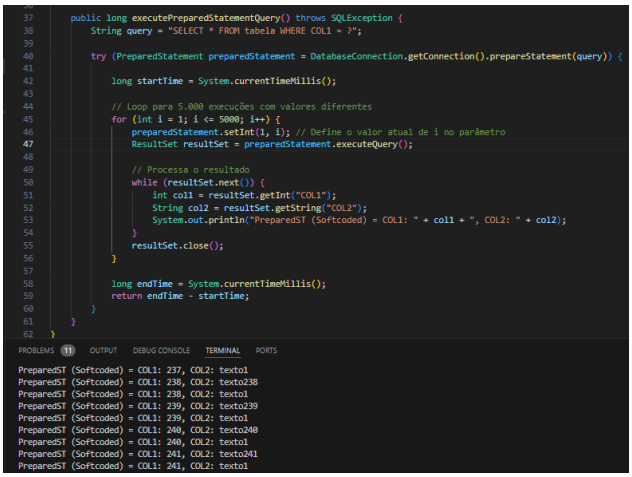
Interpretação dos Resultados

1. Otimização do Cache de Consultas

Quando um **Prepared Statement** é utilizado, o SGBD armazena o formato da consulta em cache, permitindo que a etapa de *parsing* da query seja pulada em execuções subsequentes. Apenas as variáveis das cláusulas `WHERE` são processadas, resultando em uma execução mais rápida.

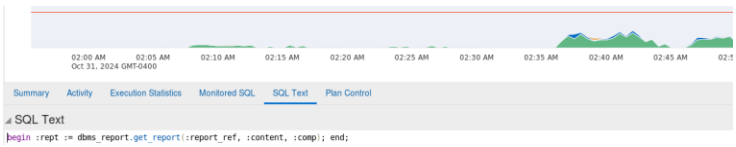


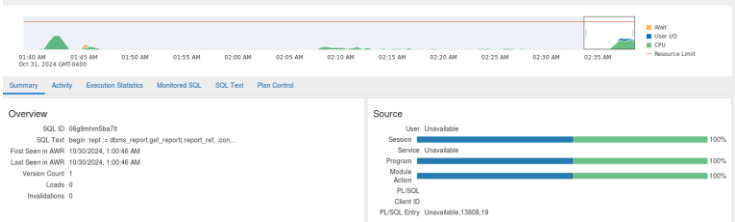
Codigo utilizado (**Prepared Statement**):



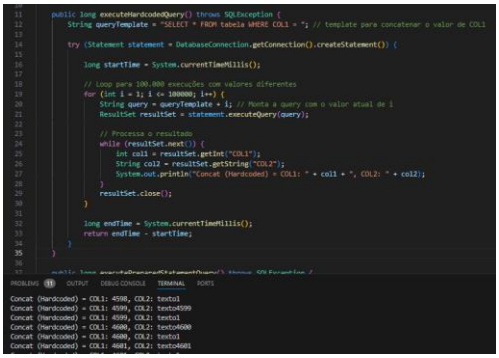
2. Processamento de Consultas Concatenadas

No método de concatenação, o SGBD não consegue identificar o formato da consulta previamente, o que o obriga a realizar o *parsing* da query toda vez que ela é executada. Isso contribui para um aumento significativo no tempo de execução, como evidenciado nos resultados.





Codigo utilizado (concatenacao):



3. Benefícios Adicionais do Prepared Statement:

Além da melhoria no desempenho, o uso de **Prepared Statements** também oferece um nível de segurança superior contra SQL injection, protegendo a aplicação contra ataques maliciosos.

4. Importância do prepared statement em Ambientes Corporativos

Em ambientes corporativos que utilizam o SGBD Oracle, a eficiência do **Prepared Statement** pode resultar em melhorias de desempenho de 100% ou mais, dependendo da complexidade da consulta. Embora todos os SGBDs se beneficiem dessa abordagem, a intensidade da melhoria varia.

Conclusão

Este teste simples, que envolveu apenas um parâmetro, demonstrou claramente a vantagem dos **Prepared Statements** em termos de desempenho e segurança. Com a

evidência coletada, é evidente que não há motivos para implementar consultas SQL sem o uso de **Prepared Statements**, preferindo sempre evitar a concatenação de strings.

- **API REST criada usando Spring Boot**:

```java
@RestController
@RequestMapping("/test-performance")
public class PerformanceTestController {

    private final OracleQueryExecutor executor;

    public PerformanceTestController() {
        this.executor = new OracleQueryExecutor();
    }

    @GetMapping
    public Map<String, Long> testPerformance() {

        Map<String, Long> result = new HashMap<>();

        try {
            // Executa a query hardcoded e armazena o tempo de execução
            long hardcodedTime = executor.executeHardcodedQuery();
            result.put("hardcodedQueryTime", hardcodedTime);

            // Executa a query com prepared statement e armazena o tempo de execução
            long preparedStatementTime = executor.executePreparedStatementQuery();
            result.put("preparedStatementQueryTime", preparedStatementTime);

        } catch (SQLException e) {
            e.printStackTrace();
            result.put("error", -1L); // Indica erro
        }

        return result;
    }
```

Retorno da API

Os resultados dos testes podem ser representados na seguinte estrutura JSON:

```
{
    "hardcodedQueryTime": 213573,
    "preparedStatementQueryTime": 187738
}
```