Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

OTIMIZAÇÃO DE CONSULTAS SQL COM FUNÇÕES ANALÍTICAS NO BANCO DE DADOS ORACLE

Douglas Rodrigues Garcia[1]

Claudio Eduardo Paiva[2]

Resumo

Bancos de dados foram criados diante da necessidade de guardar informações para utilização futura. Este tipo de sistema computacional para armazenamento tem vivido um expressivo crescimento desde a década de 1990, passando por adequações nas suas estruturas e por melhorias nos seus processos de gerenciamento, sobretudo no que diz respeito aos métodos para consulta de dados. A reutilização das informações guardadas tem se tornado essencial em diversos setores nos dias atuais e isto tem direcionado pesquisas na busca de novas e melhores técnicas. A escrita de códigos que executem com bom desempenho é uma constante preocupação dos profissionais da área, visto que obter respostas com agilidade pode ser decisivo em muitas situações. Assim, este trabalho é um estudo aplicado que dispõe, a partir de um levantamento bibliográfico, apresentar alternativas de melhorias para realização de consultas no banco de dados Oracle mediante o uso de suas funções analíticas. Visa também o entendimento de como estas funções podem auxiliar na criação de comandos de manipulação de dados, diminuir a complexidade de escrita e favorecer a sua legibilidade. Testes comparativos serão realizados em cenários com e sem uso das funções analíticas e os resultados obtidos serão confrontados para conhecer vantagens e possibilidades do uso desta técnica.

Palavras-chave: Desempenho. Manipulação de dados. Testes comparativos.

Abstract

Databases were created on the need to store information for future use. This type of computational system for storage has experienced an expressive growth since the 1990s, through adaptations in its structures and through improvements in its management processes, especially with regard to data query methods. The re-use of stored information has become essential in several sectors in current days and this has directed research in the search for new and better techniques. The writing of codes that execute with good performance is a constant concern of the professionals of the area, since to obtain answers with agility can be decisive in many situations. Thus, this paper is an applied study that has to present alternatives of improvements to perform queries in the Oracle database through the use of its analytical functions from a bibliographical survey. It also aims at understanding how these functions can help the creation of commands to manipulate data, reduce the complexity of writing and favor its readability. Comparative tests will be performed in scenarios with and

[1] Graduando em Análise e Desenvolvimento de sistemas pela Fatec Dr Thomaz Novelino – Franca/SP. Endereço eletrônico: dougrgarcia92@gmail.com

[2] Docente do curso de Análise e Desenvolvimento de Sistemas pela Fatec Dr Thomaz Novelino – Franca/SP. Endereço eletrônico: claudio.paiva01@fatec.sp.gov.br

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

without the use of analytical functions and the results obtained will be confronted to know the advantages and possibilities of using this technique.

Keywords: *Performance. Data manipulation. Comparative tests.*

1 Introdução

A importância da análise das informações contidas nos bancos de dados intervém diretamente nas pesquisas para produção de métodos mais eficientes de consulta e recuperação nestes sistemas. Entender a razão de se tirar proveito de dados guardados revela o valor da reutilização da informação e, por isto, desde a sua criação, passando pelo grande aumento de uso nos anos 1990, até os dias de hoje, bancos de dados estão sendo cada vez mais empregados não apenas como repositórios, mas como fontes para geração de conhecimento.

Dada a necessidade de se construir consultas que operem com agilidade, há de se considerar ainda a relevância da criação de códigos SQL de fácil entendimento, escrita e manutenção. Neste sentido, as funções analíticas do Oracle oferecem meios alternativos para concepção de *scripts* menores, portanto de compreensões mais fáceis, e que apresentam respostas com bom desempenho. É indispensável a execução de teste que usem tais funções em busca de códigos mais performáticos.

Este trabalho é um estudo aplicado que tem como objetivo, a partir de um levantamento bibliográfico, avaliar o uso das funções analíticas como opção para construção de melhores comandos SQL para consultar no banco de dados Oracle. Propõe a análise da complexidade do código e outras grandezas da resposta, como por exemplo, mas não se limitando apenas, ao estudo do tempo para recuperação dos dados.

Foram criados cenários para a aplicação de testes envolvendo consultas com utilização de funções analíticas e sem uso delas. Os resultados coletados foram comparados e confrontados a fim de apontar vantagens e oportunidades de uso deste recurso no banco de dados estudado.

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

2 Conceitos fundamentais

Esta seção mostra os conceitos fundamentais sobre os quais este trabalho está baseado. Apresenta conteúdos sobre bancos de dados e seus tipos, relata a preocupação com a otimização de consultas para melhoria da performance e apresenta o que são as funções de agregação e as funções analíticas do Oracle.

2.1 Banco de dados e banco de dados relacional

Presente nas empresas desde a década de 1960, os bancos de dados (BD) têm crescido desde sua criação (SILBERSCHATZ et al., 2016). Segundo os autores, eles surgiram da grande necessidade de guardar informações para utilização futura e tiveram um aumento de uso significativo na década de 1990, graças à revolução causada pela Internet nas empresas, quando muitos dos sistemas empresariais começaram a disponibilizar informações em aplicações *Web*.

Esta carência de consumo de informações fez com que os BDs fossem projetados para armazenar grandes quantidades de dados e permitir novos processamentos e reutilizações (NASCIMENTO, 2015).

Segundo Date (2004), um BD é um sistema que administra de forma computadorizada uma coleção de registros e, por si só, pode ser considerado como um repositório para um conjunto de arquivos de dados onde os usuários podem realizar operações de seu interesse.

Um BD representa alguma questão do mundo real por meio de uma coleção lógica e coerente de dados com significado pertinente, que é projetado, produzido e populado com dados com um propósito comum e específico (NAVATHE; ELMASRI, 2005).

Silberschatz *et al.* (2016) concordam com a ideia de que os BDs podem ser entendidos como uma coleção de dados que armazena informações empresariais importantes. Em sua obra, os autores apresentam as muitas maneiras de estruturação de tais coleções e trata as principais propostas de organização feitas ao longo do tempo, conhecidas como modelo de dados.

Um modelo de dados define os objetos necessários para a criação da estrutura de armazenamento de dados, ou seja, define a estrutura de dados do BD (DATE, 2004). Muitos destes modelos de dados foram propostos desde a criação

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

dos primeiros BDs e os principais são os modelos de dados hierárquicos, em rede, orientados a objetos e relacional.

Um BD projetado sob uma estrutura relacional, ou simplesmente banco de dados relacional (BDR), armazena os dados e suas relações em estruturas baseadas em tabelas, compostas por linhas e colunas inter-relacionadas (SILBERSCHATZ *et al.*, 2016).

2.2 Consultas em banco de dados

A necessidade de reutilização de informações é um grande propulsor das melhorias implementadas nas tecnologias de BDs. As operações realizadas para leitura dos dados armazenados possuem importância equivalente àquelas que tratam do seu armazenamento e métodos de acesso, e a possiblidade de fazer uma nova leitura dos dados guardados, gerando reaproveitamento, agrega valor às estruturas de registro de informações.

À medida em que as tecnologias de *hardware* avançam, maiores quantidades de dados são registradas nos BDs, contudo, se não for possível consultar e analisar de maneira adequada estes dados, eles perdem sua utilidade e suas bases de dados podem ser consideradas desnecessárias (KEIM; SIPS; ANKERST, 2005). Segundo os autores, considerando-se os desafios encontrados na busca de bons recursos para a realização de consultas de dados e a grande importância deste tipo de operação, frequentemente, novas pesquisas são realizadas a fim de descobrir melhores técnicas para recuperação dos dados coletados.

Em um BDR, uma consulta é escrita em uma linguagem de alto nível e seu processamento acontece em várias etapas, envolvendo atividades especiais em cada fase. Tais estágios incluem funcionalidades para análise da sintaxe utilizada, escolha da melhor estratégia de execução pelo otimizador, criação de planos de execução e, por fim, a efetivação deste plano, obtendo-se o resultado (NAVATHE; ELMASRI, 2005).

Enquanto um BD pode ser considerado como uma estrutura computacional para armazenamento de dados integrados, as operações realizadas sobre estes dados, como escrita, exclusão, atualização e consulta, são ações de responsabilidade do sistema gerenciador de banco de dados (SGBD) (DATE, 2004). O autor explica que este é um *software* de gerenciamento de todas as operações

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

executadas no BD, como as ações necessárias para garantir a integridade dos dados, sua segurança, controle de acesso e falhas e também as consultas aos dados guardados.

A oferta de um ambiente eficiente de armazenamento e recuperação de dados é um dos principais objetivos de um SGBD. Tal ambiente pode ser dividido em subsistemas de acordo com cada tarefa a ser executada, como o gerenciador de armazenamento e o processador de consultas (SILBERSCHATZ *et al.*, 2016).

Assim, para recuperar dados em um BD, uma consulta deve ser escrita em linguagem própria para que possa ser interpretada e executada pelo SGDB e um exemplo de linguagem de alto nível usada para comunicação e manipulação dos dados adotada por muitos dos BDs é a linguagem *Structured Query Language* (SQL). Esta linguagem permite escrever comandos para execução de consultas simples, que retornam os registros de uma única tabela, e também comandos para pesquisas mais complexas, envolvendo várias tabelas.

Na década de 1970, nos laboratórios da IBM foi criada uma linguagem para consulta em BDRs conhecida como *Structured English Query Language* (SEQUEL) e que teve seu nome alterado para SQL no início dos anos 1980 (NAVATHE; ELMASRI, 2005). Em uma iniciativa dos órgãos American National Standards Institute (ANSI) e a International Standards Organization (ISO), a linguagem se tornou um padrão para consultas em BDR e sua primeira versão padronizada foi lançada em 1986.

Esta linguagem tornou possível que os usuários fizessem requisições de dados por meio de instruções SQL de forma simplificada, já que, para este tipo de operação em BDs, não é preciso dizer como será feito, mas sim, informar o que deve ser feito. Atualmente, a maioria dos sistemas comerciais de BDRs utiliza a linguagem SQL como linguagem padrão para manipulação de dados (SILBERSCHATZ *et al.*, 2016).

2.2.1 Processo de otimização de consultas

Diante da grande importância que as tarefas de recuperação de informações em um BD possuem, explicam-se a relevância de estudos na busca de métodos de trabalho cada vez mais eficientes e da preocupação com a boa escrita de comandos para consulta de dados (DATE, 2004).

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

Em se tratando do uso de métodos capazes de oferecer ótimos resultados em consultas em BD, para Silberschatz *et al.* (2006), a otimização de consulta está relacionada com a escolha, pelo processador de consulta do SGBD, do melhor plano de execução dentre as muitas estratégias de processamento geradas.

Por outro lado, para que o código de uma consulta SQL seja otimizado é preciso que o administrador de BD analise, além das estruturas das tabelas e seus índices, as instruções usadas. Ao construir tais comandos testando alternativas, é possível medir e comparar resultados até que seja criado um *script* de qualidade e que execute em tempo ideal (MACEDO, 2013). Assim, otimizar uma consulta SQL também envolve escrever, testar comandos e escolher os melhores códigos, a fim de tornar estas execuções mais performáticas.

Um estudo sobre otimização de consultas em BDRs pode ser encontrado em Nascimento (2015), onde o autor fez uso de índices para aferir o desempenho em consultas em um ambiente controlado, destacando a importância da aplicação deste recurso para melhorar o desempenho de consultas de dados.

Oliveira e Paiva (2018) apresentaram um estudo sobre *tuning* de BDs e citam que, de acordo com sua pesquisa, grande parte dos problemas de desempenho em consultas em BDs está relacionada às instruções SQL mal escritas, à falta ou criação indevida de índices e às estruturas inadequadas dos BDs. Os autores destacaram a importância da criação de instruções SQL bem ajustadas.

2.3 Funções de agregação e funções analíticas no Oracle

Algumas vezes, as consultas em um BD são feitas com o intuito de se fazer análise de dados e consideram certos tipos de agrupamentos e sumarizações de valores para gerar informações estatísticas como resultado (HOKAMA *et al.*, 2004).

Em geral, em um BD é usado o nível mais baixo de granularidade dos dados para o seu armazenamento, contudo, algumas consultas exigem a sumarização (ou agrupamento ou agregação) dos dados para melhor entendimento ou para responder a questões específicas (BEAULIEU, 2010).

De acordo com Puga *et al.* (2013), as funções de agregação são capazes de manipular vários registros de uma vez, podendo envolver todos os registros da tabela e fornecer um único resultado, ou agrupar colunas com uso do operador *GROUP BY*.

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

A linguagem SQL padrão implementa cinco funções de agregação de valores: *COUNT* ( ) para contar a quantidade de registros em uma tabela; *SUM* ( ) para retornar a soma dos valores de uma coluna; *MIN*( ) para retornar o valor mínimo especificado em uma coluna; *MAX* ( ) para selecionar o valor máximo contido em uma coluna e a função *AVG* ( ) para calcular e retornar o valor médio de uma coluna (HOKAMA *et al.*, 2004).

Apesar da importância deste tipo de função para responder consultas sumarizadas, algumas questões analíticas específicas sobre os dados estão além do poder de solução oferecido por elas. A análise de dados pode apresentar maior complexidade e exceder o que as funções implementadas com o SQL padrão podem tratar (GRAY, 1997).

Assim, embora as funções de agregação sejam úteis no dia-a-dia para resolver diversas questões com SQL, elas podem ser insuficientes para atender a toda demanda de associações exigidas quando processos analíticos mais complexos precisam ser feitos sobre os dados (HOKAMA *et al.*, 2004).

Para resolver isto, muitos BDs comerciais implementaram novas funções para tratamento de tarefas de análise de dados, como é o caso do BD da ORACLE, que adotou o uso de um tipo especial de função analítica em sua linguagem SQL a partir da sua versão 8i. Foram criadas quatro famílias de funções analíticas, cada uma das quais contendo várias funções que permitem obter melhor desempenho em consultas específicas e geram maior produtividade ao desenvolvedor (ORACLE CORPORATION, 1999).

Em 2001 estas funções foram revisadas pela ANSI e passaram a integrar oficialmente o padrão SQL, de acordo com a emenda à norma ANSI/ISO/IEC 9075 que descreve o padrão da linguagem (PRICE, 2009). Segundo o autor, elas podem ser usadas em conjunto com as funções de agregação para retornar múltiplas linhas para cada grupo dentro de uma instrução SQL.

No BD Oracle o processamento de consultas usando o SQL analítico ocorre em três fases: Primeiramente todas as junções, cláusulas *WHERE*, *GROUP BY* e *HAVING* são executadas. Em seguida estes resultados iniciais são usados como entrada para os cálculos das funções analíticas. Por fim, caso a instrução de consulta tenha uma cláusula *ORDER BY*, ela será processada para garantir uma ordem de saída precisa (ORACLE CORPORATION, 2014).

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

O uso dessas funções torna possível resolver problemas comuns no dia a dia e oferece ganho de performance, pois evita o uso de cláusulas de grupo, *sub-selects*, *union* e instruções que deixam o código maior ou dificultam a sua manutenção (MEIRA; SOUZA, 2002). De acordo com os autores, devido às otimizações que implementam, o uso destas funções traz melhorias significativas no desempenho de consultas ao BD além de contribuir para o aumento de produtividade dos desenvolvedores, já que estes podem criar análises com códigos escritos mais rapidamente e com manutenção facilitada, mais claros e reduzidos.

Os testes gerados para a conclusão deste artigo, fizeram uso das funções analíticas *SUM*, *LISTAGG*, *LAG* e *RANK*.

A função *SUM* é uma função de agregação, porém quando é complementada pelas cláusulas *OVER* e *PARTITION BY*, passa a produzir resultados como função analítica. A cláusula *OVER* define o escopo de uma função analítica e é obrigatória, sendo usada para indicar que a função opera em um conjunto de resultados da consulta. *PARTITION BY* divide o conjunto de resultados em grupos lógicos definidos pelas expressões de partição ou tratará todas as linhas do conjunto resultado como um único grupo quando for omitida (MEIRA; SOUZA, 2002).

A função *LISTAGG* é considerada um caso especial de função analítica. Ela concatena os dados de várias linhas em uma única linha, gerando uma lista. Não é comutativa, o que significa que a concatenação do primeiro e do segundo valor não é a mesma que a concatenação do segundo e do primeiro, diferentemente da soma ou da média, por exemplo e não permite o uso de *ORDER BY* na cláusula analítica (REPRINTSEV, 2018).

A função analítica *LAG* permite o acesso a mais de uma linha da tabela ao mesmo tempo e faz com que o campo se desloque conforme especificado. Se nenhum valor for indicado, o deslocamento padrão será de 1. Ela pode fornecer um ganho de desempenho significativo, já que não é necessária nenhuma associação automática de tabelas (ORACLE CORPORATION, 1999).

A função *RANK* produz uma classificação ordenada das linhas da consulta (*ranking*). Uma cláusula *PARTITION* (opcional) pode ser especificada para definir onde a classificação será redefinida e uma cláusula *ORDER BY* (obrigatória) deve ser usada para determinar a coluna a ser classificada. Se nenhuma partição for especificada, a classificação será executada em todo o conjunto resultado (MEIRA; SOUZA, 2002).

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

A maioria das funções analíticas também pode atuar como funções de agregação (se a cláusula *OVER* não for especificada) no entanto, algumas delas são puramente analíticas, como a função *RANK* (REPRINTSEV, 2018).

3 Materiais e métodos

Os testes descritos neste trabalho foram realizados em um computador com Sistema Operacional Windows na versão 7 Professional de 64 Bits, processador Intel® Core(TM) i5-7400 CPU @ 3.00 GHz e 8,00 GB de memória RAM.

Foi usado o BD relacional Oracle Database Express Edition 11g Release 2 para manter a base de testes e a ferramenta PL/SQL Developer para importação dos dados (ORACLE CORPORATION, 2014)

Dados públicos do programa Bolsa Família referentes ao mês de março de 2019 foram utilizados para os testes de consulta (PORTAL DA TRANSPARÊNCIA, 2019). Estes dados estão disponíveis em formato .CSV e são organizados em arquivos de 8 colunas: mês de referência, mês de competência, UF, código do município, nome do município, NIS do favorecido, nome do favorecido e valor da parcela.

Os códigos das consultas foram organizados automaticamente pelo recurso de indentação do PL/SQL Developer para análises de legibilidade e estudos de redução de linhas de código. Este mecanismo organiza os *scripts* de forma automática com recuos e espaçamentos que facilitam a leitura e entendimento, seguindo o padrão de organização de código SQL.

Os testes analisaram os resultados das consultas sob as métricas de custos, cardinalidade e *bytes*. O valor indicado pelo custo mostra a eficiência da busca e é um número calculado para auxiliar o otimizador de consultas do BD nos processos de escolha do melhor plano de execução. Seu cálculo considera fatores como I/O, ciclos de CPU e memória. A cardinalidade indica o nível de exclusividade de dados em uma coluna de uma tabela no BD, sendo uma medida da estimativa do número de linhas acessadas pela operação. Bytes indica a estimativa, feita pelo otimizador de consultas, do número de bytes que serão acessados pela operação de consulta.

Em cada cenário de teste, as consultas foram realizadas cinco vezes, desconectando-se e reconectando-se ao BD a cada nova execução para que o *cache* e o plano de execução gerados não interferissem nos resultados.

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

4 Resultados e discussão

Quatro diferentes cenários de testes permitiram executar códigos escritos com funções analíticas e os seus equivalentes em SQL padrão sem uso destas funções. Os resultados estão apresentados a seguir.

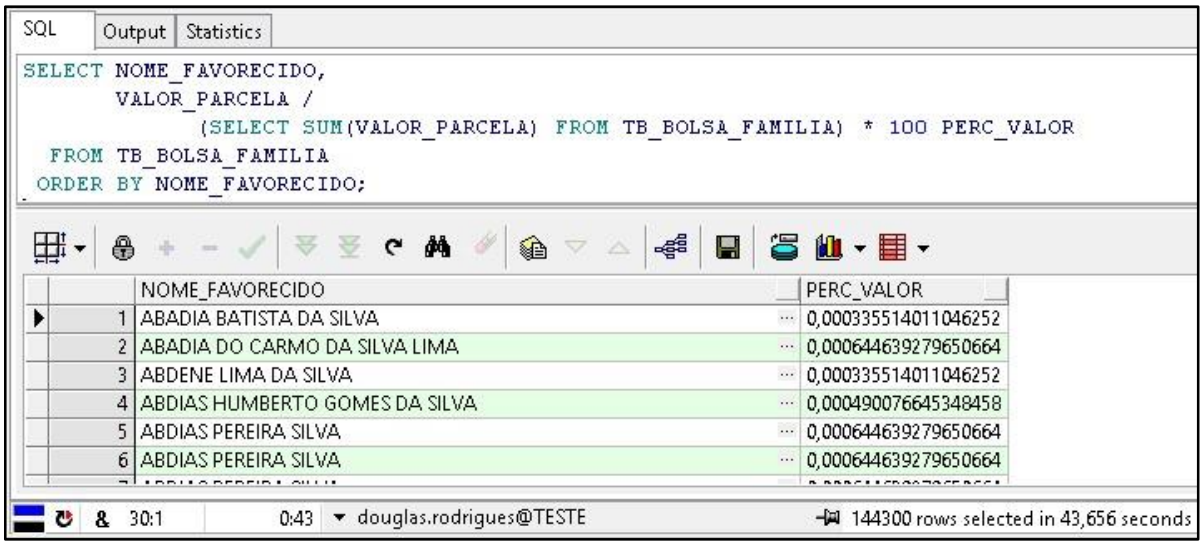
4.1 Consulta beneficiários

No primeiro cenário foi feita uma consulta aos nomes dos beneficiários e o percentual que o valor recebido representa sobre o valor total pago no período analisado (março de 2019). O resultado foi ordenando pelo nome do favorecido.

A primeira etapa deste cenário executou uma consulta escrita somente com comandos SQL padrão, onde um *SELECT* encadeado foi utilizado para calcular o percentual recebido. Foram retornados 144.300 linhas após 50,328 segundos na primeira execução (em cada cenário os comandos foram realizados 5 vezes).

Na Figura 1 é possível ver este comando SQL, uma parte dos registros recuperados e o tempo gasto para consultar pela primeira vez. O tempo médio das cinco execuções deste *script* foi de 48,994 segundos.

Figura 1: Consulta nomes dos beneficiários.



Em seguida, executou-se uma consulta para recuperação dos mesmos dados, porém com uso da função *SUM* associada às cláusulas *OVER* e *PARTITION BY*.

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

Neste caso, a primeira resposta da consulta retornou em um tempo 40,281 segundos e apresentou os mesmos 144.300 registros obtidos anteriormente. O tempo médio após as cinco execuções foi de 42,023 segundos.

A Tabela 1 apresenta um comparativo dos resultados das consultas deste cenário, considerando-se o uso da função analítica *SUM* com cláusulas *OVER* e *PARTITION BY* e daquela escrita com *SELECT* encadeado.

Tabela 1: Análise comparativa do cenário 1.

	Custo	**Cardinalidade**	**Bytes**	**Tempo médio (s)**
SELECT encadeado	4.929	174.297	20.044.155	48,994
Função analítica *SUM*	411	174.297	20.044.155	42,023

Nota-se uma melhoria no desempenho em relação ao tempo médio de execução e pode-se afirmar que o uso de funções analíticas ofereceu um ganho de 6,971 segundos neste cenário, representando 14,23% a menos.

Ao observar os valores atingidos na análise do custo, nota-se que a utilização da função analítica resultou em uma diferença a menos de 4.518 pontos.

Neste primeiro cenário a indentação automática da ferramenta PL/SQL Developer gerou código com 4 linhas quando a função analítica foi utilizada (Figura 2) e com 5 linhas com uso de *SELECT* encadeado (Figura 1).

Figura 2: Consulta com função analítica *SUM*.

```
SELECT NOME_FAVORECIDO,
       VALOR_PARCELA / SUM(VALOR_PARCELA) OVER(PARTITION BY NULL) * 100 PERC_VALOR
  FROM TB_BOLSA_FAMILIA
 ORDER BY NOME_FAVORECIDO;
```

4.2 Consulta beneficiários com valores acima de R$ 1000,00

Em seguida, desejou-se criar uma lista com os nomes de todos os beneficiários que recebem parcelas acima de R$ 1000,00.

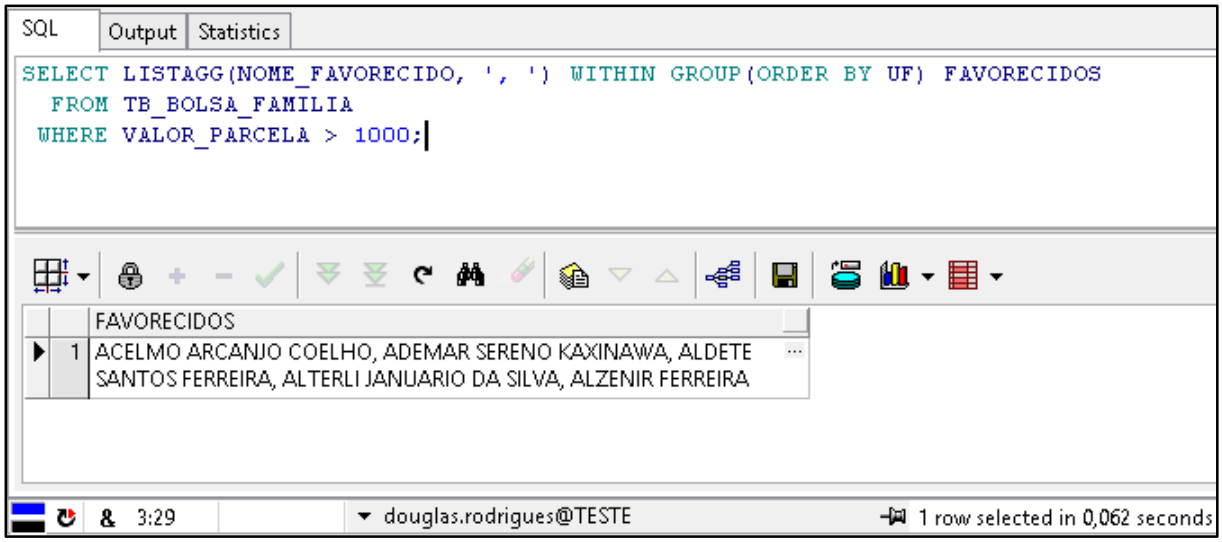
Na primeira etapa deste cenário, o *script* criado para consultar os dados usou a função analítica *LISTAGG*. A primeira execução foi realizada em 0,062 segundos e criou uma lista contendo 47 nomes.

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

A Figura 3 exibe o comando SQL usado, uma parte dos registros do resultado e o tempo da primeira execução. O tempo médio para as cinco aplicações do comando foi de 0,07 segundos.

Figura 3: Consulta com função analítica *Listagg*.



Como não há um comando na linguagem SQL padrão para criação de listas, foi necessário criar uma função para concatenar os nomes dos beneficiários pesquisados e fazer sua chamada em um *script* principal. A Figura 4 exibe o código da função criada e a Figura 5 apresenta a sua chamada.

Figura 4: Função de concatenação de nomes.

```
CREATE FUNCTION FU_CONCATENAR_FAVORECIDO(P_VALOR IN NUMBER) RETURN VARCHAR2 IS
  FAVORECIDO VARCHAR2(320000) := '';
BEGIN
  FOR X IN (SELECT NOME_FAVORECIDO
              FROM TB_BOLSA_FAMILIA
             WHERE VALOR_PARCELA > 1000) LOOP
    IF FAVORECIDOIS NULL THEN
      FAVORECIDO := X.NOME_FAVORECIDO;
    ELSE
      FAVORECIDO := FAVORECIDO || ', ' || X.NOME_FAVORECIDO;
    END IF;
  END LOOP;
  RETURN FAVORECIDO;
END FU_CONCATENAR_FAVORECIDO;
```


ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

O tempo de resposta da consulta na primeira vez foi de 0,391 segundos e recuperou a mesma lista contendo 47 nomes. O tempo médio após cinco execuções foi de 0,75 segundo.

O comparativo dos resultados deste cenário, considerando-se o uso da função analítica *LISTAGG e* o uso da função criada para concatenação dos nomes pode ser visto na Tabela 2.

Tabela 1: Análise comparativa do cenário 2.

	Custo	Cardinalidade	Bytes	Tempo médio (s)
Função para concatenação	412	143	1.859	0,75
Função analítica *LISTAGG*	411	1	118	0,07

Figura 5: Chamada da função para concatenação.

```
SELECT DISTINCT FU_CONCATENAR_FAVORECIDO(VALOR_PARCELA) CIDADES
  FROM TB_BOLSA_FAMILIA
 WHERE VALOR_PARCELA > 1000;
```

Nota-se uma diferença de 0,68 segundos no tempo médio de execução das consultas, o que representa um expressivo ganho de 90,67% ao fazer uso da função analítica. Merece destaque também as diferenças encontradas nas análises da cardinalidade (142) e dos bytes (1.741).

Ao analisar os códigos indentados automaticamente pela ferramenta PL/SQL Developer observa-se que o uso da função analítica *LISTAGG* reduziu o número de linhas de código em 14 linhas, além de ter evitado a criação de um objeto do tipo “função” no BD.

4.3 Consulta da variação dos totais pagos por município

No terceiro cenário de testes foi realizada uma consulta para análise da variação entre os totais pagos por município do estado de SP. Os registros foram filtrados com uso da cláusula *WHERE* e apresentados em ordem decrescente do valor total por cidade.

Na primeira etapa deste estudo, foi usada a função analítica *LAG*, gerando um código com 12 linhas. A primeira execução levou 0,218 segundos para retornar

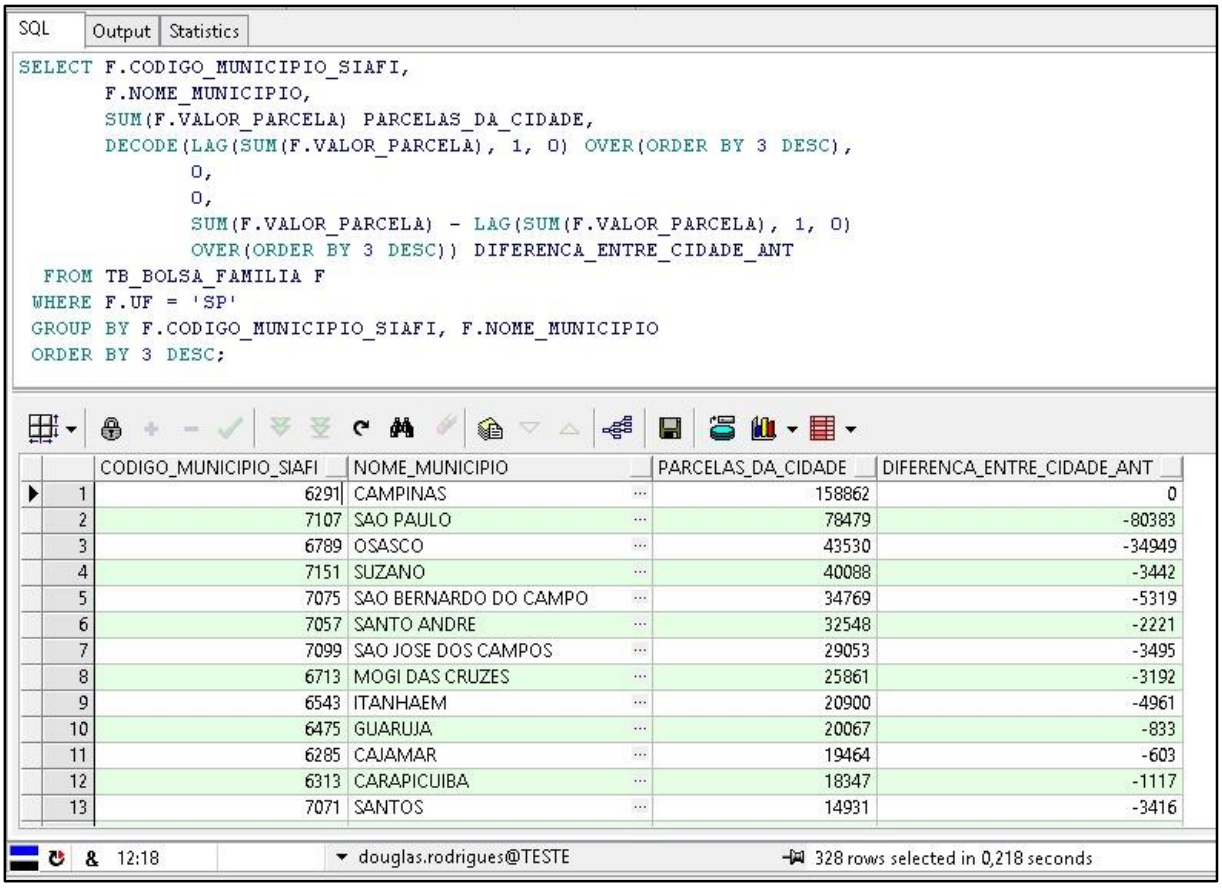
ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

328 registros e esta é a quantidade de municípios de SP disponíveis na base de dados pesquisada.

A Figura 6 exibe este comando SQL, uma parte dos registros recuperados e o tempo da primeira execução. O tempo médio das cinco execuções para esta etapa foi de 0,204 segundos.

Figura 6: Consulta com função analítica *LAG*.



SQL | Output | Statistics

```
SELECT F.CODIGO_MUNICIPIO_SIAFI,
       F.NOME_MUNICIPIO,
       SUM(F.VALOR_PARCELA) PARCELAS_DA_CIDADE,
       DECODE(LAG(SUM(F.VALOR_PARCELA), 1, 0) OVER(ORDER BY 3 DESC),
              0,
              0,
              SUM(F.VALOR_PARCELA) - LAG(SUM(F.VALOR_PARCELA), 1, 0)
              OVER(ORDER BY 3 DESC)) DIFERENCA_ENTRE_CIDADE_ANT
  FROM TB_BOLSA_FAMILIA F
 WHERE F.UF = 'SP'
 GROUP BY F.CODIGO_MUNICIPIO_SIAFI, F.NOME_MUNICIPIO
 ORDER BY 3 DESC;
```

	CODIGO_MUNICIPIO_SIAFI	NOME_MUNICIPIO	PARCELAS_DA_CIDADE	DIFERENCA_ENTRE_CIDADE_ANT
1	6291	CAMPINAS	158862	0
2	7107	SAO PAULO	78479	-80383
3	6789	OSASCO	43530	-34949
4	7151	SUZANO	40088	-3442
5	7075	SAO BERNARDO DO CAMPO	34769	-5319
6	7057	SANTO ANDRE	32548	-2221
7	7099	SAO JOSE DOS CAMPOS	29053	-3495
8	6713	MOGI DAS CRUZES	25861	-3192
9	6543	ITANHAEM	20900	-4961
10	6475	GUARUJA	20067	-833
11	6285	CAJAMAR	19464	-603
12	6313	CARAPICUIBA	18347	-1117
13	7071	SANTOS	14931	-3416

12:18 | douglas.rodrigues@TESTE | 328 rows selected in 0,218 seconds

Um *script* SQL padrão sem uso de função analítica para consultar os mesmos dados pode ser escrito com *SELECT* encadeado. Na Figura 7 é possível ver o que código criado, após a indentação, necessitou de 25 linhas e é um comando de difícil entendimento.

A primeira consulta com este comando executou em tempo de 2,048 segundos e recuperou os mesmos dados da consulta com uso de *LAG*. O tempo médio para 5 consultas foi de 2,057 segundos.

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

Figura 7: Consulta com comandos *SELECTs* encadeados.

```
SELECT CODIGO_MUNICIPIO_SIAFI,
       NOME_MUNICIPIO,
       SUM(VALOR_PARCELA),
       NVL((SELECT Y1.VALOR_PARCELA
             FROM (SELECT ROWNUM ORDEM, X1.*
                     FROM (SELECT F1.CODIGO_MUNICIPIO_SIAFI,
                                  F1.NOME_MUNICIPIO,
                                  SUM(F1.VALOR_PARCELA) VALOR_PARCELA
                             FROM TB_BOLSA_FAMILIA F1
                            WHERE F1.UF = 'SP'
                            GROUP BY F1.CODIGO_MUNICIPIO_SIAFI,
                                     F1.NOME_MUNICIPIO
                            ORDER BY 3 DESC) X1) Y1
            WHERE Y1.ORDEM = Y.ORDEM - 1),
           0) DIFERENCA
  FROM (SELECT ROWNUM ORDEM, X.*
          FROM (SELECT F.CODIGO_MUNICIPIO_SIAFI,
                       F.NOME_MUNICIPIO,
                       SUM(F.VALOR_PARCELA) VALOR_PARCELA
                  FROM TB_BOLSA_FAMILIA F
                 WHERE F.UF = 'SP'
                 GROUP BY F.CODIGO_MUNICIPIO_SIAFI, F.NOME_MUNICIPIO
                 ORDER BY 3 DESC) X) Y
 GROUP BY CODIGO_MUNICIPIO_SIAFI, NOME_MUNICIPIO, ORDEM
 ORDER BY 3 DESC;
```

A Tabela 3 apresenta o comparativo dos resultados deste cenário considerando-se o uso da função analítica *LAG* e o comando com *SELECTs* encadeados.

Tabela 3: Análise comparativa do cenário 3.

	Custo	Cardinalidade	Bytes	Tempo médio (s)
SELECTs encadeados	415	17	2.397	2,057
Função analítica *LAG*	412	17	2.227	0,204

Nota-se que o otimizador de consultas resolveu a consulta com uma vantagem de 1,853 segundos quando usou função analítica, o que representa um ganho expressivo de 90,08%. Obteve-se também uma diferença de 3 pontos na análise de custo, cardinalidades iguais e de 170 na análise de bytes.

A comparação dos códigos utilizados, quando indentados automaticamente pela ferramenta PL/SQL Developer, apresentou diferença de 13 linhas a menos do que o código com função analítica.

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

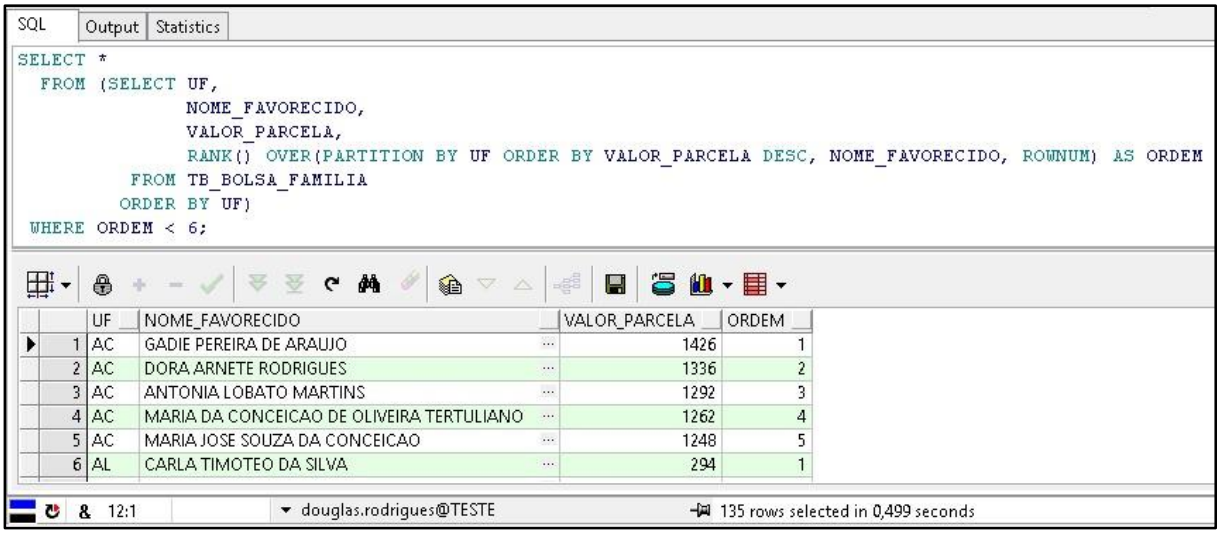
4.4 Consulta aos cinco beneficiários com maior valor recebido por estado

O quarto cenário de testes propôs-se a retornar os cinco beneficiários em cada Estado (UF) com maior valor de parcela recebida. Uma coluna indicativa de ordem também foi requerida como resposta.

A função analítica *RANK* foi utilizada para obter o resultado e um filtro foi aplicado com a cláusula *WHERE* para que somente os cinco primeiros nomes de cada Estado fossem retornados.

O código de 8 linhas executou pela primeira vez em um tempo de 0,516 segundos e pode ser visto na Figura 8, juntamente com uma parte dos resultados da consulta (inclusive a coluna criada para indicar a ordem dos registros) e o tempo da primeira execução. O tempo médio obtido pelas 5 tentativas foi de 0,490 segundos.

Figura 8: Consulta com função analítica *Rank*.



Uma consulta para recuperação dos mesmos dados, porém sem uso de função analítica, foi obtida com o comando *UNION ALL*. A primeira execução desta consulta levou um tempo de 0,687 segundos e trouxe os mesmos dados da consulta com função analítica. O tempo médio para 5 execuções foi de 0,6 segundos.

A Figura 9 exibe o comando usado, mas sem a aplicação da indentação automática da ferramenta PL/SQL Developer. Com a indentação foi gerado um *script* com 163 linhas.

A Tabela 4 mostra um comparativo dos resultados obtidos, considerando-se o uso da função analítica *RANK* e o comando *UNION ALL*.

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

Tabela 4: Análise comparativa do cenário 4.

	Custo	Cardinalidade	Bytes	Tempo médio (s)
Uso de *UNION ALL*	14.462	174.366	22.841.946	0,6
Função analítica *RANK*	5.027	174.297	22.832.907	0,490

Figura 9: Consulta com comando *Union All* e sem indentação automática.

```
SELECT * FROM (SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'AC' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'AL' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'AM' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'AP' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'BA' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'CE' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'AL' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'DF' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'ES' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'GO' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'MA' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'MG' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'MS' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'MT' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'PA' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'PB' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'PE' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'PI' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'PR' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'RJ' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'RN' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'RO' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'RR' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'RS' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'SC' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'SE' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'SP' ORDER BY VALOR_PARCELA DESC )X
     UNION ALL SELECT X.*, ROWNUM ORDEM FROM (SELECT UF, NOME_FAVORECIDO,VALOR_PARCELA FROM TB_BOLSA_FAMILIA WHERE UF = 'TO' ORDER BY VALOR_PARCELA DESC )X
) Z WHERE Z.ORDEM < 6;
```

Percebe-se um ganho no tempo médio com o uso da função analítica de 0,11 segundos (18,33%) quando comparado com a consulta com *UNION ALL*. O custo foi menor em 9.435 pontos, a cardinalidade menor em 69 e os bytes menores em 9.039 comparados com o uso da função analítica.

A comparação dos códigos das duas consultas, indentados automaticamente pela ferramenta PL/SQL Developer mostrou uma diferença de 155 linhas a menos de código para a opção com função analítica *RANK*.

Os cenários testados permitiram notar que o uso de funções analíticas em comandos SQL facilita a escrita dos *scripts* de consulta, diminui e organiza o código usado e os torna mais performáticos.

Considerações finais

Desde a criação dos BDs, as operações de consultas são de grande importância, especialmente quando algum tipo de análise precisa ser feito para tomada de decisão. Cálculos sobre os dados podem ser obtidos com funções de agregação oferecidas pela linguagem SQL padrão, porém certas análises estatísticas mais complexas resultam em comandos extensos, de difícil

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

entendimento e que apresentam funcionamento de má qualidade. Funções analíticas são capazes de consultar dados empregando comandos com menor quantidade de linhas e que oferecem boa performance, se comparados com comandos equivalentes escritos em SQL padrão.

Assim, o objetivo deste trabalho foi estudar como as funções analíticas podem ser usadas como opção para construção de comandos SQL para consultas no BD Oracle que sejam de menor complexidade e que executem com bom desempenho.

Os cenários de testes criados permitiram conhecer vantagens oferecidas por este tipo de função. Ao comparar os códigos utilizando-se funções analíticas com códigos sem a presença delas, foi possível notar uma grande diminuição da quantidade de linhas necessárias, o que melhora o entendimento, facilita a criação e manutenção dos *scripts*.

Os resultados comparativos das métricas estudadas se mostraram promissores e evidenciaram vantagens da utilização destas funções como alternativa de implementação para recuperação de informações no BD Oracle.

Agradecimentos

Agradeço a Deus, por ter me fortalecido para conseguir superar os obstáculos e dado saúde para alcançar esta etapa da minha vida. Aos funcionários da FATEC Franca pelo apoio. À minha família e amigos por terem acreditado em mim desde o primeiro instante. Sou o que sou, devido a vocês estarem sempre ao meu lado. A todas as demais pessoas que fizeram parte do meu percurso de alguma forma.

Referências

BEAULIEU, Alan. **Aprendendo SQL: Dominando os Fundamentos de SQL**. Novatec Editora, 2010.

DATE, Christopher J. **Introdução a sistemas de bancos de dados.** Elsevier Brasil, 2004.

GRAY, Jim et al. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. **Data mining and knowledge discovery**, v. 1, n. 1, p. 29-53, 1997.

HOKAMA, Daniele Del Bianco et al. **A modelagem de dados no ambiente Data Warehouse**. São Paulo, p. 32, 2004.

ISSN: 2595-5942

Revista EduFatec: educação, tecnologia e gestão
V.2 N.1 - janeiro-junho/2019

KEIM, D. A.; SIPS, M.; ANKERST, M. Visual Data-Mining Techniques. In: **Visualization Handbook**. [s.l.] Elsevier, 2005. p. 831–843.

MACEDO, Bruno P. et al. Uma abordagem prática sobre otimização de Banco de Dados utilizando o gerenciamento automático de memória no Sistema Gerenciador de Banco de Dados Oracle. **Caderno de Estudos Tecnológicos**, v. 1, n. 1, 2013.

MEIRA, C. A. A.; SOUZA, T. A. L. Funções analíticas da linguagem Sql do Oracle. **Embrapa Informática Agropecuária-Comunicado Técnico (INFOTECA-E),** 2002.

NASCIMENTO, Guilherme Pereira do. **Otimização de consultas em banco de dados relacionais com o uso de índices.** 2015. Trabalho de Graduação (Tecnólogo em Análise e Desenvolvimento de Sistemas) - Fatec Franca, Franca, 2015.

NAVATHE, Shamkant B.; ELMASRI, Ramez. **Sistemas de banco de dados.** Sham, Addison. Ribeirão Preto SP, 2005.

OLIVEIRA, K. L.; PAIVA, C. E. Tuning de banco de dados com SQL Server. **Revista EduFatec**, Franca, 1.sem. 2018. Disponível em: https://revistaedufatec.fatecfranca.edu.br/. Acesso em: 11 mar. 2019.

ORACLE CORPORATION. **Analytic functions for Oracle8i:** an Oracle technical white paper. Redwood Shores, 1999. 20 p.

ORACLE CORPORATION. **Oracle Database Online Documentation 11g Release 2 (11.2)**. [S. l.], 2014. Disponível em: https://docs.oracle.com/cd/E11882_01/index.htm. Acesso em: 11 mar. 2019.

PORTAL DA TRANSPARÊNCIA. **Bolsa Família - Pagamentos - Portal da Transparência.** 2019. Disponível em: http://www.portaltransparencia.gov.br/download-de-dados/bolsa-familia-pagamentos. Acesso em: 10 abr. 2019.

PRICE, Jason. **Oracle Database 11g SQL: Domine SQL e PL/SQL no banco de dados Oracle.** Bookman Editora, 2009.

PUGA, Sandra; FRANÇA, Edson; GOYA, Milton. **Banco de Dados:** Implementação em SQL, PL/SQL e Oracle 11g. FRANÇA, E.; GOYA, M. São Paulo: Ed. Pearson Education do Brasil, 2013.

REPRINTSEV, Alex. Analytic Functions. In: **Oracle SQL Revealed.** Apress, Berkeley, CA, 2018. p. 85-102.

SILBERSCHATZ, Abraham; SUNDARSHAN, S.; KORTH, Henry F. **Sistema de banco de dados.** Elsevier Brasil, 2016.

ISSN: 2595-5942