UNIVERSIDADE DE CAXIAS DO SUL
CENTRO DE COMPUTAÇÃO E TECNOLOGIA DA INFORMAÇÃO

BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

FELIPE BOGO

MELHORES PRÁTICAS PARA O DESENVOLVIMENTO DE *STORED PROCEDURES* EM PL/SQL

Trabalho de Conclusão de Curso do Centro de Computação e Tecnologia – Curso de Bacharel em Ciência da Computação da Universidade de Caxias do Sul

Orientadora: Helena G. Ribeiro

Caxias do Sul, RS

2013

RESUMO

Muitas aplicações desenvolvidas utilizando banco de dados Oracle possuem sua lógica de negócio escritas em PL/SQL no banco através de *stored procedures* para utilizar o desempenho que este SGBD pode oferecer. O PL/SQL é uma linguagem muito ampla e possibilita implementar uma função de muitas maneiras diferentes, mas para obter um bom desempenho na aplicação, é necessário escrever da melhor maneira possível.

O objetivo deste trabalho é estudar recursos disponibilizados pelo SGBD Oracle até a versão 11g release 2, testar em diversos cenários e fazer uma análise da execução destes recursos afim de saber em que cenários a partir de quais recursos pode se obter um melhor desempenho na aplicação.

Palavras-chaves: Oracle 11g. Tuning. PL/SQL. Stored Procedure.

ABSTRACT

Many applications developed using Oracle database have your business logic written in PL/SQL in the database through stored procedures to use performance DBMS that this can offer. The PL/SQL language is very broad and allows to implement a function in many different ways, but to get a good performance in the application, it is necessary to write the best possible way.

The objective of this work is to study the resources provided by the DBMS Oracle up to version 11g release 2, testing in various scenarios and make an analysis of the implementation of these resources in order to know in which scenarios from which resources can achieve better performance in the application.

Keywords: Oracle 11g. Tuning. PL/SQL. Stored Procedure.

LISTA DE SIGLAS

ADDM	Automatic Database Diagnostic Monitor
AWR	Automatic Workload Respository
CPU	Centra Processing Unit
DBA	Database administrator
DCL	Data Control Language
DDL	Data definition language
DML	Data manipulation language
I/O	Input/Output
NDS	Native Dynamic SQL
PL/SQL	Procedural Language/Structured Query Language
RAID	Redundant Array of Independent Drives
SGBD	Sistema gerenciado de banco de dados
SGA	System Global Area
SQL	Structured Query Language

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1. INTRODUÇÃO

Banco de dados é uma coleção de dados relacionados. Com dados, queremos dizer fatos conhecidos que podem ser registrados e possuem significado implícito (ELMARSRI, 2011).

É correto afirmar que os bancos de dados desempenham um papel crítico em quase todas as áreas em que os computadores são usados, incluindo negócios, comércio eletrônico, educação e biblioteconomia (ELMARSRI, 2011).

Para realizar a administração de um banco de dados conta-se com o auxílio de um SGBD. Um sistema gerenciador de banco de dados é uma coleção de programas que permite ao usuário criar e manter um banco de dados. O SGBD é um sistema de software de uso geral que facilita o processo de definição, construção, manipulação e compartilhamento de bancos de dados entre diversos usuários e aplicações (ELMARSRI, 2011).

Existem diversos SGBD's, cada um com suas particularidades, finalidades e nichos de mercado. Entre os de maior popularidade destacam-se Oracle, IBM DB2, Microsoft SQL Server, PostgreSQL e MySQL (ANDRADE, 2005), no entanto ainda existe a aplicação que se comunica com o SGBD, se ela não for desenvolvida de forma adequada, existirão problemas de performance que tornam inviável o seu uso, onde encontra-se o foco deste trabalho.

Dos problemas de performance em banco de dados, 65% das causas de baixa performance são pelas aplicações, 25% são causados pelos modelos de dados, 5% são problemas do SGDB, 5% são causadas por configurações ou deficiências do Sistema Operacional (MILENE, 2007). Portanto a maneira mais eficiente para reduzir estes problemas é através da otimização da aplicação.

A maneira mais comum para otimizar uma aplicação de banco de dados é através do *tuning* dos componentes do banco de dados. *Tuning* de banco de dados ou sintonia fina é atividade relacionada a fazer com que a aplicação execute mais rapidamente. Entende-se por "mais rapidamente", maior rendimento da aplicação, ou seja, menor tempo de resposta da aplicação (SHASHA,2003).

Para que um banco de dados execute mais rapidamente será necessário mudar a maneira que a aplicação é desenvolvida, as estruturas de dados e parâmetros do SGBD, a configuração do sistema operacional, ou *hardware (*SHASHA,2003*)*, no entanto por ser tratar de uma área bem ampla. Para este trabalho, serão estudadas maneiras de realizar o *tuning* da aplicação extraindo o melhor dos recursos do sistema gerenciador de banco de dados.

Devido a cada SGBD possuir suas próprias características e particularidades, para este trabalho será utilizado o SGBD ORACLE versão 11g release 2 para realizar melhores práticas de implementação e otimizar o custos das aplicações.

Através do Oracle é possível desenvolver *stored procedures* utilizando *PL/SQL*. Segundo Thomas Kyte, um dos maiores especialistas em Oracle do mundo, *PL/SQL* é a forma mais rápida de processar dados em Bancos de Dados Oracle. Entenda “processar dados” como não apenas submeter instruções SQL (UPDATE, SELECT, INSERT e DELETE) para o banco, mas sim, efetuar transações que consistem de várias estruturas de decisão, *loops* e outros tipos de processamento (PRADO, 2011). Porém para aproveitar o máximo destes recursos é necessário ter um conhecimento apropriado da linguagem.

O Objetivo deste trabalho é otimizar as aplicações desenvolvidas em banco de dados. Para atingir este objetivo, será abordado o conceito de *tuning* para contextualizar o meio ao qual este trabalho está inserido. No capítulo 2, os principais conceitos serão explicados, serão explicados alguns tipos de *tuning* nos componentes do SGBD, para se ter conhecimento da quantidade de cenários que podem ser otimizados através do *tuning*. E em seguida será explicado sobre as *stored procedures*, pois são através delas que as aplicações comunicam-se como o banco de dados, e é nelas que a lógica de programação é escrita. No final do capítulo 2 será falado sobre o PL/SQL, a linguagem que será o estudo deste trabalho na otimização da aplicação.

Para otimizar o desenvolvimento de *stored procedures* utilizando de forma adequada recursos avançados em PL/SQL disponibilizados pelo SGBD ORACLE 11g, o capítulo 3 vai explicar os recursos estudados para resolver esta tarefa.

Ao término deste trabalho foram realizadas comparações dos algoritmos estudados e maneira usual de resolver tais problemas. Além disto, através deste estudo também foi possível entender quais foram os cenários que cada algoritmo teve um melhor desempenho.

2. PRINCÍPIOS DO *TUNING* DE BANCO DE DADOS

Tuning ou sintonia fina, é a atividade responsável por fazer com que a aplicação execute mais rapidamente, diminuir o tempo de resposta entre as consultas e transações e melhorar o rendimento geral das transações (ELMARSRI, 2003). A vantagem da utilização desta técnica é que não é necessário o conhecimento de fórmulas matemáticas complicadas, no entanto para ser um bom *tuner*[1] é preciso ter um grande domínio sobre a aplicação, sobre o sistema de gerenciador do banco de dados, sobre sistema operacional e até mesmo sobre hardware. Portanto por ser uma área tão abrangente existem *tuners* que se especializam em cada uma destas categorias.

O *tuning* pode ser aplicado de diversas maneiras. Pode ser necessário alterar a maneira como a aplicação foi desenvolvida, as estruturas de dados e os parâmetros do sistema gerenciador do banco de dados, a configuração do sistema operacional ou até mesmo *hardware.*

Para extrair maior aproveitamento do *tuning* é necessário a compreensão de cinco princípios básicos explicados em SHASHA (2003):

- pense global; conserte local;
- particionamento soluciona gargalos;
- custos de inicialização são altos; custos de execução são baixos;
- deixe para o servidor o que é responsabilidade do servidor;
- esteja preparado para resolver conflitos.

2.1. PENSE GLOBAL; CONSERTE LOCAL

Para ser eficiente é necessário pensar na melhor solução alterando o mínimo possível, ou seja, para resolver um problema é preciso conhecer o objetivo da aplicação para realizar a melhor alteração visando maior custo benefício. Por

[1] Responsável pelo *tuning* da aplicação.

exemplo, ao se analisar o desempenho do sistema nota-se um grande uso da CPU, *input/output*(I/O)[2] e paginação. À primeira vista pode-se dizer que a solução é melhorar o *hardware* para resolver o problema, no entanto este custo de processamento pode ser reduzido encontrando *queries*[3] mal elaboradas e consertando-as. Porém, melhorar o desempenho de uma *query* pode não resolver o problema se ela é executada raramente. Caso esta consulta seja executada várias vezes em várias partes do sistema, o ganho na economia de recursos pode ser muito grande.

Com este exemplo fica claro que não basta conhecer o problema local para encontrar a melhor solução, mas é necessário o conhecimento do todo para se aplicar o *tuning* na parte mais significativa e proporcionar o maior ganho.

2.2. PARTICIONAMENTO SOLUCIONA GARGALOS

Gargalos podem ocorrer quando muitos dados vindos de origens diferentes acumulam-se em um determinado ponto, todos dependentes dos mesmos recursos para serem processados. A primeira estratégia é resolver o problema de forma local, ou seja, se necessário criar um índice ou reescrever a consulta para tentar aproveitar melhor os índices existentes. Se mesmo com a primeira estratégia os resultados não forem satisfatórios, então parte-se para a segunda solução, o particionamento.

Particionamento em um sistema de banco de dados é a técnica responsável por reduzir a carga de um determinado componente do sistema dividindo a carga entre várias fontes ou distribuindo a carga ao longo do tempo. Resumidamente, particionamento é a divisão de um trabalho em partes distintas e iguais para serem processadas. No entanto, deve-se prestar atenção, pois particionamento nem sempre é uma boa solução, por exemplo, após a divisão do processo em partes distintas existe um custo para juntar estas informações, e muitas vezes este custo pode ser maior que o ganho obtido, portanto, esta técnica deve ser utilizada com cuidado.

[2] Leitura e escrita em disco.
[3] São consultas em *SQL*.

2.3. CUSTOS DE INICIALIZAÇÃO SÃO ALTOS; CUSTOS DE EXECUÇÃO SÃO BAIXOS

Para inicializar recursos em um banco de dados gasta-se um tempo muito maior do que gasta-se para utilizar pela segunda vez. Então deve-se tentar aproveitar ao máximo possível os recursos carregados e programar de forma a conseguir fazer este reaproveitamento. Por exemplo, quando um SQL é executado pela primeira vez existe um custo de *parse*[4]. Quando não há alteração de sintaxe e este mesmo SQL é executado novamente, este *parse* não é realizado. Se este SQL fosse chamado milhares de vezes este custo poderia se tornar um problema, se o *parse* fosse realizado várias vezes.

2.4. DEIXE PARA O SERVIDOR O QUE É RESPONSABILIDADE DO SERVIDOR

Para conseguir o melhor desempenho de um sistema de banco de dados apenas com o *tuning* esse resultado não será obtido. É necessário modelar de forma correta o trabalho realizado pelo servidor e pela aplicação.

Os seguintes fatores definem como uma tarefa deveria ser alocada:

- a distribuição de recursos entre servidor e cliente: Por exemplo, quando o servidor está sobrecarregado, todas as demais tarefas poderiam ser realocadas para o cliente;
- onde a alteração ocorre: Por exemplo, para atualizar uma tela de consulta quando alguma informação no banco de dados mudar é possível realizar consultas periodicamente através da aplicação, mas ao invés disto em um sistema bem projetado, o servidor dispara uma *trigger* sempre que uma alteração no banco de dados ocorrer e atualiza apenas as informações que mudaram evitando a sobrecarga de informações desnecessárias;

[4] Parse (análise) é a fase onde o Oracle analisa se a sintaxe do SQL e se os objetos utilizados existem ou estão acessíveis e em seguida cria o plano de execução para executar a consulta (BURLESON, 2009).

- se as tarefas do banco de dados interagem com a tela: Neste caso o acesso ao banco de dados deve ser realizado em uma interação independente, caso contrário, este período onde usuário interage com a tela outros processos ficarão bloqueados até a conclusão da tarefa.

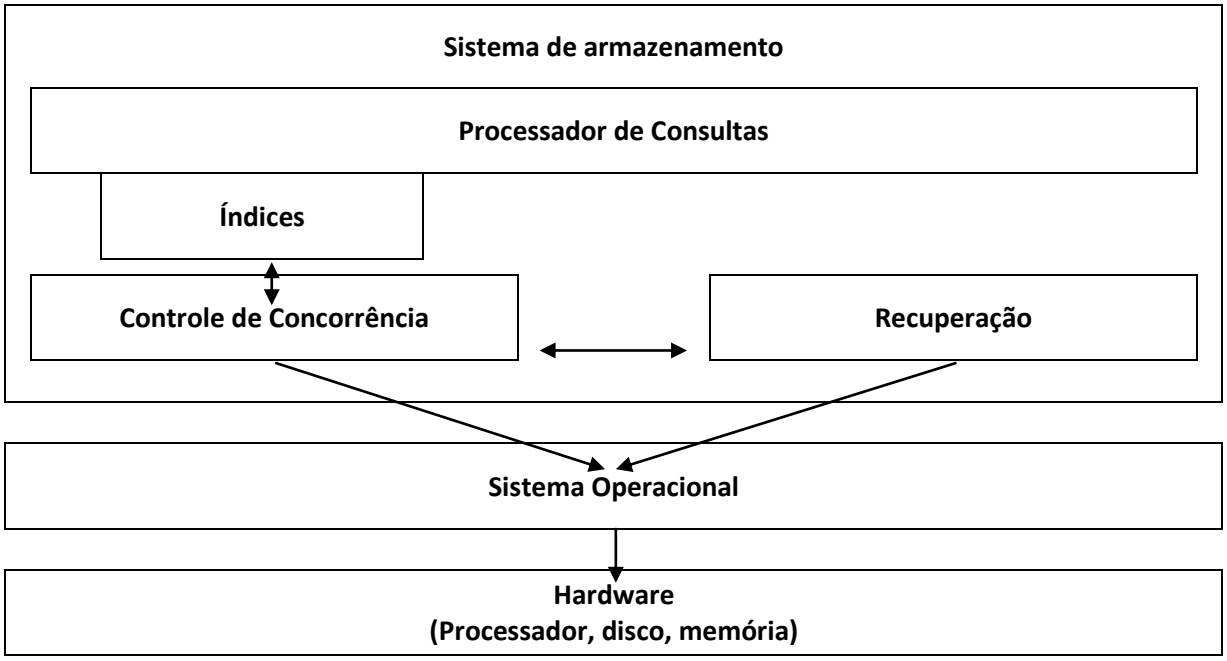
2.5. ESTEJA PREPARADO PARA RESOLVER CONFLITOS

Nas técnicas citadas nos itens acima não foi mencionado o custo de cada escolha, por exemplo, quando é utilizado um índice para otimizar uma consulta, para armazenar este índice é necessário mais espaço em disco, e dependendo o volume de dados pode ser necessário a aquisição de um novo disco rígido. Portanto, quando está realizando *tuning* é necessário saber quando se está disposto a pagar pela otimização que se deseja conseguir.

2.6. *TUNING* DE COMPONENTES DO SGBD

O *tuning* de componentes abrange todos os componentes relacionados ao sistema, como pode ser visto na Figura 1, *tuning* não está limitado somente ao banco de dados. É possível melhorar o desempenho diretamente no banco de dados otimizando o processador de consultas, índices, como pode ser necessário a configuração do sistema operacional ou até mesmo a aquisição de hardware.

Figura 1 - Componentes de um SGBD



A maior parte dos sistemas de bancos de dados é composta pelo processador de consultas, índices, controle de concorrência, recuperação, sistema operacional e *hardware*.

2.6.1. CONTROLE DE CONCORRÊNCIA

A concorrência ocorre quando duas aplicações atuam sobre os mesmos dados ao mesmo tempo. Por exemplo, em um sistema que controla o estoque, quando o item é retirado do estoque o sistema verifica se a quantidade a ser retirada no estoque e subtrai quando existir ou informa ao usuário que o estoque não possui a quantidade informada. Mas supondo que dois usuários utilizem o sistema simultaneamente, ao consultar o item é mostrado 10, o primeiro usuário retira 7 itens e o segundo retira 5 itens. Como os dois usuários estão realizando a retirada ao mesmo tempo somente será visível para o segundo usuário a quantidade atual do item após o primeiro usuário finalizar a sua retirada, e quando o segundo usuário for finalizar a sua retirada irá gerar erro porque o estoque ficará negativo. Para evitar este tipo de erro é necessário do controle de concorrência através do bloqueio das transações.

O bloqueio de transações pode tornar as aplicações bem lentas, por isto é importante o *tuning* nesta área.

Conforme é situado em SHASHA (2003), abaixo estão algumas dicas de bloqueio que são melhores para a performance da aplicação:

- menos bloqueios quanto for possível;
- bloqueios de leitura são os menos custosos;
- quanto menor for o tempo de bloqueio em uma transação melhor;
- analisar com cuidado a aplicação e configurar os níveis de isolamento.

2.6.2. RECUPERAÇÃO DO SISTEMA

A recuperação do sistema se faz necessária em casos de falhas de sistema ou de *hardware*. Um erro de *hardware* pode ser a falha de um processador ou um disco, mas quando esta situação acontecer, o SGBD deve ser capaz de desfazer a transação que ainda não foi completada ou refazer a que foi completada, pois os efeitos não foram salvos no banco.

Para ser capaz de desfazer ou refazer uma alteração o banco de dados armazena *logs* das transações, são armazenadas imagens das tabelas alteradas antes e após as alterações para ser possível desfazer uma alteração não completada.

O *tuning* de recuperação consiste de técnicas para melhor administrar a recuperação.

Seguem abaixo as principais maneiras citadas em SHASHA (2003) para *tuning* de recuperação do sistema:

- armazenar o *log* em discos separados para evitar buscas de dados;
- realizar menos atualizações nos discos do banco de dados quanto for possível;
- reduzir o tamanho de transações de atualização de dados.

2.6.3. SISTEMA OPERACIONAL

O sistema operacional realiza diversas funções que podem ter um impacto significativo na performance da aplicação. Quando se trata de sistema operacional, diferentes configurações irão influenciar na execução da aplicação. A quantidade de processos controlada pelo sistema operacional, mapeamento de memória virtual e física, número de *threads* que terão acesso ao banco de dados concorrentemente, prioridades dos processos, tamanho da página/bloco entre outras configurações terão impacto direto sobre a performance do banco. O ideal é que o sistema de banco de dados tenha prioridade na escolhas destas configurações para ter um bom desempenho.

2.6.4. HARDWARE

O *tuning* do *hardware* está diretamente ligado ao *software*. O fato de um recurso estar sobrecarregado não necessariamente implica na compra de mais discos, memória, etc. Em muitos casos a correção do software é a solução, mas ainda assim pode ser necessário a aquisição de novos recursos. Desta forma, o *tuning* do *hardware* pode ir da compra de novos recursos até a configuração dos recursos existentes, com o objetivo de obter o melhor desempenho para aplicação. Por exemplo, se a necessidade da aplicação for desempenho, os discos podem ser configurados em RAID 0[5]. Caso a necessidade seja segurança, a configuração que poderia ser utilizada seria RAID 1[6].

[5] Conhecido como *striping* o RAID 0 é composto de dois ou mais discos, os dados são escritos sequencialmente e o objetivo é a performance.
[6] Conhecido como espelhamento o RAID 1é composto de dois ou mais discos, sendo que o objetivo do segundo disco é duplicar as informações do primeiro, o objetivo desta configuração é a segurança dos dados.

2.7. SELF-TUNING

Self-tuning é a capacidade do sistema alterar seus parâmetros internos com o objetivo de melhorar o desempenho de suas funções. O SGBD coleta os dados durante o funcionamento, e utiliza estas informações para se auto ajustar (CHAUDHURI, 2013).

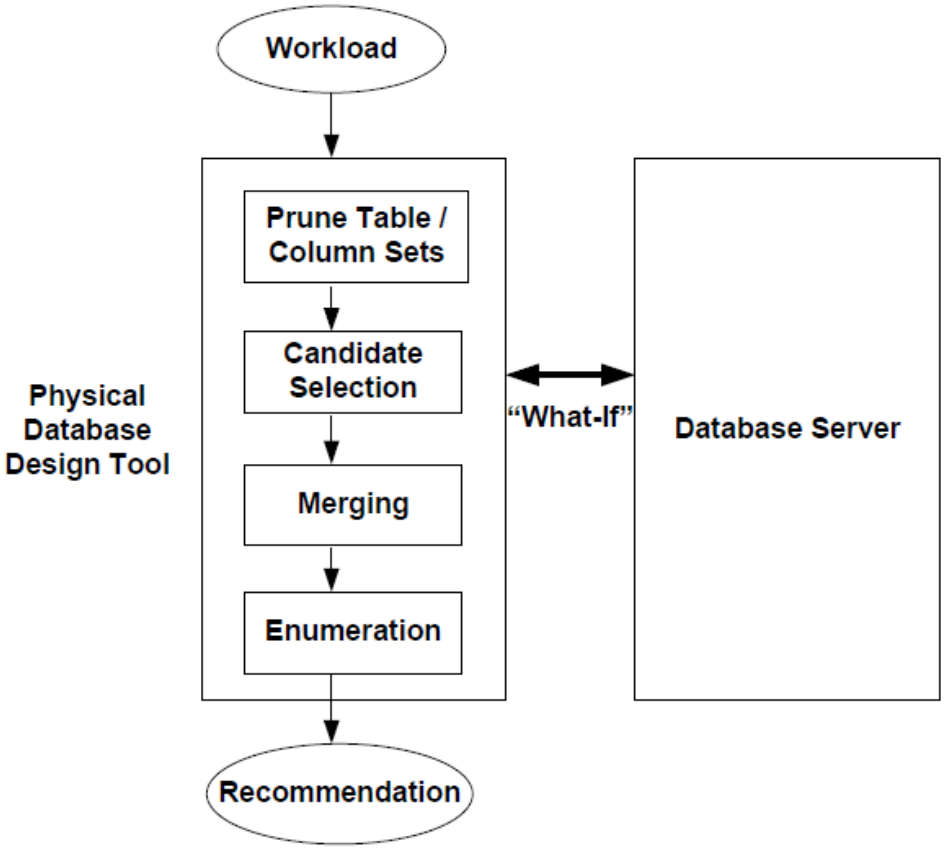
Uma maneira interessante que as empresas que comercializam SGBDs estão introduzindo o *self-tuning* é através de ferramentas de diagnósticos. Por exemplo, através do *Automatic Database Diagnostic Monitor* (ADDM) disponível na versão 10g do SGBD da Oracle. Esta ferramenta analisa dos dados no *Automatic Workload Respository* (AWR) para identificar potenciais gargalos no sistema. Para cada problema detectado, a ferramenta localiza a raiz do problema e fornece ao DBA recomendações para corrigí-lo. Ferramentas como esta auxiliam o trabalho de otimizar a aplicação.

A introdução de *materialized views* e particionamento de resultados deu inicio à uma explosão de alternativas no campo das modelagens físicas alternativas (CHAUDHURI, 2013).

A Figura 2 mostra os avanços na arquitetura das ferramentas de modelagem física do banco de dados após os avanços nesta área. Os algoritmos de busca destas ferramentas passam por estas quatro etapas para serem capazes de encontrar a configuração com o menor custo de processamento.

Em 1998, Microsoft SQL Server 7.0 foi o primeiro sistema gerenciador de banco de dados comercial a introduzir uma ferramenta de modelagem de banco de dados física chamada Tuning Wizard. Atualmente, já existem outras empresas disponibilizando estas ferramentas, como por exemplo, Oracle 10g com SQL Access Adivisor.

Figura 2 - Arquitetura das ferramentas de modelagem física dos bancos de dados após os avanços nos estudos de *self-tuning*



Fonte: Disponível em <http://www.cs.washington.edu/education/courses/cse544/07au/lecture-notes/lecture15/lecture15.pdf> Acessado em 15/05/2013.

2.8. *TUNING* DE ÍNDICES

Um índice para uma tabela é um conjunto de dados que permite a consulta a certos registros de uma tabela mais rapidamente. A aplicação correta dos índices é fundamental para uma boa performance no banco de dados. Os índices são mantidos pelo gerenciador de armazenamento, eles organizam o acesso aos dados na memória, também são fortemente integrados ao controle de concorrência e utilizados pelo processador de consultas na otimização de uma consulta (SHASHA,2003).

A utilização inadequada de índice implica no efeito contrário. A má utilização de um índice irá resultar em vários problemas no banco de dados, os seguintes exemplos podem ser citados (SHASHA,2003):

- índices que geram custo sendo mantidos, mas nunca são utilizados;
- arquivos escaneados que retornam um único registro;
- junções de várias tabelas que ficam executando por horas devido aos índices errados.

O benefício da utilização dos índices vai depender de como eles são utilizados nas consultas. Por exemplo, ao criar um índice sobre uma consulta, inserções, exclusões e algumas atualizações irão gerar manutenção deste índices. Para que este índice implique em um efeito positivo, a coluna indexada deve ser um filtro relevante para as consultas realizadas nesta tabela, caso contrário o índice estará apenas gerando custo de manutenção da tabela.

2.9. *TUNING* DE CONSULTAS

O *tuning* de consultas é o primeiro método a ser utilizado quando se fala de otimizar a aplicação. De todos os métodos citados, este é o único método que traz somente benefícios à aplicação. Muitas vezes uma adição de um índice, alteração no esquema ou modificação nos parâmetros do sistema gerenciador do banco podem causar um impacto global no funcionamento da aplicação podendo causar mais prejuízos que benefícios (SHASHA,2003)..

Segundo MILENE (2007), 65% da causa de baixa performance em banco de dados ocorre no desenvolvimento de consultas, portanto só com a reescrita da consulta, grande parte dos problemas de lentidão já serão resolvidos.

Existem duas maneiras de identificar que uma consulta está executando mais lenta do que o normal:

- quando é verificado que a consulta está fazendo muito acesso ao disco;
- analisando o plano de execução da consulta pode ser verificado se a consulta está usando os índices corretamente, realizando acesso total à tabela, ou até mesmo um produto cartesiano que consome muitos recursos desnecessariamente.

Após a análise da consulta e correções necessárias, se mesmo assim o problema de performance não for resolvido, parte-se para as outras estratégias de *tuning.*

2.10. *TUNING* DE *TRIGGERS*

Uma *trigger* (gatilho) nada mais é que um procedimento armazenado em banco que é disparado como resultado de um evento. O *tuning* da *trigger* está diretamente ligado ao procedimento que é chamado. Um procedimento é desenvolvido utilizando SQL e linguagem de programação em banco, desta forma para otimizar o desenvolvimento da própria *trigger,* as regras para *tuning* de consulta podem ser aplicadas.

Uma *trigger* pode ser usada em três situações (SHASHA,2003):

- podem ser utilizadas para fazer auditoria em tabelas, ou seja, sempre que um registro for inserido, atualizado ou apagado é possível usar a *trigger* para salvar cada ação realizada em uma tabela de *log;*
- para manter a integridade dos dados. Por exemplo, quando os dados de uma tabela A são excluídos, e esta tabela possui dependentes na tabela B, pode ser criada uma *trigger* para eliminar todos dos registros dependentes da tabela B que ficaram órfãos[7];
- o terceiro caso que é o único que pode ser considerado o uso de *triggers* para *tuning.* Nesta situação a *trigger* é utilizada para responder aos eventos disparados pelas aplicações. Por exemplo, uma aplicação que tenha a necessidade de estar com os dados sempre atualizados, através da *trigger* o evento de atualização dos dados só será disparado quando houver modificação nos dados. Caso não fosse feito o uso de uma *trigger,* seria necessário realizar consultas periódicas ao banco de dados consumindo muito mais recursos.

[7] Refere-se aos registros que eram dependentes de um registro de uma tabela que foi excluído.

2.11. STORED PROCEDURES

Stored procedures (procedimentos armazenados em banco) são rotinas que possuem acesso direto ao banco de dados, geralmente armazenadas no dicionário de dados do sistema gerenciador de banco de dados (SHASHA,2003)..

Uma grande vantagem destas rotinas é quando se trabalha com aplicações desenvolvidas em diferentes plataformas, mas que acessam o mesmo banco de dados, independente da linguagem utilizada em cada plataforma todas elas utilizarão os mesmos procedimentos armazenados no banco de dados. Outra vantagem relacionada ao mesmo exemplo é em relação a quantidade de dados trafegada entre cliente e servidor. Como o *stored procedures* tem acesso direto ao banco de dados, todo o trabalho é realizado diretamente no servidor retornando somente o resultado final à aplicação. Neste modelo, há uma grande quantidade de dados que não precisa ser enviada ao cliente e vice versa otimizando o processo.

A segurança é um dos motivos para se usar *stored procedures*. Por exemplo, na documentação do MySQL são citados os bancos que usam funções e procedimentos armazenados em banco para qualquer operação do dia-a-dia. Este modelo de trabalho garante um ambiente consistente e seguro, e rotinas podem garantir que cada operação será registrada corretamente.

A maior desvantagem é que a linguagem utilizada nestes procedimentos é específica de cada fornecedor. Desta forma quando se fala de *tuning* de *stored procedures* é necessário focar em um fornecedor específico.

As linguagens utilizadas são específicas de cada SGBD, por exemplo, ao trabalhar com DB2 pode ser utilizado SQL PL ou Java, PostgreSQL usa-se PL/pgSQL, no Oracle é usado o PL/SQL, e assim por diante, cada banco com sua própria linguagem. Como cada linguagem tem seus benefícios próprios não é possível elaborar um manual de *tuning* geral para todos bancos, portanto para este trabalho serão abordadas as melhores práticas de programação para PL/SQL no banco de dados Oracle até a versão 11g release 2.

2.12. PL/SQL

PL/SQL é extensão procedural da Oracle do SQL, é uma linguagem de programação completamente portátil (portátil em relação ao sistema operacional) de alta performance e processo de transações (ORACLE, 2013).

A linguagem é completamente integrada com SQL. PL/SQL tem suporte completo aos tipos de dados do SQL, ou seja, não é necessário realizar conversão entre os tipos de dados utilizados no SQL e PL/SQL (ORACLE, 2013).

É uma linguagem de alta performance, pois permite o envio de diversas instruções ao banco de dados de uma só vez. Este processo reduz drasticamente o tráfego de dados entre o servidor e o cliente. É possível utilizar blocos de PL/SQL *e* rotinas (funções e procedimentos) para agrupar consultas (SQL) antes de mandá-las ao banco de dados para execução. As funções e procedimentos são compiladas uma vez e armazenadas no banco de dados de forma que só precisam ser chamadas pelas aplicações. Uma chamada de um procedimento pela aplicação através da rede pode desencadear grande processamento no servidor. Esta divisão de trabalho reduz o custo de tráfego de rede e melhora o tempo de resposta entre cliente e servidor (ORACLE, 2013).

PL/SQL também é uma linguagem de alta produtividade. Ela permite a escrita de códigos muito compactos para a manipulação de dados. Da mesma maneira que linguagens de *script*[8] como PERL[9] podem ler, manipular e escrever dados de arquivos, PL/SQL pode consultar, manipular e atualizar os dados do banco de dados (ORACLE, 2013).

Aplicações escritas com PL/SQL podem ser executadas em qualquer sistema operacional ou plataforma onde o banco de dados possa ser executado. Com PL/SQL é possível escrever bibliotecas e reutilizá-las em diferentes ambientes.

[8] Linguagens de *script* são voltadas à aplicações responsáveis por interligar duas aplicações. Elas trabalham com variáveis fracamente tipadas com objetivo de alcançar o desenvolvimento de aplicações com alto nível de programação buscando agilizar o desenvolvimento.(OUSTERHOUT, 1998)

[9] Linguagens como PERL não são utilizadas para desenvolver uma aplicação do zero, mas realizam a junção de outras aplicações. (OUSTERHOUT, 1998)

Estas características a tornam uma linguagem de alta portabilidade (ORACLE, 2013).

As rotinas desenvolvidas no banco implicam em total segurança, pois movem o código da aplicação do cliente para o servidor. Desta forma é possível proteger o código de adulterações, esconder detalhes internos, e restringir o acesso. Por exemplo, é possível dar ao usuário acessos à rotinas que atualizam uma determinada tabela, mas não à tabela e nem as instruções de como realizar a atualização (ORACLE, 2013).

Além de todos os benefícios da própria linguagem, o Oracle ainda possui diversas packages com diversas tarefas que facilitam o desenvolvimento das aplicações. Por exemplo, DBMS_ALERT para usar *triggers*, DBMS_FILE para ler e escrever arquivos de texto no sistema operacional, entre outras.

2.13. *TUNING* de PL/SQL

PL/SQL é uma linguagem de alto desempenho quando se trabalha com dados, mas para ter bons resultados é necessário saber como trabalhar com a linguagem.

O tuning de *PL/SQL* está diretamente ligado ao *tuning* de SQL, muitos problemas podem ser resolvidos simplesmente reescrevendo as consultas, no entanto a maneira como o código é escrito também pode ser otimizada.

Com a evolução do sistema gerenciador do banco de dados, a Oracle também evolui a linguagem de programação. A cada nova versão lançada, novas packages e novos recursos são disponibilizados pela Oracle para a otimização da ferramenta. Por exemplo, na versão 11 G *release 2*, foi adicionado o comando CONTINUE, que permite dar um sinal ao LOOP para finalizar a iteração atual e retornar à primeira instrução do LOOP. Outro recurso que já era esperado há muitas versões e também foi adicionado nesta versão é a possibilidade de incrementar o valor de uma sequência diretamente do PL/SQL.

Para o *tuning* de PL/SQL neste trabalho serão estudados os recursos disponibilizados pela Oracle até a versão 11g release 2 e serão apresentadas comparações com a maneira não otimizada para as mesmas funcionalidades.

O *tuning* de SQL tem um papel importante no desempenho da aplicação, mas a reescrita das consultas não faz parte do objetivo deste trabalho. Serão estudados recursos ligados à linguagem de programação apenas.

Nos próximos capítulos, serão descritas orientações para uso das melhores práticas de programação e recursos disponibilizados pela Oracle, pesquisados para otimizar o desenvolvimento de *stored procedures* com a linguagem PL/SQL.

3. RECURSOS PARA OTIMIZAR O DESEMPENHO

Para melhorar o desempenho do banco de dados, o Oracle conta com uma séries de recursos que vão desde parâmetros, *packages,* configurações, ferramentas para *self-tuning* e recursos que podem ser usados no desenvolvimento das aplicações.

Para cumprir com o objetivo deste trabalho serão estudados recursos do PL/SQL para otimizar o desenvolvimento da aplicação.

Abaixo estão citados os recursos escolhidos para serem estudados neste trabalho que podem otimizar o desempenho da aplicação:

- BULK COLLECT;
- FORALL;
- a importância da troca de contexto;
- a utilização do SQL dinâmico no Oracle;
- DBMS_SQL x NDS;
- utilização de variáveis BIND;
- o uso de cursores em PL/SQL;
- *hint* NOCOPY.

A seguir cada um deste tópicos será explicado detalhadamente.

3.1. BULK COLLECT

BULK COLLECT é usado quando há necessidade de retornar várias linhas de uma tabela em uma única consulta. Esta maneira de consulta aumenta significativamente a velocidade na busca dos dados.

Dado que PL/SQL é uma linguagem completamente integrada com SQL, então porque há a necessidade de recursos especiais para a execução instruções em SQL nos blocos de PL/SQL? A explicação para resposta está diretamente relacionada a como os motores de PL/SQL e SQL comunicam-se entre si através da troca de contexto (FEUERSTEIN, 2012).

3.1.1. CARACTERÍSTICAS DO BULK COLLECT

Para utilizar este recurso em um consulta, basta incluir BULK COLLECT antes de palavra-chave INTO, e após informar uma ou mais *collections* para retornar os dados, conforme pode ser visto na Figura 3.

Figura 3 - Consulta utilizando BULK COLLECT

```
1  DECLARE
2     TYPE rec_empregado IS RECORD
3     ( empregado_id   empregados.empregado_id%TYPE
4     ,  salario        empregados.salario%TYPE
5     );
6
7     TYPE tab_empregados IS TABLE OF rec_empregado;
8
9     v_empregados   tab_empregados;
10 BEGIN
11    SELECT empregado_id, salario
12      BULK COLLECT INTO v_empregados
13      FROM empregados
14     WHERE departamento_id = 10;
15 END;
```

É importante ter conhecimento das seguintes informações ao trabalhar com BULK COLLECT (FEUERSTEIN, 2012):

- os dados da consulta podem ser retornados para os três tipos de *collections* existentes: *associative arrays, nested tables* e *varrays;*
- os dados podem ser buscados para *collections* individuais, ou seja, uma *collection* para cada coluna do SELECT*;*

- a *collection* sempre é preenchida densamente, ou seja, partindo do índice um e incrementado o índice de um em um sem deixar falhas no índice;

- se nenhuma linha for buscada, a *collection* ficará vazia para todos os elementos.

É preciso ficar atento quando se está trabalhando com BULK COLLECT, pois as listas de valores são carregadas na memória. Para evitar um estouro de memória pode ser definido um limite de registros a serem carregados, e ao invés de usar uma única consulta como é mostrado na Figura 3, é possível usar a instrução FETCH-BULK COLLECT INTO como é mostrada na Figura 4.

Figura 4 - Instrução FETCH-BULK COLLECT INTO

```
1   DECLARE
2      v_limite PLS_INTEGER := 100;
3
4      CURSOR c_empregados IS
5         SELECT empregado_id
6           FROM empregados
7          WHERE departmento_id = 15;
8
9      TYPE tab_empregados_id IS TABLE OF empregados.id%TYPE;
10
11     v_empregados tab_empregados;
12  BEGIN
13     OPEN employees_cur;
14     LOOP
15        FETCH c_empregados
16        BULK COLLECT INTO v_empregados
17        LIMIT v_limite;
18
19        EXIT WHEN v_empregados.COUNT = 0;
20     END LOOP;
21  END;
```

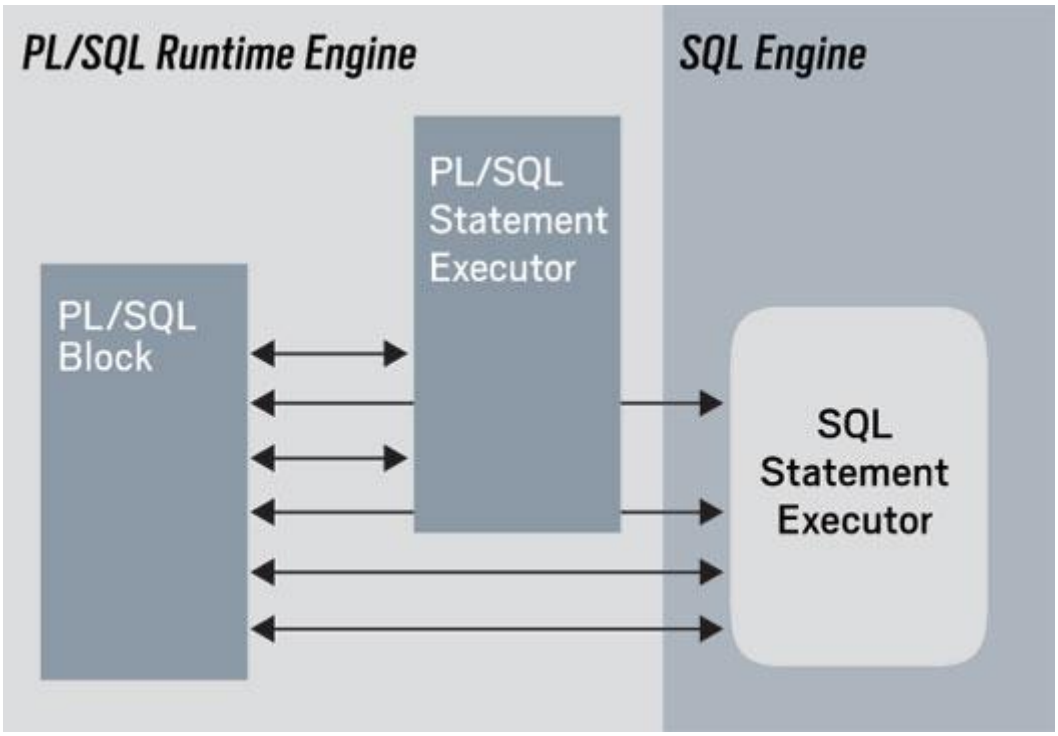
Como pode ser visto na Figura 4, com esta abordagem ao invés buscar todos os dados em uma única rajada, é aberto um cursor com as linhas que se quer buscar. Dentro do LOOP é usado o comando FETCH-BULK COLLECT INTO para buscar o número de linhas especificado pela variável v_limite. Desta forma não importa a quantidade de dados, serão buscados no máximo 100 registros, conforme

foi informado. Neste caso serão buscados de 100 em 100 registro até que todas as linhas do cursor sejam buscadas para e armazenadas na variável v_empregados.

3.1.2. TROCA DE CONTEXTO E PERFORMANCE

Em quase todos os programas, desenvolvedores de PL/SQL utilizam essa linguagem procedural em conjunto com o SQL. Instruções em PL/SQL são executadas por um executor de PL/SQL enquanto que instruções em SQL são executadas por um executor de SQL. Quando um programa está sendo executado, no momento em que o motor de PL/SQL encontra uma instrução SQL, ele para e passa as instruções para o motor de SQL, o motor de SQL executa as instruções e retorna ao motor de PL/SQL (Figura 5). Esta transferência de controle é chamada troca de contexto, e cada vez que ela ocorre há um custo adicional à aplicação que no final implica em uma significativa redução de desempenho (FEUERSTEIN, 2012).

Figura 5 - Troca de contexto



A figura mostra a troca de contexto entre o motor de PL/SQL e o motor de SQL na execução de instruções em SQL em um bloco PL/SQL (FEUERSTEIN,2012).

É necessário ficar atento que a troca de contexto também pode ocorrer na execução do próprio SQL quando são utilizadas funções PL/SQL diretamente no SQL, conforme mostra a Figura 6.

Figura 6 - Exemplo troca de contexto

```
1   FUNCTION LIMITE_SALARIO_DEPT (pi_departamento_id NUMBER)
2   RETURN NUMBER IS
3   .
4   .
5   .
6   END LIMITE_SALARIO_DEPT;
7
8   SELECT emp.empregado_id
9     FROM empregados emp
10   WHERE emp.salario < LIMITE_SALARIO_DEPT(emp.departamento_id)
11     AND emp.departamento_id = 10
```

Conforme pode ser visto na Figura 6, suponha que queira-se buscar todos os funcionários do departamento 10 para verificar quais receberão aumento. No entanto, é utilizada a função LIMITE_SALARIO_DEPT (implementação desta função está fora do contexto desta explicação) para retornar o teto salarial do departamento e selecionar apenas os funcionários que ainda não atingiram esta quantia. Para cada empregado retornado no SELECT, a função será executada para verificar se o salário do empregado é menor que o salário limite do departamento. Se tivesse 100 empregados no departamento, para cada empregado retornado ocorreria uma troca de contexto. Neste caso, teríamos 100 trocas de contexto. Por isso, é importante ficar atento a este tipo de situação.

3.2. FORALL

Da mesma maneira que BULK COLLECT permite retornar um grande volume de dados para uma *collection,* FORALL permite inserir, atualizar ou excluir estes mesmos dados através de uma única instrução (FEUERSTEIN, 2012).

O uso de FORALL juntamente com BULK COLLECT reduz drasticamente o custo para a aplicação, pois o principal objetivo destes recursos é reduzir o número de troca de contexto entre os motores de PL/SQL *e* SQL (FEUERSTEIN, 2012).

3.2.1. CARACTERÍSTICAS DO FORALL

A instrução FORALL não é um LOOP, é uma instrução declarada dentro do motor de PL/SQL. Esta instrução gera um pacote com todas as instruções DML que deveriam ser executadas linha por linha e envia todas de uma vez para o motor de SQL realizando apenas uma troca de contexto (FEUERSTEIN, 2012).

Como pode ser visto na Figura 7, o cabeçalho da instrução FORALL se parece com um FOR LOOP numérico, no entanto, as palavras-chave LOOP ou END LOOP não são utilizadas.

Figura 7 - Exemplo Instrução FORALL

```
1   DECLARE
2      TYPE tab_empregados_id IS TABLE OF empregados.empregado_id%TYPE
3      INDEX BY PLS_INTEGER;
4
5      v_empregados_id    tab_empregados_id;
6   BEGIN
7      v_empregados_id := RETORNA_EMPREGADOS;
8
9      FORALL r_emp IN 1 .. v_qualificados_id.COUNT
10        UPDATE empregados emp
11           SET emp.salario = emp.salario + emp.salario * 0.05
12         WHERE emp.empregado_id = v_empregados_id(r_emp);
13  END;
```

É importante ter conhecimento das seguintes informações ao trabalhar com FORALL (FEUERSTEIN, 2012):

- cada instrução FORALL pode contar uma única instrução DML. Se o LOOP contém dois UPDATES e um INSERT, será necessário escrever três instruções FORALL;
- o motor de PL/SQL declara a variável utilizada para iterar a instrução FORALL como inteiro, assim quando se trabalhar com as instruções FOR LOOP, não há necessidade de declarar esta variável;

- para utilizar a instrução FORALL em pelo menos um lugar da instrução DML deve ser utilizada uma referência à *collection* que a instrução FORALL está iterando, como por exemplo, na linha 12 da Figura 7;
- quando usar a sintaxe IN menor_valor..maior_valor, no cabeçalho da instrução FORALL, o índice deve ser denso, ou seja, o índice deve iterar de um em um do menor valor ao maior valor;
- se a *collection* utilizada possuir um índice do tipo esparso, ou seja, não segue do menor valor ao maior valor sequencialmente, é necessário utilizar a sintaxe INDICE OF ou VALUES OF no cabeçalho do FORALL.

3.2.2. TRATAMENTO DE ERROS DML COM FORALL

Ao inserir mil linhas em uma tabela, se na linha 901 ocorresse um erro, as demais linhas não seriam inseridas. Para evitar este tipo de problema é possível tratar as exceções nas linhas que estiverem com problema, inserir tantas linhas quanto for possível e retornar as exceções para aplicação após a execução do FORALL (FEUERSTEIN, 2012).

Para salvar os erros, é necessário utilizar a condição SAVE_EXCEPTIONS no cabeçalho do comando FORALL. Após a execução, iterar o vetor de exceções SQL SQL%BULK_EXCEPTIONS para receber detalhes dos erros, conforme a Figura 8.

Figura 8 - Exceções com FORALL

```
1  BEGIN
2      FORALL r_emp IN 1 .. v_qualificados_id.COUNT SAVE EXCEPTIONS
3          UPDATE empregados emp
4             SET emp.salario = emp.salario + emp.salario * pi_perc_aumento
5           WHERE emp.empregado_id = v_qualificados_id(r_emp);
6  EXCEPTION
7      WHEN OTHERS THEN
8          IF SQLCODE = -24381 THEN
9             FOR r_erros IN  1..SQL%BULK_EXCEPTIONS.COUNT LOOP
10               DBMS_OUTPUT.put_line (
11                  SQL%BULK_EXCEPTIONS (r_erros).ERROR_INDEX
12                  || ': '
13                  || SQL%BULK_EXCEPTIONS (r_erros).ERROR_CODE);
14            END LOOP;
15         ELSE
16            RAISE;
17         END IF;
18 END;
```

Como pode ser visto na Figura 8, na linha 2 foram adicionadas as palavras-chave SAVE EXCEPTIONS para que o Oracle entenda que é necessário salvar as exceções.

Nas linhas 6-15 é realizado o tratamento no caso de ocorrer algum erro DML durante execução do FORALL. Neste caso a exceção -24381 é tratada e os erros são armazenados no vetor SQL%BULK_EXCEPTIONS.

3.3. OTIMIZAÇÃO COM BULK COLLECT E FORALL

Para explicar como este recursos reduzem o custo de execução em uma aplicação vamos utilizar um exemplo concreto. Supondo que foi solicitado o desenvolvimento de um procedimento em PL/SQL que identificasse todos os funcionários identificados pelo ID do departamento e desse a eles um aumento de salário identificado por um percentual específico, seria possível desenvolver o procedimento mostrado na Figura 9 (FEUERSTEIN, 2012).

Figura 9 - Exemplo procedimento AUMENTAR_SALARIO

```
1  PROCEDURE AUMENTAR_SALARIO ( pi_departamento_id   IN empregados.departamento_id
2                             , pi_perc_aumento      IN NUMBER
3                             )
4  IS
5  BEGIN
6     FOR r_emp IN ( SELECT empregado_id
7                      FROM empregados
8                     WHERE departmento_id = pi_departmento_id
9                  )
10    LOOP
11       UPDATE empregados emp
12          SET emp.salario = emp.salario + emp.salario * pi_perc_aumento
13        WHERE emp.empregado_id = r_emp.empregado_id;
14    END LOOP;
15 END AUMENTAR_SALARIO;
16
17 BEGIN
18    AUMENTAR_SALARIO (15, 0.10);
19 END;
```

Conforme mostrado na Figura 9, nas linhas 6 a 10 é utilizado um cursor FOR LOOP que busca todos os empregados do departamento especificado pelo parâmetro da linha 1. Para cada empregado retornado na consulta, nas linhas 11 a 13, a seu salário é somado o aumento informado no parâmetro da linha 2.

Supondo que existem 100 empregados no departamento 15, quando o bloco de código na linha 17 da Figura 9 é executado, o motor de PL/SQL fará 100 trocas de contexto para o motor de SQL, uma para cada funcionário atualizado. Tom Kyte, do AskTom (asktom.oracle.com), refere-se a cada troca de contexto por linha com um "*slow-by-slow processing*", interpretado como um processamento lento (FEUERSTEIN, 2012).

Sempre que for desenvolver um procedimento em PL/SQL para evitar trocas de contexto desnecessárias, é importante realizar o máximo do trabalho possível apenas com SQL e usar ambos somente quando não houver outra opção (FEUERSTEIN, 2012).

Como pode ser visto na Figura 10, esta outra implementação do procedimento AUMENTAR_SALARIO onde a instrução SELECT identifica todos os empregados no departamento informado, e a instrução de UPDATE executa o mesmo percentual de aumento para cada um dos empregados. Este é um cenário onde apenas com uma instrução SQL é possível resolver o problema evitando o uso

desnecessário de um cursor FOR LOOP, assim utilizando apenas uma troca de contexto.

Figura 10 - Exemplo procedimento AUMENTAR_SALARIO melhorado

```
PROCEDURE AUMENTAR_SALARIO ( pi_departamento_id   IN empregados.departamento_id
                           , pi_perc_aumento      IN NUMBER
                           )
IS
BEGIN
    UPDATE empregados emp
      SET emp.salario = emp.salario + emp.salario * pi_perc_aumento
     WHERE emp.empregado_id = r_emp.empregado_id;
END AUMENTAR_SALARIO;
```

O exemplo utilizado acima é apenas uma demonstração de como otimizar através da troca de contexto. Em um cenário do mundo real uma implementação normalmente não é tão simples. Geralmente existem etapas a serem executadas antes de atualizar dos dados através das instruções DML[10].

No próximo exemplo, conforme Figura 11, suponhamos que antes de aumentar o salário do empregado é necessário verificar sua qualificação. O aumento será aplicado apenas aos empregados considerados dignos e, em seguida, será enviado um e-mail de notificação ao gerente.

[10] Instruções de manipulação de dados - *INSERT,UPDATE* e *DELETE*.

Figura 11 - Exemplo procedimento AUMENTAR_SALARIO com etapa adicional

```
1  PROCEDURE AUMENTAR_SALARIO ( pi_departamento_id   IN empregados.departamento_id
2                             , pi_perc_aumento      IN NUMBER
3                             )
4  IS
5     v_qualificado BOOLEAN;
6  BEGIN
7     FOR r_emp IN ( SELECT empregado_id
8                      FROM empregados
9                     WHERE departmento_id = pi_departmento_id
10                  )
11    LOOP
12       VERIFICAR_QUALIFICACAO( r_emp.empregado_id
13                             , pi_perc_aumento
14                             , v_qualificado
15                             );
16       IF v_qualificado THEN
17          UPDATE empregados emp
18             SET emp.salario = emp.salario + emp.salario * pi_perc_aumento
19           WHERE emp.empregado_id = r_emp.empregado_id;
20       END IF;
21    END LOOP;
22 END AUMENTAR_SALARIO;
```

No exemplo acima não é mais possível executar uma instrução SQL para aplicar o salário a todos os funcionários, no entanto através dos recursos BULK COLLECT E FORALL é possível evitar a troca de contexto.

No exemplo, exibido na Figura 12, é explicado como trabalhar com DMLs e não ter o ônus da troca de contexto.

Figura 12 - Exemplo procedimento AUMENTAR_SALARIO com etapa adicional otimizado.

```
1  PROCEDURE AUMENTAR_SALARIO ( pi_departamento_id   IN empregados.departamento_id
2                             , pi_perc_aumento      IN NUMBER
3                             )
4  IS
5     TYPE tab_empregados_id IS TABLE OF empregados.empregado_id%TYPE
6     INDEX BY PLS_INTEGER;
7
8     v_empregados_id    tab_empregados_id;
9     v_qualificados_id  tab_empregados_id;
10
11    v_qualificado BOOLEAN;
12 BEGIN
13    SELECT empregado_id
14      BULK COLLECT INTO v_empregados_id
15      FROM empregados
16     WHERE departmento_id = pi_departmento_id;
17
18    FOR r_emp IN 1..v_empregados_id.COUNT
19    LOOP
20       VERIFICAR_QUALIFICACAO( v_empregados_id(r_emp)
21                             , pi_perc_aumento
22                             , v_qualificado
23                             );
24       IF v_qualificado THEN
25          v_qualificados_id(v_qualificados_id.COUNT +1) :=
26          v_empregados_id(r_emp);
27       END IF;
28    END LOOP;
29
30    FORALL r_emp IN 1 .. v_qualificados_id.COUNT
31       UPDATE empregados emp
32          SET emp.salario = emp.salario + emp.salario * pi_perc_aumento
33        WHERE emp.empregado_id = v_qualificados_id(r_emp);
34
35 END AUMENTAR_SALARIO;
```

Nas linhas 5-9, são declarados um tipo tabela aninhada e duas variáveis deste tipo. A variável v_empregados_id irá armazenar todos os empregados do departamento. A variável v_qualificados_id irá armazenar todos os empregados qualificados para receber o aumento.

Nas linhas 13-16, a instrução BULK COLLECT é utilizada para buscar o id de todos os empregados do departamento especificado e armazenar na *collection* v_empregados_id.

Nas linhas 18-28, é verificado se o empregado é qualificado para receber o aumento, se não for, um e-mail de notificação é enviado ao gerente. Caso seja, o id do empregado é armazenado na *collection* v_qualificados_id (a implementação do

procedimento VERIFICAR_QUALIFICACAO não está incluída neste exemplo e não faz parte do contexto estudado).

Nas linhas 30-33 a instrução FORALL é utilizada para atualizar o salário de todos os empregados identificados pelo id armazenado na *collection* v_qualificados_id.

Como pôde ser visto na Figura 12, foram necessárias 3 etapas para aplicar o aumento de salário.

Foram buscados todos os empregados do departamento em uma única troca de contexto através da instrução BULLK COLLECT.

Na segunda etapa os empregados qualificados foram selecionados.

Na terceira etapa através da instrução FORALL, ao invés de atualizar linha a linha, esta instrução "empacota" todos os registros para serem passados em uma única instrução para o motor de SQL atualizar os registros no banco de dados. Desta maneira apenas duas trocas de contexto são necessárias resultando em um significativo aumento de performance em relação à Figura 11.

3.4. NDS (*NATIVE DYNAMIC SQL*)

Nos últimos anos, o uso de SQL dinâmico se tornou bem comum na programação em PL/SQL. Antes da chegada do NDS na versão 10g, a package DMBS_SQL era encarregada de executar o SQL dinâmico, no entanto ela é mais lenta e mais complicada de se usar (FEUERSTEIN, 2004).

SQL Dinâmico refere-se à instruções DDL[11], DCL[12], DML e consultas que são construídas, analisadas e executadas em tempo de execução. É chamado de SQL dinâmico, pois a instrução que se quer executar ainda não é completamente conhecida no momento do desenvolvimento código. Na maioria das vezes estas

[11] DDL*(Data definition Language)* - Instruções para modificar a estrutura das tabelas, como por exemplo a instrução ALTER TABLE.
[12] DCL(*Data Control Language)* - Instruções para dar/restringir acesso aos recursos do banco. Por exemplo a instrução GRANT.

informações são passadas pelo usuário na execução do programa (FEUERSTEIN, 2004).

3.4.1. CARACTERÍSTICAS DO NDS

O NDS pode ser necessário em três situações no desenvolvimento de uma aplicação em PL/SQL (FEUERSTEIN, 2004):

- quando há necessidade de executar uma instrução SQL para definição de dados (CREATE), uma instrução para controle de dados (GRANT), ou uma instrução para controle de sessão (ALTER SESSION). Nenhuma destas instruções pode ser executada estaticamente no PL/SQL;
- quando há a necessidade de mais flexibilidade. Por exemplo, caso queira-se desenvolver um programa com diferentes condições de pesquisa na cláusula WHERE de uma instrução SELECT. Um programa mais complexo poderia escolher até mesmo diferentes SQLs, condições de pesquisa, etc;
- quando o programa já havia sido desenvolvido utilizando a package DBMS_SQL em uma versão anterior do Oracle e o objetivo é a melhora de desempenho da aplicação.

É possível executar NDS através da instrução EXECUTE IMMEDIATE ou através das instruções OPEN-FOR, FETCH e CLOSE.

A instrução EXECUTE IMMEDIATE será usada para DDL, DML, DCL, controle de sessão e instruções SELECT que retornem apenas um registro. A instrução OPEN-FOR, FETCH e CLOSE para instruções SELECT que retornem mais de uma linha.

3.4.2. EXECUTE IMMEDIATE

A instrução EXECUTE IMMEDIATE verifica se a instrução SQL ou PL/SL é um SQL válido e em seguida executa instruções de SQL dinâmico ou blocos de PL/SQL.

A Figura 13 mostra um exemplo de execução de um SQL dinâmico como era utilizando antes da versão 10g.

Figura 13 - Exemplo SQL dinâmico utilizando a package DBMS_SQL

```
1  CREATE PROCEDURE INSERE_NA_TABELA (
2                    pi_nome_tabela  VARCHAR2
3                  , pi_num_dept     NUMBER
4                  , pi_nome_dept    VARCHAR2
5                  , pi_local        VARCHAR2
6                                     ) IS
7      v_cursor          INTEGER;
8      v_sql             VARCHAR2(200);
9      v_cont_linhas     BINARY_INTEGER;
10
11 BEGIN
12     v_sql := 'INSERT INTO ' ||
13       v_nome_tabela || ' VALUES
14       (:deptno, :dname, :loc)';
15
16     -- abre o cursor
17     v_cursor := dbms_sql.open_cursor;
18
19     -- analisa cursor
20     dbms_sql.parse(v_cursor, v_sql,
21       dbms_sql.native);
22
23     -- fornece valores para variáveis bind
24     dbms_sql.bind_variable
25       (v_cursor, ':deptno', pi_num_dept);
26     dbms_sql.bind_variable
27       (v_cursor, ':dname', pi_nome_dept);
28     dbms_sql.bind_variable
29       (v_cursor, ':loc', pi_local);
30
31      -- executa cursor
32      v_cont_linha :=
33      dbms_sql.execute(v_cursor);
34
35      -- fecha cursor
36      dbms_sql.close_cursor(v_cursor);
37 END;
```

Conforme pode ser visto na Figura 13:

- nas linhas 12-14 a string com o SQL o código com a instrução DML é atribuído à variável v_sql;
- na linha 17 o cursor é aberto;
- nas linhas 19-20 é realizado o *parse* verificando se a string da variável v_sql é um SQL válido;
- nas linhas 24-29 os valores das variáveis BIND utilizadas na instrução SQL são mapeados;
- nas linhas 32-33 o cursor é executado;
- na linha 36 o cursor é fechado.

Este mesmo comando pode ser executado utilizando NDS de uma forma muito mais simples como é mostrado na Figura 14.

Figura 14 - Exemplo SQL dinâmico utilizando NDS

```
1  CREATE PROCEDURE INSERE_NA_TABELA (
2                    pi_nome_tabela  VARCHAR2
3                  , pi_num_dept     NUMBER
4                  , pi_nome_dept    VARCHAR2
5                  , pi_local        VARCHAR2
6                                     ) IS
7      v_sql             VARCHAR2(200);
8  BEGIN
9      v_sql := 'INSERT INTO ' ||
10       v_nome_tabela || ' VALUES
11       (:deptno, :dname, :loc)';
12     EXECUTE IMMEDIATE v_sql
13     USING pi_num_dept
14         , pi_nome_dept
15         , pi_local;
16 END;
17
```

A Figura 14 realiza a mesma funcionalidade de Figura 13, no entanto com muito mais praticidade e rapidez. Nas linhas 9-11, a *string* com o SQL é montada e nas linhas 12-15, o SQL é executado. A instrução USING faz o mapeamento das variáveis BIND utilizadas no SQL.

3.4.3. OPEN-FOR, FETCH e CLOSE

As instruções OPEN-FOR, FETCH e CLOSE são utilizadas para processar consultas que retornam mais de uma linha. Primeiro é utilizada a instrução OPEN para abrir uma variável de cursor para uma consulta SQL. Em seguida com a instrução FETCH, as linhas são buscadas de uma única vez. Após o processamento de todas as linhas é utilizada a instrução CLOSE para fechar o cursor (FEUERSTEIN, 2004).

Da mesma maneira que o EXECUTE IMMEDIATE, as instruções OPEN-FOR, FETCH e CLOSE também podem ser executadas através da package DBMS_SQL, conforme mostra a Figura 15.

Figura 15 - Exemplo consulta dinâmica utilizando a *package* DBMS_SQL.

```
1  DECLARE
2    v_sql varchar2(200);
3    v_cursor int;
4    v_cont_linhas int;
5    v_nome varchar2(10);
6    v_salario int;
7  BEGIN
8    -- Abre o cursor
9    v_cursor := dbms_sql.open_cursor;
10   -- Monta SQL
11   v_sql := 'SELECT nome, salario FROM empregados WHERE
12   cargo = :cargo';
13   dbms_sql.parse(v_cursor, v_sql, dbms_sql.native);
14   -- Fornece variáveis bind
15   dbms_sql.bind_variable(
16   v_cursor, 'cargo', 'Vendedor');
17   -- Descreve defingção das variáveis
18   dbms_sql.define_column(v_cursor, 1, v_nome, 200);
19   dbms_sql.define_column(v_cursor, 2, v_salario);
20   -- Executa cursor
21   v_cont_linhas := dbms_sql.execute(v_cursor);
22   LOOP
23     -- Se conseguiu buscar linha
24     IF dbms_sql.fetch_rows(v_cursor) > 0 then
25       -- busca colunas das linhas
26       dbms_sql.column_value(v_cursor, 1, v_nome);
27       dbms_sql.column_value(v_cursor, 2, v_salario);
28       -- processa dados
29     ELSE
30       -- sai do cursor
31       EXIT;
32     END IF;
33   END LOOP;
34   -- close cursor
35   dbms_sql.close_cursor(cur_hdl);
36 END;
```

A diferença entre as figuras 13 e 15 são as instruções DBMS_SQL.DEFINE_COLUMN, linhas 18-20, onde são definidas as variáveis que irão retornar as colunas da consulta. A instrução DBMS_SQL.FETCH_ROWS (v_cursor), linha 24, que verifica se ainda existem linhas a serem buscadas no cursor. E por último, a instrução DBMS_SQL_COLUMN_VALUE que busca o valor da coluna para cada variável mapeada.

O mesmo código pode ser desenvolvido mais facilmente através do NDS, assim como mostra a Figura 16.

Figura 16 - Exemplo consulta dinâmica utilizando NDS

```
DECLARE
  TYPE cursor_empregado IS REF CURSOR;
  c_emp cursor_empregado;
  v_sql varchar2(200);
  v_nome varchar2(10);
  v_salario int;
BEGIN
   -- Monta SQL
   v_sql := 'SELECT nome, salario FROM empregados WHERE
   cargo = :1';
   -- Abre o cursor
   OPEN c_emp FOR v_sql USING 'Vendedor';
   LOOP
      -- Busca valor das colunas para variáveis
      FETCH c_emp INTO v_nome, v_salario;
      EXIT WHEN c_emp%NOTFOUND;
      -- processa dados
   END LOOP;
   -- Fechar cursor
   CLOSE c_emp;
END;
```

A figura mostra a execução de consulta dinâmica utilizando a package NDS.

Da mesma maneira na Figura 15, todos os empregados onde o cargo seja "Vendedor" são buscados, no entanto é muito menos complicado trabalhar com NDS do que com a package DBMS_SQL. Na Figura 16, é utilizado um tipo chamado REF CURSOR que aponta para um cursor. As linhas são buscadas para as variáveis através das instruções FETCH-INTO, e o LOOP encerra quando o cursor não retornar mais dados, sinalizado através da variável c_emp%NOTFOUND.

3.4.4. DBMS_SQL X NDS

O NDS surgiu no Oracle 10g para substituir a package DBMS_SQL. É muito mais fácil escrever código neste modelo e muito mais performático. Sempre que possível deve-se utilizar NDS. No entanto existem algumas situações específicas onde a utilização da package DBMS_SQL se faz necessária. Os exemplos abaixo encaixam-se nestas exceções (FEUERSTEIN, 2004):

- quando for necessário executar um SQL dinâmico que ultrapasse os 32 mil caracteres: a instrução EXECUTE IMMEDIATE analisa e executa apenas strings, o que significa que é limitado aos 32 mil caracteres. Nestes casos, é necessário utilizar a instrução DBMS_SQL.PARSE, que aceita coleções de strings e analisa a instrução formada de uma concatenação de todas as strings;

- quando é necessário executar o método 4 do SQL dinâmico. O método 4 é o mais complexo e genérico formato de SQL dinâmico. Ele é apropriado quando não se conhece alguns dos seguintes itens quando se está escrevendo o código: quantas colunas são retornadas em uma consulta, ou quantas variáveis BIND estão sendo usadas na string. Embora seja possível implementar o método 4 com NDS, é muito mais simples utilizar a package DBMS_SQL;

- quando é necessário diminuir o número de parses na instrução SQL: uma desvantagem do NDS é que a instrução SQL será analisada cada vez que for executada, mesmo que a mesma instrução já tenha sido analisada momentos atrás. Se a instrução SQL for muito complicada, a sobrecarga pode ser significativa para a performance. Com DBMS_SQL é possível passar a fase de parse e simplesmente reexecutar a mesma instrução com diferentes variáveis BIND.

3.4.5. *BINDING VARIABLES* (VARIAVEIS DE LIGAÇÃO)

As figuras 17 e 18 executam a mesma funcionalidade, a diferença é que na primeira o id do departamento foi concatenado diretamente na *string*, enquanto que na Figura 18 foi utilizada uma variável BIND para armazenar o id do departamento.

Figura 17 - Exemplo EXECUTE IMMEDIATE

```
1  DECLARE
2     v_sql VARCHAR2(1000);
3     v_departamento_id NUMBER;
4     v_nro_empregados NUMBER;
5  BEGIN
6     v_departamento_id := 10;
7     v_sql := 'SELECT COUNT(1) FROM empregados WHERE departamento_id = '
8     ||v_departamento_id||';
9     EXECUTE IMMEDIATE v_sql
10    INTO v_nro_empregados;
11
12    DBMS_OUTPUT.PUT_LINE('Nro empregados: '||v_nro_empregados);
13 END;
```

Conforme pode ser visto na Figura 17, a instrução é uma *string* atribuída à variável v_sql, na linha 7. Através da instrução EXECUTE IMMEDIATE, linha 9, o conteúdo da variável v_sql é analisado e executado. A variável v_nro_empregados, linha 10, será utilizada para armazenar o retorno do SELECT.

Figura 18 - Exemplo EXECUTE IMMEDIATE com variável BIND

```
1  DECLARE
2     v_sql VARCHAR2(1000);
3     v_departamento_id NUMBER;
4     v_nro_empregados NUMBER;
5  BEGIN
6     v_departamento_id := 10;
7     v_sql := 'SELECT COUNT(1) FROM empregados WHERE departamento_id = :1';
8     EXECUTE IMMEDIATE v_sql
9     INTO v_nro_empregados
10    USING v_departamento_id;
11
12    DBMS_OUTPUT.PUT_LINE('Nro empregados: '||v_nro_empregados);
13 END;
```

No código descrito na Figura 17, o id departamento foi concatenado na string, sempre que o valor da variável v_departamento_id mudar, um novo SELECT terá que ser analisado na *System Global Area*[13] (SGA) do banco de dados. Se o id do departamento mudasse 100 vezes dentro de um LOOP, seriam 100 SELECT diferentes analisados e carregados na SGA. Ao utilizar variáveis BIND, Figura 18,

[13]É um grupo de áreas com memória compartilhada dedicada à uma "instância" do Oracle(uma instância é composta dos programas do banco de dados e memória RAM). Todos os processos utilizam a SGA para armazenar informações sobre dados e controles internos realizados pelo banco de dados.

um parâmetro é referenciado no SELECT e seu valor é passado através da instrução USING. Este SELECT será carregado somente uma vez na SGA e sempre que for executado novamente não será necessário a realização do parse (análise). Desta forma, um SELECT executado 100 vezes seria analisado e carregado uma vez na SGA e nas outras 99 vezes seria apenas executado economizando muito tempo de *parse*.

Quando se trabalha com NDS o recomendado é sempre que possível utilizar variáveis BIND (FEUERSTEIN, 2004).

3.5. CURSORES

Estruturas de cursores retornam o resultado de instruções SELECT. No PL/SQL é possível processar linha por linha ou instruções em massa. Existem dois tipos de cursores: implícitos e explícitos. Um cursor é explicito quando é definido dentro na declaração de um bloco. Instruções DML são exemplo de cursores implícitos (MCLAUGHLIN, 2008). Também é possível utilizar consultas dinâmicas com ambos cursores. SQL dinâmico significa que no desenvolvimento do código ainda não se tem toda a informação necessária. Portanto, é necessário em tempo de execução completar as informações necessárias antes de analisar e executar o SELECT (FEURSTEIN, 2013).

Nesta seção são estudados os cursores implícitos, explícitos e cursores dinâmicos e em seguida será feita uma análise em que situação cada cursor deve ser utilizado.

3.5.1. CURSOR IMPLÍCITO

Cada instrução SQL em um bloco de PL/SQL é um cursor implícito (MCLAUGHLIN, 2008). A instrução SELECT-INTO oferece a mais simples e rápida maneira de buscar dados de uma única linha de uma instrução SELECT (FEURSTEIN, 2013).

O cursor FOR LOOP é uma elegante e natural extensão do FOR LOOP numérico em PL/SQL. Com um FOR LOOP numérico, o corpo do LOOP executa uma vez para cada valor inteiro entre o menor e o maior valor especificado no intervalo. Com um cursor FOR LOOP, o corpo do LOOP é executado para cada linha retornada pela consulta (FEURSTEIN, 2013). Na Figura 19 abaixo, pode ser visualizado como usar um cursor implícito para retornar todos os registros da tabela empregados que trabalham no departamento 10.

Figura 19 - Exemplo de cursor implícito

```
1  BEGIN
2      FOR r_emp IN (
3          SELECT *
4            FROM empregados emp
5           WHERE emp.departamento_id = 10
6           )
7      LOOP
8        DBMS_OUTPUT.PUT_LINE('Empregado: '||r_emp.codigo||
9                             ' Nome: '||r_emp.nome);
10     END LOOP;
11 END;
```

Conforme pode ser visto na Figura 19:

- nas linhas 2-7, implicitamente o cursor é aberto e busca todos os registros da consulta. Para cada linha retornada da consulta é declarado um registro com o tipo do cursor (r_emp);
- nas linhas 7-10, linha a linha é carregada para o registro r_emp até que não existam mais linhas a serem retornadas pela consulta. E em seguida o Oracle fecha o cursor.

O melhor de tudo é que o Oracle automaticamente otimiza o cursor FOR LOOP para realizar uma busca similar à utilizada nas consulta através da instrução BULK COLLECT. Então, mesmo que pareça que a busca está sendo realizada linha a linha, o banco de dados irá buscar 100 linhas de uma vez e possibilitar que se trabalhe individualmente com cada uma (FEURSTEIN, 2013).

3.5.2. CURSOR EXPLÍCITO

Um cursor explícito é aquele que é declarado, aberto, busca os dados e então é fechado (FEURSTEIN, 2013).

Conforme Figura 20, suponha que seja necessário escrever um bloco de código que busque todos os empregados do departamento 10 em ordem crescente de salários. Para cada empregado retornado na consulta será aplicado um bônus através do procedimento APLICAR_BONUS. A cada chamada do procedimento, o bônus atribuído ao funcionário é descontado do total disponível. Quando a quantidade disponível para o bônus terminar, o cursor será encerrado e irá salvar as alterações.

Figura 20 - Exemplo de cursor explícito

```
1  PROCEDURE APLICAR_BONUS( pi_emp_id IN empregados.id%TYPE
2                         , pio_bonus IN OUT INTEGER
3                         )
4  IS
5  BEGIN
6  .
7  .
8  END APLICAR_BONUS;
9
10 DECLARE
11    v_total INTEGER := 10000;
12    CURSOR c_emp IS
13       SELECT empregado_id
14         FROM empregados emp
15     ORDER BY emp.salario;
16    v_emp_id c_emp%ROWTYPE;
17 BEGIN
18    OPEN c_emp;
19
20    LOOP
21       FETCH c_emp INTO v_emp_id;
22       EXIT WHEN c_emp%NOTFOUND;
23
24       APLICAR_BONUS(v_emp_id,v_total);
25
26       EXIT WHEN v_total <= 0;
27    END LOOP;
28
29    CLOSE c_emp;
30    COMMIT;
31 END;
```

Conforme pode ser visto na Figura 20:

- nas linhas 1-8 é mostrado o cabeçalho do procedimento APLICAR_BONUS que será usado no bloco logo em seguida;
- na linha 12 é declarada a variável com o total de bônus disponível;
- nas linhas 12-15 o cursor é declarado e a instrução SELECT é movida para a declaração do bloco;
- na linha 16 é declarado uma variável do tipo do cursor;
- na linha 18 o cursor é aberto para que em seguida possa buscar os registros da consulta;
- na linha 21 inicia a buscar os registro e retornar para a variável v_emp_id;
- na linha 22 se nenhuma linha for retornada da busca irá sinalizar no atributo do cursor %NOTFOUND e encerrar o LOOP;
- na linha 24 chama o procedimento APLICAR_BONUS e decrementa o total disponível;
- na linha 26 encerra o cursor e para de buscar empregados quando o total disponível para bônus for menor ou igual a zero;
- na linha 29 fecha o cursor;
- na linha 30 salva as alterações no banco de dados.

Conforme FEURSTEIN (2013), é preciso ficar atento aos seguintes itens ao se trabalhar com cursores explícitos:

- se a consulta não buscar nenhuma linha, o banco de dados Oracle não irá disparar o erro NO_DATA_FOUND. Ao invés disto, o atributo nome_do_cursor%NOT_FOUND irá retornar TRUE;
- se a consulta retornar mais de uma linha, o banco de dados não irá disparar o erro TOO_MANY_ROWS;

- quando um cursor é declarado em um package e o cursor é aberto, ele ficará aberto até que seja fechado ou a sessão seja encerrada;

- quando um cursor é declarado na sessão de declarações de um bloco PL/SQL, o Oracle automaticamente fecha o cursor quando o bloco terminar a execução. No entanto, ainda é uma boa ideia fechar o cursor. Caso ele seja movido para uma package, será necessário fechá-lo, e por não ter sido fechado, poderá gerar problemas futuros.

3.5.3. SQL DINÂMICO

SQL dinâmico se faz necessário quando ainda não se tem toda a informação necessária no desenvolvimento do código. Portanto, é necessário em tempo de execução completar as informações necessárias antes de analisar e executar o SELECT (FEURSTEIN, 2013).

É possível executar uma consulta dinâmica através da instrução EXECUTE IMMEDIATE vista no capítulo 3.4.2 através dos cursores explícitos.

Quando é utilizada a instrução EXECUTE IMMEDIATE é necessária a utilização de uma COLLECTION para retornar os registros da consulta, conforme pode ser visto na Figura 21.

Figura 21 - Exemplo EXECUTE IMMEDIATE com consulta dinâmica

```
1   DECLARE
2      TYPE tab_numeros IS TABLE OF NUMBER;
3
4      v_emp_ids tab_numeros
5   BEGIN
6      EXECUTE IMMEDIATE
7            'SELECT empregado_id
8               FROM empregados'
9         BULK COLLECT INTO v_emp_ids;
10
11     FOR r_emp IN 1 .. v_emp_ids.COUNT
12     LOOP
13        DBMS_OUTPUT.put_line
14        (v_emp_ids (c_emp));
15     END LOOP;
16  END;
```

Conforme mostra a Figura 21, a instrução EXECUTE IMEMDIATE busca todos os registros da tabela empregados e através da instrução BULK COLLECT insere os registros na COLLECTION.

O mesmo exemplo pode ser escrito utilizando um cursor explícito, conforme mostra a Figura 22.

Figura 22 - Exemplo de cursor explícito com consulta dinâmica

```
1   DECLARE
2      c_emp  SYS_REFCURSOR;
3      v_emp_id empregados.id%TYPE;
4      v_query  VARCHAR2(4000);
5   BEGIN
6      v_query := 'SELECT empregado_id
7                  FROM empregados';
8      OPEN c_emp FOR v_query;
9      LOOP
10        FETCH c_emp INTO v_emp_id;
11        EXIT WHEN c_emp%NOTFOUND;
12        DBMS_OUTPUT.put_line(v_emp_id);
13     END LOOP;
14     CLOSE c_emp;
15  END;
```

Conforme pode ser visto na Figura 22:

- na linha 2 é declarada uma variável do tipo SYS_REFCURSOR. SYS_REFCURSOR é um sistema de referência fracamente tipado. Um cursor fracamente tipado pode assumir qualquer estrutura de registro em tempo de execução, ao contrário de um cursor fortemente tipado que é vinculado a um objeto do banco de dados (MCLAUGHLIN, 2008);
- na linha 8 o cursor é aberto para o conteúdo da variável v_query;
- na linha 10 cada registro do cursor c_emp será buscado e inserido na variável v_emp_id;
- na linha 11 quando não existirem mais registro para serem buscados o atributo c_emp%NOTFOUND irá retornar TRUE e finalizar o LOOP;
- na linha 14 o cursor é fechado.

Ambos os cursores realizam o mesmo trabalho, no entanto é importante saber quando utilizar cada cursor.

3.5.4. QUAL CURSOR UTILIZAR

Nos subcapítulos anteriores foram explicados diversos exemplos de cursores, no entanto para se ter um melhor desempenho é necessário saber qual cursor deve ser utilizado para cada situação.

De acordo com FEURSTEIN (2013), abaixo estão descritas algumas das situações que levam o uso destes cursores:

- quando for necessário buscar apenas uma linha, a instrução SELECT-INTO ou EXECUTE IMMEDIATE INTO (quando a consulta for dinâmica) deve ser utilizada. Não utilizar um cursor explícito ou um cursor FOR LOOP;
- quando for necessário buscar *todos* os registros de uma tabela, a melhor opção é um cursor FOR LOOP, exceto se o corpo do LOOP executar alguma instrução DML. Nestes casos, devem ser utilizadas as instruções BULK COLLECT e FORALL (abordadas nos capítulos 13 e 14);
- quando for necessário buscar registros através da instrução BULK COLLECT, mas limitando o número de registros retornados a cada busca, então a melhor escolha é um cursor explícito;
- quando for necessário buscar várias linhas de uma tabela, mas durante a execução do LOOP alguma condição provocar a saída do cursor antes de todos os registros serem buscados, deve ser utilizado um cursor explícito;
- quando for necessário construir uma consulta SQL em tempo de execução, então a instrução EXECUTE IMMEDIATE deve ser utilizada.

3.6. HINT NOCOPY

Os parâmetros IN OUT passados para funções e procedimentos no PL/SQL são passados por valor, ou seja, os valores do parâmetro IN OUT são copiados antes da execução da subrotina ser executada. Durante a execução da subrotina, variáveis temporárias guardam os valores dos parâmetros OUT. Se a subrotina terminar normalmente, este valores são copiados para os parâmetros atuais. Se gerar um erro na subrotina, os valores dos parâmetros originais não serão alterados (Oracle Docs, 2013).

Em situações onde os parâmetros carregam grandes estrutura de dados como *collections*, registros e instâncias de objetos, a cópia dos parâmetros pode impactar na execução e memória consumida pela aplicação. Neste caso, a sobrecarga ocorre a cada nova chamada da subrotina (Oracle Docs, 2013).

Neste caso específico para evitar a sobrecarga pode ser o *hint* NOCOPY, que irá dizer ao compilador para passar os parâmetros OUT e IN OUT por referência.

Quando utilizar o *hint* NOCOPY é preciso ficar atento. Se o programa terminar normalmente, o comportamento será o mesmo que nos parâmetros passados por valor. No entanto se a execução do programa terminar com uma exceção, as alterações já realizadas até então nos parâmetros serão mantidas. Na Figura 23 é mostrado como utilizar o *hint* NOCOPY.

Figura 23 - Exemplo NOCOPY

```
1   DECLARE
2      TYPE tab_empregados IS TABLE OF empregados%ROWTYPE;
3      v_emp tab_empregados;
4      PROCEDURE ATUALIZA_CADASTRO
5      ( pio_emp INT OUT empregados%ROWTYPE)
6      IS
7      BEGIN
8      ..
9      END ATUALIZA_CADASTRO;
10
11     PROCEDURE ATUALIZA_CADASTRO2
12     ( pio_emp INT OUT NOCOPY empregados%ROWTYPE)
13     IS
14     BEGIN
15     ..
16     END ATUALIZA_CADASTRO2;
17  BEGIN
18     SELECT *
19       BULK COLLECT INTO v_temp
20       FROM empregados
21      WHERE departamento_id = 10;
22
23      ATUALIZA_CADASTRO(v_emp);
24
25      ATUALIZA_CADASTRO2(v_emp);
26  END;
```

Conforme pode ser visto na Figura 23, nas linhas 2-3 foi declarada uma PL/SQL *table* e uma variável deste tipo. Nas linhas abaixo é mostrado o cabeçalho de dois procedimentos ATUALIZA_CADASTRO. Neste exemplo todos os empregados do departamento 10 são carregados na variável v_emp e, em seguida, são passados para o procedimento ATUALIZA_CADASTRO atualizar seus dados. No primeiro procedimento, é mostrado um exemplo onde a variável v_emp é passada por valor. Neste caso os valores de todos os empregados são duplicados dentro na execução do procedimento ATUALIZA_CADASTRO. Em seguida é mostrado o procedimento fazendo a passagem por referência. Neste caso, não é realizada uma cópia dos dados no procedimento ATUALIZA_CADSTRO2. No entanto, se ocorrer algum erro e o procedimento terminar a execução antes do esperado, as modificações na variável não podem ser desfeitas. A utilização do *hint* NOCOPY é uma boa maneira de otimizar os custos da aplicação, no entanto quando

este modelo de implementação for utilizado é necessário ter um bom controle de erros.

Conforme a documentação do Oracle, NOCOPY é um *hint* e não uma instrução, ou seja, em determinadas situações o compilador pode ignorar o *hint* NOCOPY e passar os parâmetros por valor. Esta situação irá ocorrer nas exceções descritas abaixo:

- o parâmetro é um elemento de um *array* associativo. Esta restrição não se aplica se o parâmetro for um *array* associativo (apenas elementos);
- o parâmetro possui alguma restrição (*constraint*) , exemplo NOT NULL;
- os parâmetros reais e informados são registros onde um ou mais registros foram declarados utilizando %ROWTYPE ou %TYPE, e as restrições nos campos correspondentes diferem;
- os parâmetros reais e informados são registros onde o parâmetro real foi declarado implicitamente (como um índice de um cursor FOR LOOP) e as restrições dos campos correspondentes diferem;
- na passagem do parâmetro exigir conversão implícita;
- a subrotina for chamada através de um link do banco de dados ou como um procedimento externo.

4. ESTUDO DE CASO

Através dos recursos estudados nos capítulos anteriores, foram realizados testes verificando o desempenho obtido na aplicação. Foram montados testes para executar a mesma funcionalidade através dos recursos estudados no capítulo 3, com o objetivo de comparar o desempenho entre cada algoritmo para permitir uma análise de qual é o algoritmo mais adequado para cada situação.

Para a realização destes testes foi necessário a instalação de um sistema executando a versão do banco de dados ORACLE 11g release 2.

Para testar o desempenho foi configurada uma base com um grande volume de dados para que fosse possível identificar a otimização proposta. Para um grande volume de dados neste trabalho foram consideradas consultas de um milhão a cinco milhões de registros.

As medições foram baseadas no tempo que as instruções levaram para completar a execução. Para evitar desigualdade na execução das instruções testadas, cada teste foi executado aproximadamente dez a quinze vezes, e para o resultado do desempenho, foi considerada a média dos resultados.

O computador utilizado para os testes teve o sistema operacional instalado exclusivamente para este objetivo com a quantidade mínima de recursos executando paralelamente com o SGBD, para evitar concorrência na execução dos testes.

Os testes foram realizados executando de dez a quinze vezes cada algoritmo e fazendo uma média das execuções.

Para gravar o tempo foi marcado a hora inicial antes da execução e gravado a hora final após a execução. Os tempos foram gravados em segundos e milissegundos.

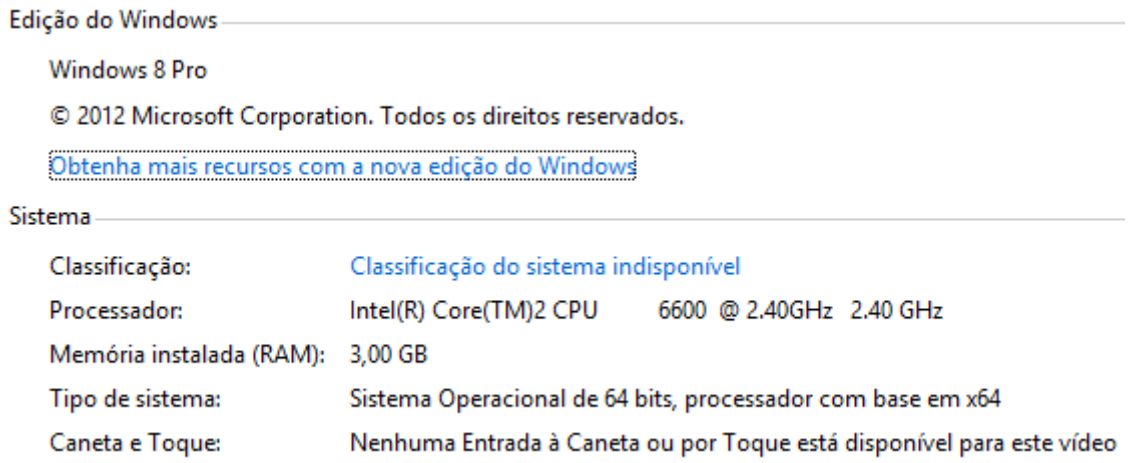
Antes de realizar cada teste é iniciada uma nova sessão. Os scripts executam os testes sequencialmente e gravam os resultados na tabela tempo_execucao.

Para os testes onde foi necessário inserir dados, foi utilizado a tabela pesquisa.

4.1. RECURSOS

Para o sistema, foi utilizado um computador com processador Intel(r) Core 2 duo 2,4GHz, 3 GB de memória RAM e disco rígido de 160 GB com um sistema operacional Windows 8 Professional 64 bits, conforme Figura 24.

Figura 24 - Informações do Sistema



A versão do Oracle instalada foi o Oracle 11g Enterprise Edition Release 11.2.0.1.0. Também foi necessário instalar a versão do Oracle 11g Release 2 para executar no cliente.

Para a realização dos testes a ferramenta utilizada foi Toad For Oracle 9.5.0.31. Por questões de compatibilidade com a ferramenta Toad, foi necessário instalar a versão 32 bits do cliente, pois quando era utilizada a versão 64 bits o *software* não reconhecia o cliente instalado.

4.2. PREPARAÇÃO DO AMBIENTE

Para realizar os testes o primeiro passo foi criar um usuário chamado TCC através do script mostrado na Figura 25.

Figura 25 - Script criação do usuário

```
CREATE USER TCC
IDENTIFIED BY tcc;

GRANT CREATE SEQUENCE TO TCC;

GRANT CREATE TABLE TO TCC;

GRANT CREATE SYNONYM TO TCC;

GRANT CREATE VIEW TO TCC;

GRANT ALTER SESSION TO TCC;

ALTER USER TCC ACCOUNT UNLOCK;

GRANT UNLIMITED TABLESPACE TO TCC;

GRANT CREATE PROCEDURE TO tcc;
```

Para registrar os tempos dos testes foi criada a tabela tempo_execucao, conforme Figura 26, e o procedimento SALVA_TEMPO para gravar os tempos, conforme Figura 27. O procedimento para gravar o tempo dos testes é utilizando antes do trecho testado ser executado e após a execução do teste, desta forma não há influência no resultado.

Figura 26 - Script tabela TEMPO_EXECUCAO

```
DROP TABLE TEMPO_EXECUCAO;

CREATE TABLE TEMPO_EXECUCAO
( ID              NUMBER(10)
, USUARIO         VARCHAR2(100)
, TEMPO_EXECUCAO INTERVAL DAY (1) TO SECOND (3)
);

CREATE SEQUENCE SEQ_ID_TEMPO_EXECUCAO MINVALUE 1 MAXVALUE 9999999999 CYCLE;
```

Figura 27 - Procedimento SALVA_TEMPO

```
CREATE OR REPLACE PROCEDURE SALVA_TEMPO ( pi_usuario     IN VARCHAR2
                                        , pi_sequencia   IN NUMBER
                                        , pi_tempo       IN INTERVAL DAY TO SECOND
                                        ) IS
PRAGMA AUTONOMOUS_TRANSACTION;

BEGIN

   INSERT INTO TEMPO_EXECUCAO
      (ID
      , USUARIO
      , SEQUENCIA
      , TEMPO_EXECUCAO
      )
   VALUES
      ( SEQ_ID_TEMPO_EXECUCAO.NEXTVAL
      , pi_usuario
      , pi_sequencia
      , pi_tempo
      );

   COMMIT;

END;
```

Para simular as inserções dos dados buscados foi criada a tabela pesquisa, conforme pode ser visto na Figura 28.

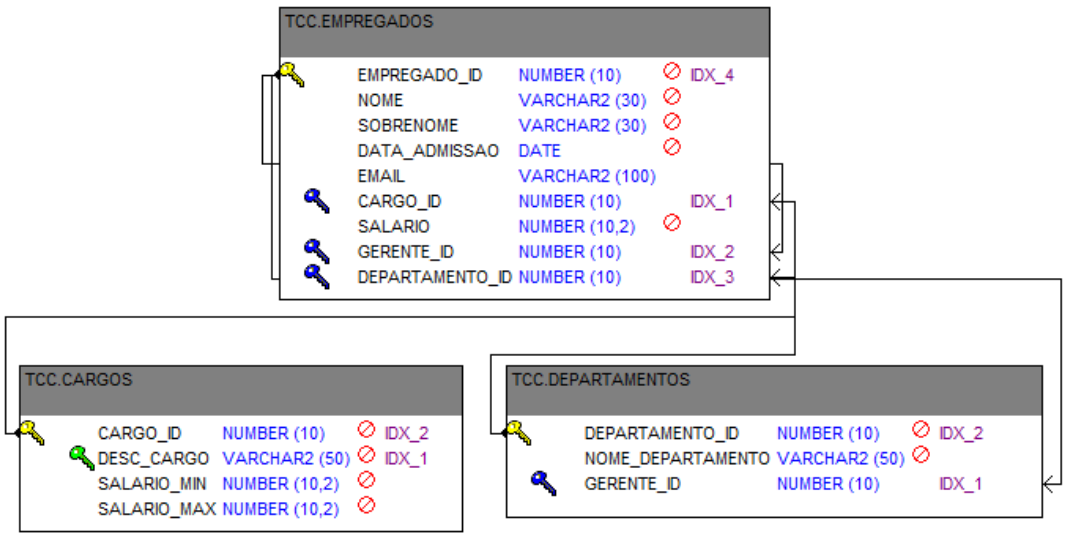
Figura 28 - Script tabela pesquisa

```
CREATE TABLE PESQUISA
(
  USUARIO    VARCHAR2(40 BYTE),
  SEQUENCIA  NUMBER(10),
  TEXTO01    VARCHAR2(4000 BYTE),
  ID         NUMBER(10)
);

CREATE SEQUENCE SEQ_ID_PESQUISA MINVALUE 1 MAXVALUE 9999999999 CYCLE
```

Em seguida foi necessário criar um ambiente de trabalho. No modelo foi criada uma tabela de empregados, cada empregado com seu cargo, departamento e gerente. Uma tabela de departamentos com o gerente responsável e uma tabela de cargos com a faixa de salário de cada cargo. A Figura 29 mostra o modelo lógico do sistema com estas tabelas.

Figura 29 - Modelo lógico do sistema



A base de dados foi montada baseando-se em parte nos dados da base HR de exemplo da Oracle.

Conforme mostra a Figura 30 e 31, os *scripts* das tabelas de cargos e departamentos foram baseados na tabelas jobs e departments do usuário HR das tabelas de exemplo da Oracle.

Figura 30 - Script inserção cargos

```
1  INSERT INTO CARGOS ( CARGO_ID
2                     , DESC_CARGO
3                     , SALARIO_MIN
4                     , SALARIO_MAX
5                     )
6  SELECT SEQ_ID_CARGOS.NEXTVAL
7       , JOB_TITLE
8       , MIN_SALARY
9       , MAX_SALARY
10    FROM hr.jobs;
```

Figura 31 - Scripts inserção departamentos

```
1  INSERT INTO departamentos ( DEPARTAMENTO_ID
2                            , NOME_DEPARTAMENTO
3                            )
4  SELECT SEQ_ID_DEPARTAMENTOS.NEXTVAL
5       , department_name
6    FROM hr.departments dep
7
```

Em seguida foi criado o *script* para preencher a tabela de empregados. Para este script foi realizado o cruzamentos de mais de oitocentos nomes por mais de mil sobrenomes diferentes para gerar aproximadamente oitocentos mil empregados (Anexo 1). Após a tabela de empregados ter sido preenchida, outro script foi executado para alterar manualmente o campo gerente_id, salario e departamento_id da tabela empregados para os cargos de gerente e preencher o campo gerente_id da tabela de departamentos (Anexo 2).

Para finalizar a configuração da base foi gerado um *script* para atribuir aos demais empregados cargos, departamentos e salários, conforme a faixa de cada cargo de maneira aleatória (Anexo 3).

4.3. CONSULTAS

4.3.1. TESTE

Para analisar o ganho de desempenho obtido ao se utilizar BULK COLLECT, foi elaborado um teste com o objetivo de retornar um grande volume de registros (de um milhão a cinco milhões). Neste teste foram executadas as mesmas consultas através do cursor explícito, cursor implícito e BULK COLLECT. A consulta utilizada busca informações dos empregados realizando JOINS com as tabelas de departamentos e cargos, porém para gerar mais registros o CROSS JOIN foi utilizado.

Neste teste foram executadas 15 consultas, 5 para cursor explícito, 5 para cursor implícito e 5 para o BULK COLLECT. Cada uma das consultas foi executada 15 vezes para fazer uma média de 10 execuções, excluindo-se os resultados que estavam mais fora da média. A execução de cada um dos scripts ocorreu em uma nova sessão e gravou os resultados na tabela tempo_execucao.

Nas figuras 32, 33, 34, estão sendo mostradas as principais diferenças entre os recursos utilizados nos testes.

Figura 32 - Teste com cursor implícito

```
36    FOR r_emp IN c_empregados LOOP
37      v_empregados(v_indice).nome := r_emp.nome;
38      v_empregados(v_indice).sobrenome := r_emp.sobrenome;
39      v_empregados(v_indice).departamento := r_emp.departamento;
40      v_empregados(v_indice).cargo := r_emp.cargo;
41      v_empregados(v_indice).salario := r_emp.salario;
42      v_indice := v_indice + 1;
43    END LOOP r_emp;
```

Figura 33 - Teste com BULK COLLECT

```
269   SELECT emp.nome
270        , emp.sobrenome
271        , dept.nome_departamento departamento
272        , cg.desc_cargo cargo
273        , emp.salario
274     BULK COLLECT INTO v_empregados
275     FROM empregados emp
276     JOIN departamentos dept
277    USING (departamento_id)
278     JOIN cargos cg
279    USING (cargo_id)
280 CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 2) emp2
281  ORDER BY emp.nome
```

Figura 34 - Teste com cursor explícito

```
485   OPEN c_empregados;
486   LOOP
487     FETCH c_empregados INTO v_empregado;
488     EXIT WHEN c_empregados%NOTFOUND;
489     v_empregados(v_indice):= v_empregado;
490     v_indice := v_indice + 1;
491   END LOOP r_emp;
492   CLOSE c_empregados;
```

As tabelas 1,2 e 3 mostram o resultado dos testes.

O código utilizado para realizar estes testes está disponível no Anexo 4.

Tabela 1 - Resultado da consulta com cursor

Consulta com cursor implícito		
Consulta	Linhas consultadas	Tempo médio exec. (segundos)
Consulta 1	1.601.856	29,4375
Consulta 2	2.402.784	45,2077
Consulta 3	3.203.712	65,0464
Consulta 4	4.004.640	77,2675
Consulta 5	4.805.569	102,2971

Tabela 2 - Resultado da consulta com BULK COLLECT

Consulta com BULK COLLECT		
Consulta	Linhas consultadas	Tempo médio exec. (segundos)
Consulta 1	1.601.856	26,1988
Consulta 2	2.402.784	41,8226
Consulta 3	3.203.712	58,9122
Consulta 4	4.004.640	73,3927
Consulta 5	4.805.569	91,4887

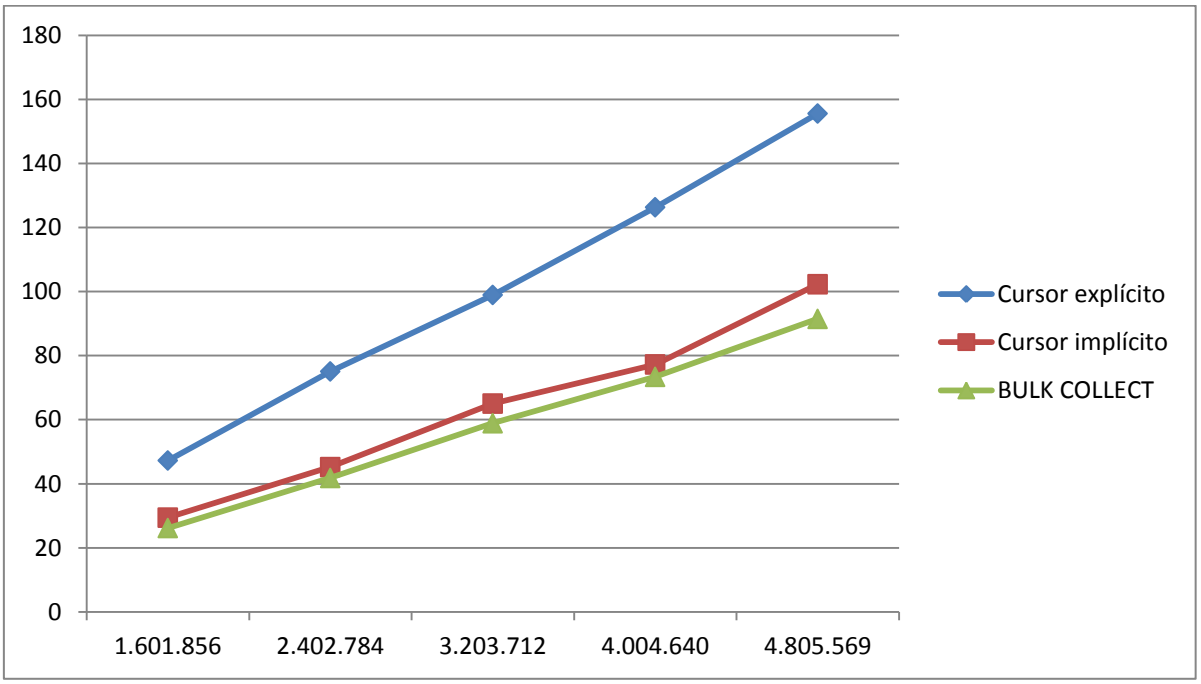
Tabela 3 - Resultado da consulta com cursor explícito

Consulta com cursor explícito		
Consulta	Linhas consultadas	Tempo médio exec. (segundos)
Consulta 1	1.601.856	47,2795
Consulta 2	2.402.784	75,0782
Consulta 3	3.203.712	98,9577
Consulta 4	4.004.640	126,324
Consulta 5	4.805.569	155,5866

4.3.2. ANÁLISE

Conforme pode ser observado no gráfico da Figura 35, mesmo com um grande volume de dados o BULK COLLECT tem um pequeno ganho de desempenho em relação ao cursor implícito. Este resultado ocorre porque o cursor implícito também trabalha de forma semelhante ao BULK COLLECT. No entanto, é perceptível a considerável perda de desempenho ao executar as mesmas consultas com o cursor explicito. Através do gráfico pode ser analisado que quanto maior a quantidade de registros consultados, maior é esta perda de desempenho.

Figura 35 - Gráfico das consultas



No gráfico da Figura 35, o eixo do X representa a quantidade de registros retornados em cada consulta, e o eixo do Y representa o tempo em segundos que cada consulta levou para executar.

Através deste resultado é possível identificar qual é o melhor cenário para utilizar cada um destes recursos:

- Sempre que for necessário consultar todos os dados, a maneira mais rápida é a busca através do BULK COLLECT. No entanto, se não for necessário armazenar estes dados em uma *collection,* o cursor

implícito também pode ser utilizado, com uma pequena perda de performance.

- O cursor explícito só deveria ser utilizado quando o cursor puder ser finalizado antes de consultar todos os registros. Pois, como o cursor explícito não carrega todos os registros de uma vez, ele pode ter melhor resultado, conforme a quantidade de registros consultados.

4.4. CONSULTA ESTÁTICA X CONSULTA DINÂMICA

4.4.1. TESTE

Para realizar uma comparação entre o custo de uma consulta estática e uma consulta dinâmica foi realizado um teste um pouco diferente do teste da seção 4.3. Neste teste foram consultados todos os departamentos e para cada departamento foram consultados todos empregados que fazem parte dele. O objetivo desta consulta é mudar o SQL consultado a cada execução para avaliar o tempo gasto parar realizar o *parse* no SQL dinâmico e estático.

Na base de dados existem 27 departamentos, então cada consulta foi executada 27 vezes, deste 27 departamentos apenas 14 possuem empregados vinculados. Para realizar a média deste teste este conjunto de informações foi executado 15 vezes.

Nas figuras 36, 37, estão sendo mostradas as principais diferenças entre os recursos utilizados nos testes.

Figura 36 - Teste com consulta estática por departamento.

```
28      -- Percorre todos os departamentos
29      FOR r_dept IN c_departamentos LOOP
30        v_data_ini_parc := SYSTIMESTAMP;
31        SELECT emp.nome
32             , emp.sobrenome
33             , dept.nome_departamento departamento
34             , cg.desc_cargo cargo
35             , emp.salario
36  BULK COLLECT INTO v_empregados
37          FROM empregados emp
38          JOIN departamentos dept
39         USING (departamento_id)
40          JOIN cargos cg
41         USING (cargo_id)
42         WHERE departamento_id = r_dept.departamento_id
43       ORDER BY emp.nome
44             , emp.sobrenome;
```

Figura 37 - Teste com consulta dinâmica por departamento

```
84      -- Percorre todos os departamentos
85      FOR r_dept IN c_departamentos LOOP
86        -- Monta string com sql
87        v_sql := 'SELECT emp.nome
88                       , emp.sobrenome
89                       , dept.nome_departamento departamento
90                       , cg.desc_cargo cargo
91                       , emp.salario
92                    FROM empregados emp
93                    JOIN departamentos dept
94                   USING (departamento_id)
95                    JOIN cargos cg
96                   USING (cargo_id)
97                   WHERE departamento_id = '||r_dept.departamento_id||'
98                ORDER BY emp.nome
99                       , emp.sobrenome';
100       v_data_ini_parc := SYSTIMESTAMP;
101       EXECUTE IMMEDIATE v_sql
102       BULK COLLECT INTO v_empregados;
```

A tabela 4 e 5 mostram o resultado deste teste.

O código utilizado para realizar estes testes está disponível no Anexo 5.

Tabela 4 - Resultado da consulta com SQL estático

Consulta com SQL estático	
Consulta	Tempo médio exec. (segundos)
Consulta 1	43,4183

Tabela 5 - Resultado da consulta com SQL dinâmico

Consulta com SQL dinâmico	
Consulta	Tempo médio exec. (segundos)
Consulta 1	23,0851

Em seguida foi realizada uma consulta dinâmica utilizando o mesmo teste utilizado na seção 4.3.1 para comparar com a consulta da Tabela 2 .

Nas figuras 33, 38, estão sendo mostradas as principais diferenças entre os recursos utilizados nos testes.

Figura 38 - Consulta dinâmica com BULK COLLECT

```
184     v_sql := 'SELECT emp.nome
185         , emp.sobrenome
186         , dept.nome_departamento departamento
187         , cg.desc_cargo cargo
188         , emp.salario
189       FROM empregados emp
190       JOIN departamentos dept
191      USING (departamento_id)
192       JOIN cargos cg
193      USING (cargo_id)
194  CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 6) emp2
195   ORDER BY emp.nome
196         , emp.sobrenome';
197     EXECUTE IMMEDIATE v_sql
198     BULK COLLECT INTO v_empregados;
```

A Tabela 6 mostra o resultado da consulta dinâmica.

O código utilizado para realizar estes testes está disponível no Anexo 6.

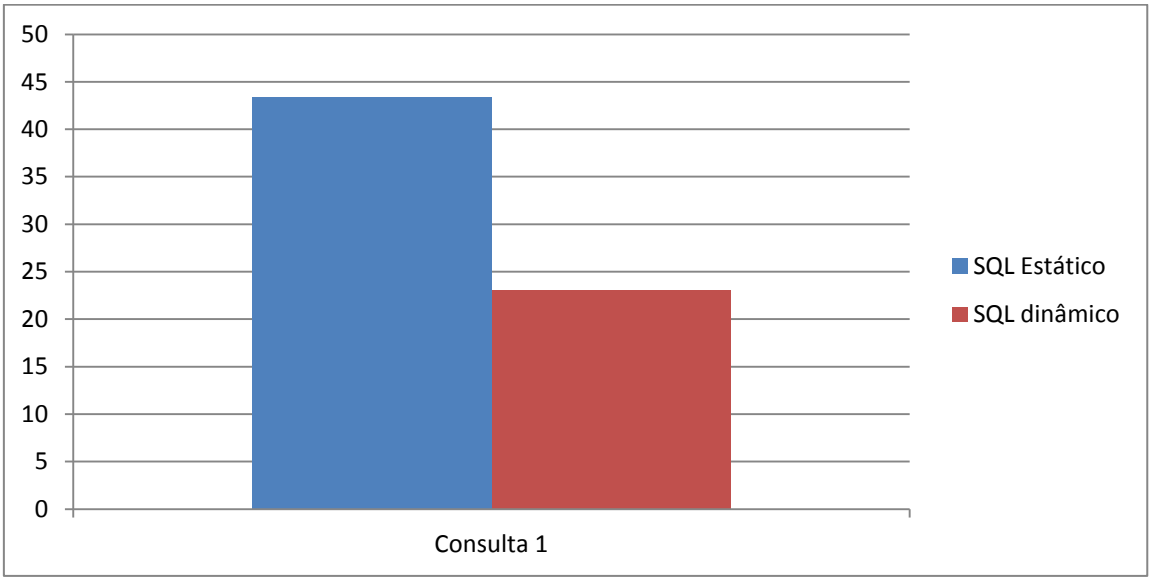
Tabela 6 - Resultado da consulta com SQL dinâmico

Consulta com SQL dinâmico		
Consulta	Linhas consultadas	Tempo médio exec. (segundos)
Consulta 1	1.601.856	25,6249
Consulta 2	2.402.784	40,192
Consulta 3	3.203.712	54,1358
Consulta 4	4.004.640	70,5844
Consulta 5	4.805.569	90,9774

4.4.2. ANÁLISE

Na maior parte das vezes o SQL estático tem melhor performance que SQL dinâmico (Oracle, 2013). No entanto, neste teste o resultado foi o oposto. Conforme pode ser visto na Figura 39, a consulta com SQL dinâmico teve um resultado muito melhor. Para entender porque a diferença foi tão grande, neste exemplo foi analisado o tempo gasto em cada consulta, e pôde ser visto que utilizando SQL dinâmico, nos casos em que foram realizadas consultas em departamentos que não tinham nenhum empregado e não retornaram informações, o custo da consulta dinâmica foi zero, porém no SQL estático, estas mesmas consultas mantiveram o mesmo custo que as consultas que retornaram informações.

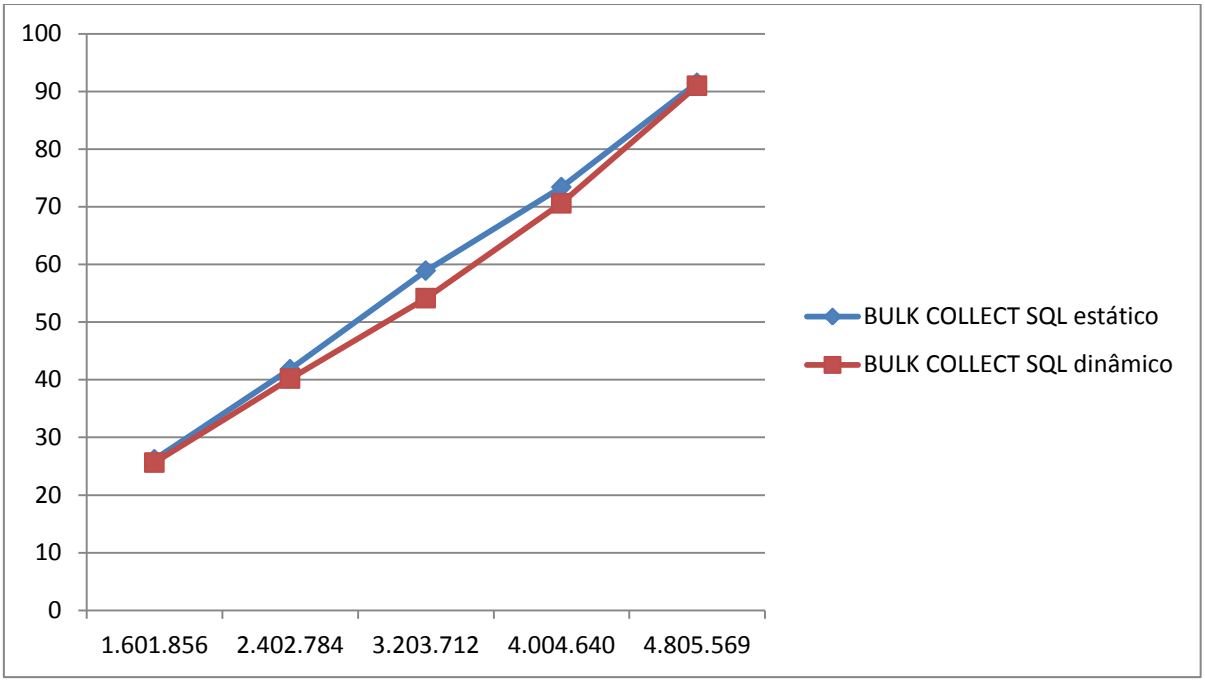
Figura 39 - Resultado consulta dinâmica x estática



No gráfico da Figura 39, o eixo do Y representa, em segundos, o tempo que cada consulta levou para executar.

No segundo teste, conforme pode ser visto na Figura 40, o SQL dinâmico teve um desempenho um pouco melhor em relação ao SQL estático. Este desempenho foi ficando mais próximo na medida que o volume de dados aumentava.

Figura 40 - Gráfico consulta estática x dinâmica



No gráfico da Figura 40, o eixo do X representa a quantidade de registros consultados, e o eixo do Y representa o tempo, em segundos, que cada consulta levou para executar.

Analisando estas informações é possível perceber que nem sempre o SQL estático possui um melhor desempenho em relação ao SQL dinâmico. Normalmente, existe este ganho de desempenho no SQL estático devido à necessidade de realizar o *parse* a cada execução do SQL dinâmico. Porém, no primeiro teste pôde ser visto que a forma como o banco executa estas consultas também é diferente, e neste caso, a utilização do SQL dinâmico resultou melhor desempenho quando a consulta não retornou nenhum registro.

4.5. *BINDING VARIABLES* (VARIAVEIS DE LIGAÇÃO)

4.5.1. TESTE

Para testar o desempenho do uso das variáveis de ligação foi utilizado o exemplo onde são consultados 27 departamentos e consultados todos os empregados de cada departamento. Neste teste foram utilizados as consultas através de BULK COLLECT e cursor explícito.

Para realizar este teste, para cada um dos tipos de consulta foi iniciado uma nova sessão, executado 15 vezes e realizado a média dos resultados.

Nas figuras 37, 41, 42 43, estão sendo mostradas as principais diferenças entre os recursos utilizados nos testes.

Figura 41 - Teste consulta dinâmica com BULK COLLECT e BIND

```
91    FOR r_dept IN c_departamentos LOOP
92       -- Monta string com sql
93       v_sql := 'SELECT emp.nome
94                      , emp.sobrenome
95                      , dept.nome_departamento departamento
96                      , cg.desc_cargo cargo
97                      , emp.salario
98                   FROM empregados emp
99                   JOIN departamentos dept
100                 USING (departamento_id)
101                  JOIN cargos cg
102                 USING (cargo_id)
103                 WHERE departamento_id = :1
104              ORDER BY emp.nome
105                      , emp.sobrenome';
106      v_data_ini_parc := SYSTIMESTAMP;
107      EXECUTE IMMEDIATE v_sql
108      BULK COLLECT INTO v_empregados
109                   USING r_dept.departamento_id;
```

Figura 42 - Teste cursor dinâmico explícito

```
156  FOR r_dept IN c_departamentos LOOP
157      -- Monta string com sql
158      v_sql := 'SELECT emp.nome
159                     , emp.sobrenome
160                     , dept.nome_departamento departamento
161                     , cg.desc_cargo cargo
162                     , emp.salario
163                  FROM empregados emp
164                  JOIN departamentos dept
165                 USING (departamento_id)
166                  JOIN cargos cg
167                 USING (cargo_id)
168                 WHERE departamento_id = '||r_dept.departamento_id||'
169               ORDER BY emp.nome
170                     , emp.sobrenome';
171      v_indice := 1;
172      v_data_ini_parc := SYSTIMESTAMP;
173      OPEN c_emp FOR v_sql;
174      LOOP
175         FETCH c_emp INTO v_empregados(v_indice);
176         EXIT WHEN c_emp%NOTFOUND;
177         v_soma := v_soma + v_empregados(v_indice).salario;
178         v_indice := v_indice + 1;
179      END LOOP;
```

Figura 43 - Teste cursor explícito com BIND

```
220  FOR r_dept IN c_departamentos LOOP
221      -- Monta string com sql
222      v_sql := 'SELECT emp.nome
223                     , emp.sobrenome
224                     , dept.nome_departamento departamento
225                     , cg.desc_cargo cargo
226                     , emp.salario
227                  FROM empregados emp
228                  JOIN departamentos dept
229                 USING (departamento_id)
230                  JOIN cargos cg
231                 USING (cargo_id)
232                 WHERE departamento_id = :1
233               ORDER BY emp.nome
234                     , emp.sobrenome';
235      v_indice := 1;
236      v_data_ini_parc := SYSTIMESTAMP;
237      OPEN c_emp FOR v_sql
238      USING r_dept.departamento_id;
239      LOOP
240         FETCH c_emp INTO v_empregados(v_indice);
241         EXIT WHEN c_emp%NOTFOUND;
242         v_soma := v_soma + v_empregados(v_indice).salario;
243         v_indice := v_indice + 1;
244      END LOOP;
```

A Tabela 7 mostra a média de cada uma destas consultas.

O código utilizado para realizar estes testes estádisponível no Anexo 7.

Tabela 7 - Resultado dos testes utilizando variáveis de ligação.

Consulta	Tempo médio exec. (segundos)
BULK COLLECT	23,2723
BULK COLLECT com BIND	43,411
Cursor explícito	32,1134
Cursor explícito com BIND	14,2772

Para entender melhor este resultado, a média da consulta realizada para cada departamento foi mostrada na Tabela 8.

Tabela 8 - Resultado das consultas por departamento

Dept.	Tempo médio exec. (segundos)			
	BULK COLLECT	BULK COLLECT com BIND	Cursor Explícito	Cursor Explícito com BIND
2	1,7285	1,7129	2,7217	1,5111
3	1,6516	1,6517	2,421	1,1639
4	1,6974	1,7077	2,7591	1,6369
5	1,6235	1,6484	2,3451	1,1315
6	1,9003	1,9456	4,1746	3,5443
7	1,6172	1,5911	2,2337	1,0151
8	1,6598	1,637	2,1693	0,8776
9	1,5746	1,5881	2,1767	0,8757
10	1,5029	1,5038	1,4956	0
11	1,6607	1,6673	2,3451	1,1461
12	1,6659	1,6545	2,4015	1,1626
13	1,5115	1,5203	1,5059	0
14	1,5265	1,5361	1,5673	0
15	1,5415	1,5381	1,5778	0
16	0,0011	1,5288	0,002	0
17	0	1,5517	0,001	0
18	0	1,5724	0,0011	0
19	0,0011	1,5444	0	0
20	0,001	1,5185	0,0011	0
21	0	1,5475	0	0
22	0	1,5631	0	0
23	0	1,5488	0	0
24	0	1,5369	0	0
25	0	1,5664	0	0
26	0	1,5622	0	0
27	0	1,5205	0	0
28	0	1,5465	0	0

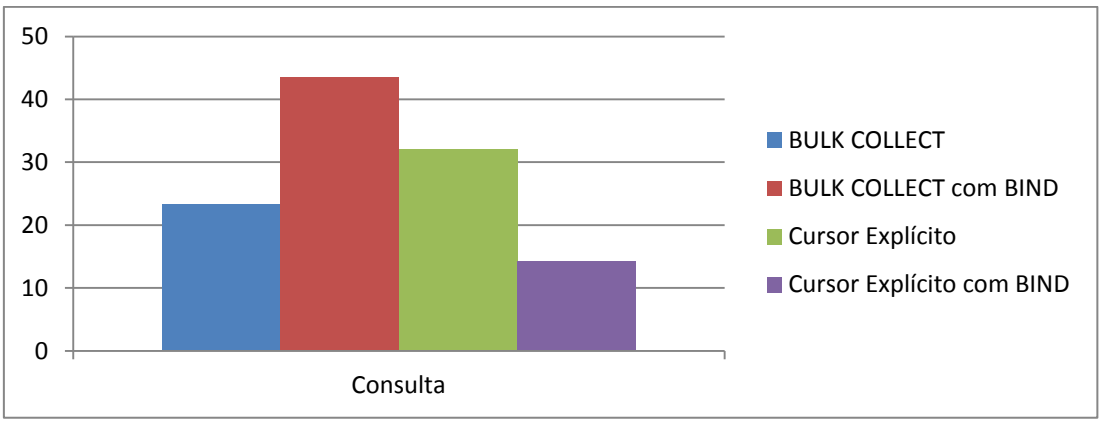
4.5.2. ANÁLISE

O teste utilizando variáveis de ligação mostrou que dependendo da situação pode ocorrer uma melhora, ou uma perda de desempenho na aplicação.

Conforme pode ser observado na Figura 44, a utilização do BULK COLLECT garante um desempenho melhor que o cursor explícito. No entanto, quando foi utilizado variáveis de ligação, o BULK COLLECT apresentou o mesmo comportamento do SQL estático e obteve o pior tempo de execução. Ao examinar a Figura 45, é possível perceber que o motivo da perda de desempenho do BULK COLLECT com variáveis BIND ocorre porque a consulta que antes não estava gerando custo quando não retornava informações, passou a ter um custo semelhante às demais consultas.

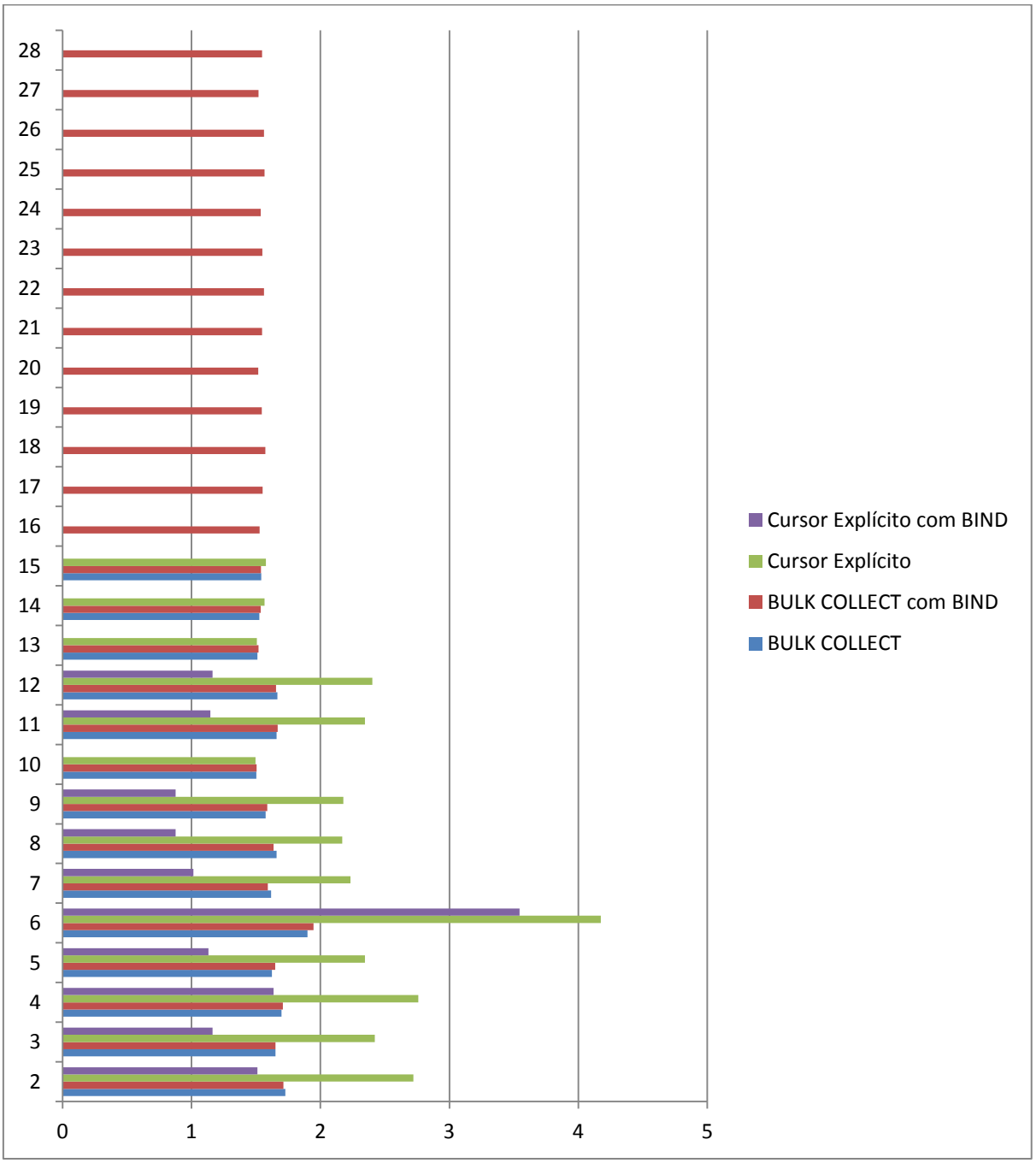
De maneira geral, pôde ser visto que nas situações onde os departamentos retornaram informações, o custo tanto do cursor explícito com BIND, quanto do cursor sem BIND, resultou em mais tempo de execução em relação ao BULK COLLECT. Fica claro que quando for necessário retornar bastante informações é melhor a utilização de BULK COLLECT ao invés do cursor explícito. No entanto, existem situações onde o sistema pode se comportar diferente do esperado. Como por exemplo, quando as consultas não retornaram dados, e neste caso o BULK COLLECT com BIND obteve o pior resultado, onerando o resultado geral do teste. Portanto, a melhor maneira de saber qual é melhor tipo de consulta a ser realizada é conhecer os dados que estão sendo consultados.

Figura 44 - Resultado da consulta com variáveis de ligação



No gráfico da Figura 44, o eixo do Y representa, em segundos, o tempo necessário para a execução da consulta.

Figura 45 - Resultado da consulta por departamento



No gráfico da Figura 45, o eixo do X representa o tempo necessário para executar a consulta em segundos, e o eixo do Y representa o departamento que foi consultado. Cada barra no gráfico representa o tipo de consulta que foi utilizado.

4.6. NDS X DBMS_SQL

4.6.1. TESTE

Para realizar este teste foi utilizada a consulta com cursor explícito e BIND da seção 4.5.1 onde o resultado está na Tabela 7. O mesmo exemplo foi executado utilizando o DBMS_SQL.

Nas figuras 43, 46, estão sendo mostradas as principais diferenças entre os recursos utilizados nos testes.

Figura 46 - Teste cursor explícito com DBMS_SQL

```
v_cursor := DBMS_SQL.OPEN_CURSOR;
DBMS_SQL.PARSE(v_cursor, v_sql, DBMS_SQL.NATIVE);
-- Fornece variáveis bind
DBMS_SQL.BIND_VARIABLE(v_cursor,'departamento_id',r_dept.departamento_id);
-- Descreve definição de variáveis
DBMS_SQL.DEFINE_COLUMN(v_cursor, 1, v_empregado.nome, 30);
DBMS_SQL.DEFINE_COLUMN(v_cursor, 2, v_empregado.sobrenome, 30);
DBMS_SQL.DEFINE_COLUMN(v_cursor, 3, v_empregado.departamento, 50);
DBMS_SQL.DEFINE_COLUMN(v_cursor, 4, v_empregado.cargo, 50);
DBMS_SQL.DEFINE_COLUMN(v_cursor, 5, v_empregado.salario);
-- Executa cursor
v_cont := DBMS_SQL.EXECUTE(v_cursor);
v_indice := 1;
LOOP
   IF DBMS_SQL.FETCH_ROWS(v_cursor) > 0 THEN
      -- Atribui valores
      DBMS_SQL.COLUMN_VALUE(v_cursor, 1, v_empregado.nome);
      DBMS_SQL.COLUMN_VALUE(v_cursor, 2, v_empregado.sobrenome);
      DBMS_SQL.COLUMN_VALUE(v_cursor, 3, v_empregado.departamento);
      DBMS_SQL.COLUMN_VALUE(v_cursor, 4, v_empregado.cargo);
      DBMS_SQL.COLUMN_VALUE(v_cursor, 5, v_empregado.salario);
      v_empregados(v_indice) := v_empregado;
      v_soma := v_soma + v_empregado.salario;
      v_indice := v_indice + 1;
   ELSE
      EXIT;
   END IF;
END LOOP;
DBMS_SQL.CLOSE_CURSOR(v_cursor);
```

Na Tabela 9 estão apresentados os resultados deste teste.

O código utilizado para realizar este teste está disponível no Anexo 8.

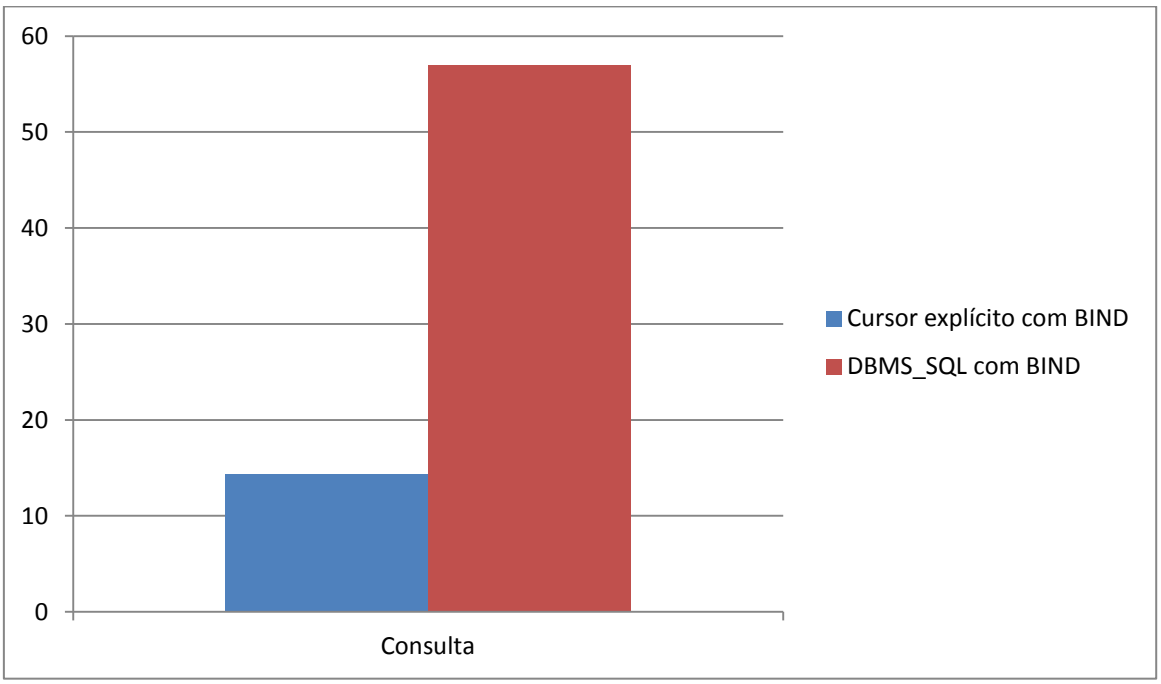
Tabela 9 - Resultado consulta NDS X DBMS_SQL

Consulta	Tempo médio exec. (segundos)
DBMS_SQL com BIND	56,9031
Cursor explícito com BIND	14,2772

4.6.2. ANÁLISE

No teste realizado para comparar o NDS e DBMS_SQL, conforme pode ser visto no gráfico apresentado na Figura 47, é visível a diferença de desempenho. A execução da consulta com NDS é cerca de 4 vezes mais rápida que a consulta com DBMS_SQL.

Figura 47 - Resultado da consulta NDS X DBMS_SQL



No gráfico da Figura 47, o eixo do Y representa, em segundos, o tempo necessário para executar cada consulta.

O resultado deste teste mostra que o DBMS_SQL só deveria ser usado em situações onde não há uma maneira de utilizar NDS, por exemplo, quando se trabalha com uma ferramenta desatualizada e não há suporte ao NDS. Nas aplicações mais antigas desenvolvidas utilizando DBMS_SQL, sempre que possível

é importante ir convertendo o código antigo para o NDS, visto o ganho de desempenho e a simplicidade de implementação.

4.7. DML

4.7.1. TESTE

Para realizar este teste foi realizada a consulta sobre a tabela de empregados, departamentos e cargos. Foram consultados nome, sobrenome, departamento, cargo e salário de todos os empregados. Esta informações foram gravadas na tabela pesquisa utilizando o INSERT de três maneiras. Primeiro foi inserido registro a registro através de um cursor implícito, em seguida em rajada através do FORALL e por último através da inserção direta sobre o SELECT.

Cada teste foi realizado 15 vezes para fazer uma média do tempo de execução. E cada um dos três testes iniciou em uma nova sessão.

Nas figuras 48, 49, 50, estão sendo mostradas as principais diferenças entre os recursos utilizados nos testes.

Figura 48 - Teste INSERT com cursor implícito

```
37  FOR r_emp IN c_empregados LOOP
38    v_info := 'Nome: '||r_emp.nome||
39              ' Sobrenome: '||r_emp.sobrenome||
40              ' Departamento:'||r_emp.departamento||
41              ' Cargo: '|| r_emp.cargo||
42              ' Salário: '||r_emp.salario;
43
44    INSERT INTO PESQUISA
45              ( USUARIO
46              , sequencia
47              , texto01
48              , id
49              )
50          VALUES
51              ( 'DML - INSERT - CURSOR'
52              , 7001
53              , v_info
54              , SEQ_ID_PESQUISA.NEXTVAL
55              );
56  END LOOP r_emp;
```

Figura 49 - Teste INSERT com BULK COLLECT

```
    SELECT 'Nome: '||emp.nome||
           ' Sobrenome: '||emp.sobrenome||
           ' Departamento:'||dept.nome_departamento||
           ' Cargo: '|| cg.desc_cargo||
           ' Salário: '||emp.salario
BULK COLLECT INTO v_informacoes
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo_id)
       ORDER BY emp.nome
           , emp.sobrenome;

   -- Popula vetor de informações
   FORALL r_info IN 1..v_informacoes.COUNT
      INSERT INTO PESQUISA
                  ( USUARIO
                  , sequencia
                  , texto01
                  , id
                  )
           VALUES
                  ( 'DML - INSERT - FORALL'
                  , 7002
                  , v_informacoes(r_info)
                  , SEQ_ID_PESQUISA.NEXTVAL
                  );
```

Figura 50 - Teste INSERT utilizando SELECT

```
    INSERT INTO PESQUISA
                ( USUARIO
                , sequencia
                , texto01
                , id
                )
    SELECT USUARIO
         , SEQUENCIA
         , TEXTO
         , SEQ_ID_PESQUISA.NEXTVAL
      FROM ( SELECT 'DML - INSERT - DIRETO' USUARIO
                  , 7003                    SEQUENCIA
                  , 'Nome: '||emp.nome||
                    ' Sobrenome: '||emp.sobrenome||
                    ' Departamento:'||dept.nome_departamento||
                    ' Cargo: '|| cg.desc_cargo||
                    ' Salário: '||emp.salario  TEXTO
               FROM empregados emp
               JOIN departamentos dept
              USING (departamento_id)
               JOIN cargos cg
              USING (cargo_id)
           ORDER BY emp.nome
                  , emp.sobrenome);
```

A Tabela 10 mostra o resultado destes testes.

O código utilizado para realizar estes testes está disponível no Anexo 9.

Tabela 10 - Resultado das inserções

Consulta	Tempo médio exec. (segundos)
INSERT com CURSOR	50,8553
INSERT com FORALL	22,5018
INSERT através de SELECT	26,2788

Após o primeiro teste foi identificado que parte do ônus na inserção era causado por causa do ORDER BY do SELECT. Então os mesmos testes foram realizados removendo a clausula ORDER BY.

A Tabela 11 mostra o resultado destes novos testes.

O código utilizado para realizar estes testes está disponível no Anexo 10.

Tabela 11 - Resultado das inserções sem ORDER BY

Consulta	Tempo médio exec. (segundos)
INSERT com CURSOR	46,4352
INSERT com FORALL	15,621
INSERT através de SELECT	19,3005

4.7.2. ANÁLISE

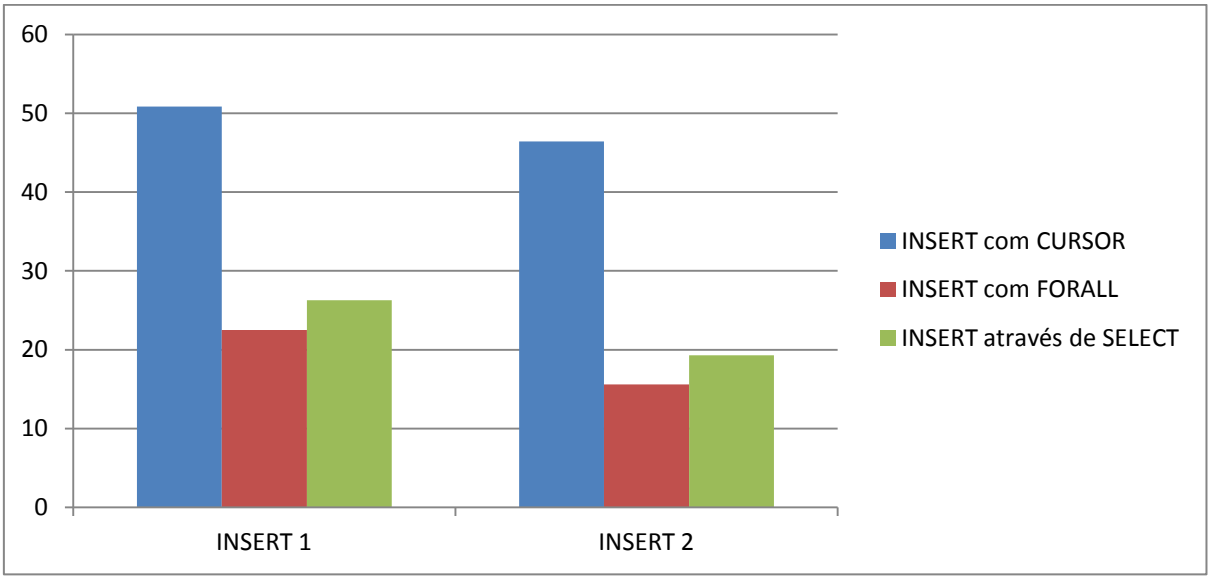
No teste de inserção ficou claro o ganho de performance quando foi utilizado FORALL e inserção através da consulta. O resultado que ocorreu diferente do esperado foi a inserção através de FORALL ser mais rápida que através da inserção utilizando a consulta.

Após o teste da primeira inserção foi visto que parte do custo foi o ORDER BY na consulta. Por este motivo, foi realizado um segundo teste removendo o

ORDER BY das consultas para verificar se este custo teve influência na inserção através da consulta.

Após a remoção do ORDER BY o tempo de consulta diminuiu de forma geral em todos as inserções, não sendo o responsável pelo ganho de performance no FORALL.

Figura 51 - Resultado do teste de inserção



No gráfico da Figura 51, o eixo do Y representa, em segundos, o tempo que cada INSERT levou para executar. O INSERT 1 representa o tempo gasto para realizar os INSERTS utilizando ORDER BY na consulta. O INSERT 2 realiza a mesma função sem utilizar ORDER BY.

Ao analisar o resultado do teste ficou visível o grande ganho de desempenho quando se trabalha com FORALL. O mais surpreendente neste teste foi que o custo do FORALL foi menor até mesmo que na inserção através da consulta.

4.8. NOCOPY

4.8.1. TESTE

Neste teste foi medido o tempo de execução e a memória utilizada por um procedimento e em seguida utilizando o *hint* NOCOPY.

Para realizar este teste foi desenvolvida uma package com os seguintes procedimentos na especificação: EXECUTA_TESTE_SEM_NOCOPY, EXECUTA_TESTE_COM_NOCOPY e EXECUTA_TESTE_COM_PARAMETRO_IN.

No corpo da package foi declarada uma TYPE do tipo TABLE de VARCHAR2(32767), e uma variável deste tipo. Na execução do teste, esta variável era inicializada com um milhão de registros. Em seguida a variável era passada para o procedimento; este procedimento fazia a chamada recursiva 99 vezes passando a variável para o próximo procedimento. Também foi incluído um contador para totalizar a quantidade de vezes que o procedimento foi chamado. A primeira chamada do procedimento foi adicionada há um LOOP que executou 500 vezes. Então, como cada procedimento executou 500 vezes, e cada vez que era chamado fazia a chamada recursiva mais 99 vezes, o procedimento foi executado 500 mil vezes neste teste.

Nas figuras 52, 53, 54, estão sendo mostradas as principais diferenças entre os recursos utilizados nos testes.

Figura 52 - Teste procedimento utilizando IN OUT

```
7   PROCEDURE TESTE_SEM_NOCOPY ( pio_entrada    IN OUT T_TAB
8                              , pi_cont        IN     NUMBER
9                              , pi_repeticoes  IN     NUMBER
10                             , pio_cont_total IN OUT NUMBER
11                             ) IS
```

Figura 53 - Teste procedimento utilizando IN OUT NOCOPY

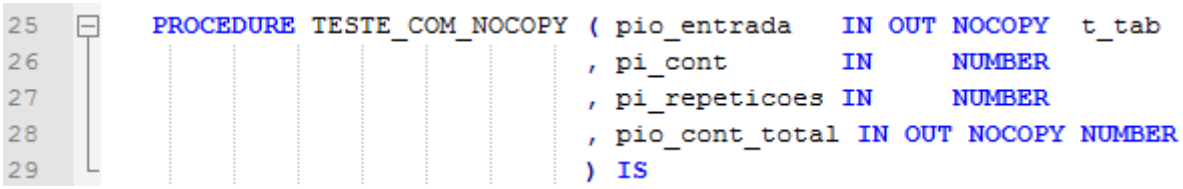
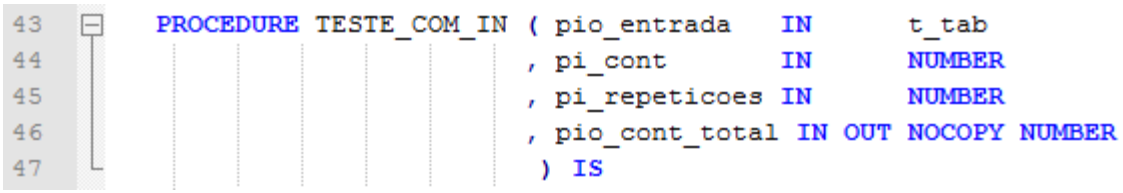


Figura 54 - Teste procedimento utilizando IN



A Tabela 12 mostra a memória utilizada pelos procedimentos e o tempo que foi necessário para a execução. Os tempos mostrados na tabela são uma média de 10 execuções de cada teste. Para cada uma das 10 execuções foi necessário inicializar uma nova sessão.

O código de implementação das packages e testes está disponível no Anexo 10.

Tabela 12 - Resultado da execução dos procedimentos

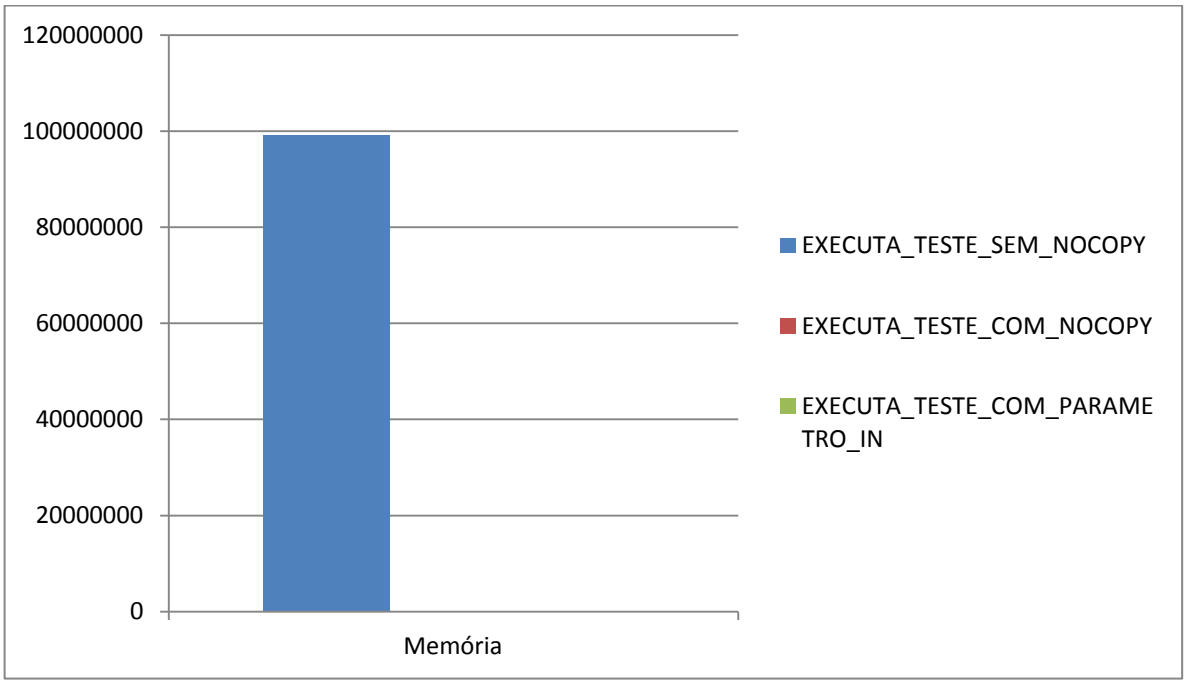
Procedimento	Memória consumida (bytes)	Tempo médio exec. (segundos)
EXECUTA_TESTE_SEM_NOCOPY	99221504	128,7244
EXECUTA_TESTE_COM_NOCOPY	65536	0,0264
EXECUTA_TESTE_COM_PARAMETRO_IN	65536	0,0281

4.8.2. ANÁLISE

Neste teste, ficou visível a diferença de performance do *hint* NOCOPY. Conforme pode ser visto nas figuras 55 e 56, tanto o custo de memória quanto o tempo de execução foi praticamente nulo se comparado com o procedimento que passou o parâmetro por valor. No entanto, foi realizado uma terceira execução não utilizando o *hint* NOCOPY e passando o parâmetro apenas como entrada. O objetivo

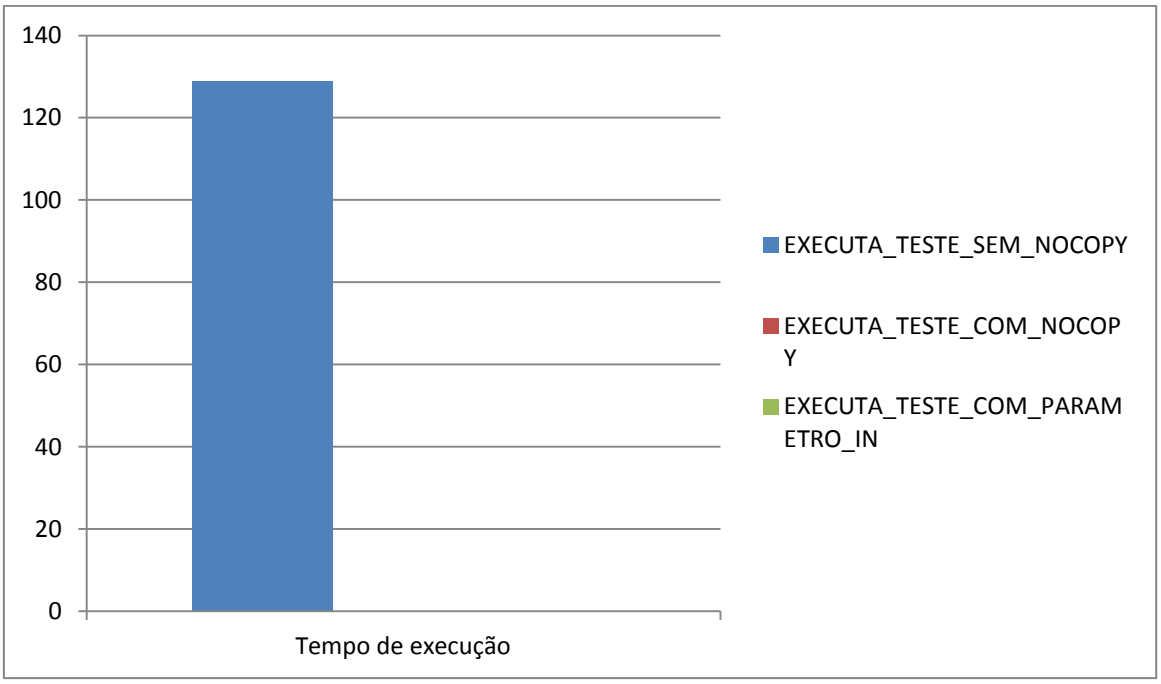
desta terceira execução foi mostrar que os parâmetros de entrada sempre são passados como referência.

Figura 55 - Resultado do custo de memória dos procedimentos



No gráfico da Figura 55, o eixo do Y representa o consumo da execução de cada procedimento em bytes.

Figura 56 - Resultado do custo de execução dos procedimentos



No gráfico da Figura 56, o eixo do Y representa o custo de execução de cada procedimento em segundos.

Analisando os gráficos deste teste, foi possível ver o ganho de desempenho ao utilizar NOCOPY. No entanto, esta alternativa só deveria ser utilizada se o parâmetro é obrigatoriamente OUT ou IN OUT. Como pôde ser visto, parâmetros IN sempre são passados por referência, então se possível esta seria a melhor alternativa para a passagem de grandes *collections* por parâmetro.

5. CONCLUSÃO

Após finalizar as análises dos testes dos recursos estudados foi possível ter uma visão do ganho de desempenho que muitos destes recursos proporcionam e quais são os melhores cenários para utilizá-los.

Dos resultados que que tiveram maior destaque, é possível mencionar a diferença de desempenho ao consultar uma grande quantidade de dados com cursor explícito, implícito e BULK COLLECT. Ao testar o DML através da inserção de dados, o FORALL resultou em um desempenho bem superior em relação à inserção através de cursores. Outro ponto importante que deve ser lembrado, é a importância de substituir as ocorrências do DMBS_SQL pelo NDS. O último recurso teve um bom resultado foi o *hint* NOCOPY, no entanto é preciso lembrar que deve ser utilizado somente quando for realmante necessário.

Uma visão muito importante que este trabalho deixou, foi que não existe uma receita exata para se obter o melhor desempenho utilizando um recurso. Para conseguir o melhor desempenho do banco é importante conhecer os dados que estão sendo consultados, os parâmetros utilizados no banco, entre outras informações. Nos testes realizados, muitos recursos que deveriam ter um melhor desempenho em relação a outros acabaram não dando o resultado esperado. Isto mostrou que muitas vezes para conseguir o melhor desempenho será necessário testar o recurso que se pretende usar, pois o mesmo comando que pode ter um melhor desempenho em uma situação, pode causar uma perda de desempenho em outra.

Para um trabalho futuro poderiam ser aprofundados outras variáveis que influenciam no desempenho da aplicação, como por exemplo, a configuração dos parâmetros utilizados no banco. Poderiam ser aprofundados mais testes com DML. Poderiam ser realizados teste com a fragmentação dos índices após muitas alterações nas tabelas.

Enfim, os resultados destes testes mostraram que a forma como o PL/SQL é escrito nas aplicações pode causar bastante diferença no desempenho da aplicação.

Ficou claro, que não basta ter apenas um bom SQL, mas um PL/SQL bem escrito também é fundamental pra o bom desempenho da aplicação.

6. REFERÊNCIAS BIBLIOGRÁFICAS

AUTOMATIC Database Diagnostic Monitor (ADDM) in Oracle Database 10g. Disponível em < http://www.oracle-base.com/articles/10g/automatic-database-diagnostic-monitor-10g.php> Acesso em 01/06/2013.

BURLESON, 2009**. Oracle Bind Variable Tips**. Disponível em <http://www.dba-oracle.com/t_bind_variables.htm> Acesso em 10/06/2013.

BURLESON, 2009**. ORACLE SQL EXECUTION STEPS**. Disponível em <http://www.dba-oracle.com/t_sql_exeution_steps.htm> Acesso em 29/04/2013.

BURLESON. **Oracle Concepts - SGA System Global Area**. Disponível em <http://www.dba-oracle.com/concepts/sga_system_global_area.htm> Acessado em 20/06/2013.

CHAUDHURI, Surajit ;NARASAYYA, Vivek; **Self-Tuning Database Systems: A Decade of Progress.** Disponível em <http://www.cs.washington.edu/education/courses/cse544/07au/lecture-notes/lecture15/lecture15.pdf> Acesso em 15/05/2013.

DOCUMENTAÇÃO do MySQL disponível em <http://dev.mysql.com/doc/refman/5.1/en/stored-routines.html> Acesso em 21/05/2013.

DOCUMENTAÇÃO da Oracle. Dipsonível em < http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/overview.htm#LNPLS001 > Acesso em 21/05/2013

DOCUMENTAÇÃO da Oracle. Disponível em <http://docs.oracle.com/cd/B10500_01/appdev.920/a96624/11_dynam.htm> Acesso em 16/05/2013

DOCUMENTAÇÃO da Oracle. Disponível em <http://docs.oracle.com/cd/B10500_01/appdev.920/a96590/adg09dyn.htm> Acesso em 05/06/2013

DOCUMENTAÇÃO da Oracle. Disponível em <http://docs.oracle.com/cd/B10500_01/appdev.920/a96590/adg09dyn.htm#26586> Acesso em 10/06/2013

DOCUMENTAÇÃO da Oracle. Disponível em <http://docs.oracle.com/cd/B28359_01/appdev.111/b28370/tuning.htm#BCGFGECG > Acesso em 11/06/2013

DOCUMENTAÇÃO da Oracle. Disponível em < http://docs.oracle.com/cd/B19306_01/appdev.102/b14251/adfns_dynamic_sql.htm#i1006243> Acesso em 16/11/2013

ELMASRI, Ramez; NAVATHE, Shamkant B. **Fundamentals Of Database Systems** - Fourth Edition. Pearson Addison Wesley, 2003.

FEUERSTEIN, Steven. Better to Best NDS. **Oracle Magazine**. Nov. Dez.2004. Disponível em <http://www.oracle.com/technetwork/issue-archive/o64sql-095035.html> Acesso em 23/05/2013.

FEUERSTEIN, Steven. Bulk Processing with BULK COLLECT and FORALL. **Oracle Magazine.** Set.Dez.2012. Disponível em < http://www.oracle.com/technetwork/issue-archive/2012/12-sep/o52plsql-1709862.html > Acesso em 03/06/2013.

FEUERSTEIN, Steven. Working with Cursors. **Oracle Magazine.** Mar.2013. Disponível em <http://www.oracle.com/technetwork/issue-archive/2013/13-mar/o23plsql-1906474.html> Acesso em 03/06/2013.

MILENE, Kenia. **Regras básicas de tuning de banco de dados**. Disponível em <http://keniamilene.wordpress.com/2007/10/25/regras-basicas-de-tuning-de-banco-de-dados/> Acesso em 24/03/2013.

SHASHA, Dennis; BONNET, Philippe. **Database Tuning: Principles, Experiments, and Troubleshooting Techniques.** 2003 by Elsevier Science (USA).

ANEXO 1

```
DECLARE
   tab nomes dbms_sql.varchar2a;
   tab sobrenomes dbms_sql.varchar2a;
   v_count NUMBER;

   v data inicio TIMESTAMP;
   v_data_fim    TIMESTAMP;


   TYPE TABELA EMPREGADOS IS TABLE OF EMPREGADOS%ROWTYPE INDEX BY
BINARY INTEGER;
   tab_empregados TABELA_EMPREGADOS;

   FUNCTION RETORNA_DATA_ADMISSAO RETURN DATE IS
      v dia NUMBER;
      v mes NUMBER;
      v ano NUMBER;
      v data DATE;
   BEGIN
      v dia := ROUND(DBMS_RANDOM.VALUE(1,28));
      v mes := ROUND(DBMS_RANDOM.VALUE(1,12));
      v ano := ROUND(DBMS_RANDOM.VALUE(1998,2012));
      v data := TO DATE(v_dia||'/'||v_mes||'/'||v_ano,'DD/MM/RRRR');
      RETURN v data;
   END RETORNA_DATA_ADMISSAO;

   FUNCTION RETORNA EMAIL( pi nome      VARCHAR2
                         , pi sobrenome VARCHAR2
                         ) RETURN VARCHAR2 IS
      tab emails dbms_sql.varchar2a;
      v indice    NUMBER;
      v email     VARCHAR2(100);
   BEGIN
     tab emails(1) := 'GMAIL.COM';
     tab emails(2) := 'GMAIL.COM';
     tab emails(3) := 'HOTMAIL.COM';
     tab emails(4) := 'YAHOO.COM.BR';
     tab_emails(5) := 'GOOGLE.COM';

     v email := UPPER(pi nome||'.'||pi sobrenome||
'@'||tab_emails(ROUND(DBMS_RANDOM.VALUE(1,5))));

     RETURN v_email;

   END RETORNA_EMAIL;
BEGIN
   -- Nomes
   tab nomes(tab nomes.count+1) := 'Arthur';
   tab nomes(tab nomes.count+1) := 'Liamara';
   tab_nomes(tab_nomes.count+1) := 'Letycia';
   .
   .
   .
   tab nomes(tab nomes.count+1) := 'Alexander';
   tab nomes(tab nomes.count+1) := 'Jason';
   tab_nomes(tab_nomes.count+1) := 'Mattea';




   -- sobrenomes
   tab sobrenomes(tab sobrenomes.count+1) := 'Abel';
   tab sobrenomes(tab sobrenomes.count+1) := 'Fripp';
   tab_sobrenomes(tab_sobrenomes.count+1) := 'Higgins';
   .
```

```
   .
   .
   tab sobrenomes(tab sobrenomes.count+1) := 'ZOZ';
   tab sobrenomes(tab sobrenomes.count+1) := 'ZUNINO';
   tab_sobrenomes(tab_sobrenomes.count+1) := 'ZVANG';

   v count := 0;
   v data inicio := SYSTIMESTAMP;
   FOR r nome IN 1..tab nomes.COUNT LOOP
      FOR r sobrenome IN 1..tab sobrenomes.COUNT LOOP
         v count := v count + 1;
         tab empregados(v count).EMPREGADO ID := SEQ ID EMPREGADOS.NEXTVAL;
         tab empregados(v count).NOME := tab nomes(r nome);
         tab empregados(v count).SOBRENOME := tab sobrenomes(r sobrenome);
         tab empregados(v count).DATA ADMISSAO := RETORNA_DATA_ADMISSAO;
         tab empregados(v count).EMAIL :=
RETORNA EMAIL(tab nomes(r nome),tab sobrenomes(r_sobrenome));
         --tab empregados(v count).CARGO ID
         tab empregados(v count).SALARIO := 0;
         --tab empregados(v count).GERENTE ID
         --tab empregados(v_count).DEPARTAMENTO_ID
      END LOOP r sobrenome;
   END LOOP r_nome;

   /*FOR r emp IN 1..tab empregados.COUNT LOOP
      DBMS OUTPUT.PUT LINE('Nome: '||tab empregados(r emp).nome ||'
Sobrenome: '||tab empregados(r emp).sobrenome||' dt. admissão:
'||to char(tab empregados(r emp).data_admissao,'dd/mm/rrrr')||' email:
'||tab empregados(r emp).email);
   END LOOP r emp;*/
   DBMS_OUTPUT.PUT_LINE ('TOTAL: '||v_count);

   v_data_fim := SYSTIMESTAMP;

   DBMS_OUTPUT.PUT_LINE(v_data_fim - v_data_inicio);

   v_data_inicio := SYSTIMESTAMP;

   FORALL r emp IN 1..tab empregados.COUNT
      INSERT INTO empregados ( EMPREGADO_ID
                             , NOME
                             , SOBRENOME
                             , DATA ADMISSAO
                             , EMAIL
                             , SALARIO
                             )
           VALUES ( tab empregados(r emp).empregado_id
                  , tab empregados(r emp).nome
                  , tab empregados(r emp).sobrenome
                  , tab empregados(r emp).data admissao
                  , tab empregados(r emp).email
                  , tab_empregados(r_emp).salario
                  );
   v_data_fim := SYSTIMESTAMP;

   DBMS_OUTPUT.PUT_LINE(v_data_fim - v_data_inicio);

END;
```

ANEXO 2

```
-- President
UPDATE (
  SELECT *
    FROM empregados
   WHERE nome like '%FELIPE%'
     AND SOBRENOME LIKE 'BOGO')
     SET CARGO_ID = (SELECT CARGO_ID FROM CARGOS WHERE DESC_CARGO =
'President')
       , SALARIO = (SELECT SALARIO_MAX FROM CARGOS WHERE DESC_CARGO =
'President')
       , DEPARTAMENTO ID = (SELECT DEPARTAMENTO_ID FROM DEPARTAMENTOS WHERE
NOME_DEPARTAMENTO = 'Executive')

UPDATE (
SELECT *
  FROM departamentos )
   SET GERENTE ID =  20906297
 WHERE nome_departamento = 'Executive'


-- Administration Vice President
UPDATE(
SELECT *
    FROM empregados
   WHERE nome like '%MOACIR%'
     AND SOBRENOME LIKE 'BOGO')
     SET CARGO ID = (SELECT CARGO_ID FROM CARGOS WHERE DESC_CARGO =
'Administration Vice President')
       , SALARIO = (SELECT SALARIO_MAX FROM CARGOS WHERE DESC_CARGO =
'Administration Vice President')
       , GERENTE ID = 20906297
       , DEPARTAMENTO ID = (SELECT DEPARTAMENTO_ID FROM DEPARTAMENTOS WHERE
NOME_DEPARTAMENTO = 'Executive')


-- Sales Manager
UPDATE(
  SELECT *
    FROM empregados
   WHERE nome like '%LETÍCIA%'
     AND SOBRENOME LIKE 'BOGO')
    SET CARGO_ID = (SELECT CARGO_ID FROM CARGOS WHERE DESC_CARGO = 'Sales
Manager')
       , SALARIO = (SELECT SALARIO_MAX FROM CARGOS WHERE DESC_CARGO =
'Sales Manager')
       , DEPARTAMENTO ID = (SELECT DEPARTAMENTO_ID FROM DEPARTAMENTOS WHERE
NOME_DEPARTAMENTO = 'Sales')

UPDATE (
SELECT *
  FROM departamentos )
   SET GERENTE ID =  21330089
 WHERE nome_departamento = 'Sales'


-- Accounting Manager
UPDATE(
   SELECT *
     FROM empregados
    WHERE nome like '%JOÃO%'
      AND SOBRENOME LIKE 'BOGO'
      AND EMPREGADO ID = 20962025)
      SET CARGO ID = (SELECT CARGO_ID FROM CARGOS WHERE DESC_CARGO =
'Accounting Manager')
       , SALARIO = (SELECT SALARIO_MAX FROM CARGOS WHERE DESC_CARGO =
'Accounting Manager')
```

```
       , DEPARTAMENTO ID = (SELECT DEPARTAMENTO_ID FROM DEPARTAMENTOS WHERE
NOME_DEPARTAMENTO = 'Accounting')

UPDATE (
SELECT *
  FROM departamentos )
   SET GERENTE ID =  20962025
 WHERE nome_departamento = 'Accounting'


-- Finance Manager
UPDATE(
   SELECT *
    FROM empregados
   WHERE nome like '%GUSTAVO%'
     AND EMPREGADO ID = 20923549)
     SET CARGO ID = (SELECT CARGO_ID FROM CARGOS WHERE DESC_CARGO =
'Finance Manager')
       , SALARIO = (SELECT SALARIO_MAX FROM CARGOS WHERE DESC_CARGO =
'Finance Manager')
       , DEPARTAMENTO ID = (SELECT DEPARTAMENTO_ID FROM DEPARTAMENTOS WHERE
NOME_DEPARTAMENTO = 'Finance')

UPDATE (
SELECT *
  FROM departamentos )
   SET GERENTE ID =  20923549
 WHERE nome_departamento = 'Finance'


-- Marketing Manager
UPDATE(
   SELECT *
    FROM empregados
   WHERE nome like '%PAULA%'
     AND EMPREGADO ID = 21192593)
     SET CARGO ID = (SELECT CARGO_ID FROM CARGOS WHERE DESC_CARGO =
'Marketing Manager')
       , SALARIO = (SELECT SALARIO_MAX FROM CARGOS WHERE DESC_CARGO =
'Marketing Manager')
       , DEPARTAMENTO ID = (SELECT DEPARTAMENTO_ID FROM DEPARTAMENTOS WHERE
NOME_DEPARTAMENTO = 'Marketing')

UPDATE (
SELECT *
  FROM departamentos )
   SET GERENTE ID =  21192593
 WHERE nome_departamento = 'Marketing'



-- Purchasing Manager
UPDATE(
   SELECT *
    FROM empregados
   WHERE nome like '%BRUNA%'
     AND EMPREGADO ID = 21197815)
     SET CARGO ID = (SELECT CARGO_ID FROM CARGOS WHERE DESC_CARGO =
'Purchasing Manager')
       , SALARIO = (SELECT SALARIO_MAX FROM CARGOS WHERE DESC_CARGO =
'Purchasing Manager')
       , DEPARTAMENTO ID = (SELECT DEPARTAMENTO_ID FROM DEPARTAMENTOS WHERE
NOME_DEPARTAMENTO = 'Purchasing')

UPDATE (
SELECT *
  FROM departamentos )
   SET GERENTE ID =  21197815
 WHERE nome_departamento = 'Purchasing'
```

```
-- Public Relations Representative
UPDATE (
   SELECT *
    FROM empregados
   WHERE nome like '%LUIS%'
     AND SOBRENOME LIKE 'ANTUNES')
     SET CARGO ID = (SELECT CARGO_ID FROM CARGOS WHERE DESC_CARGO = 'Public
Relations Representative')
       , SALARIO = (SELECT SALARIO_MAX FROM CARGOS WHERE DESC_CARGO =
'Public Relations Representative')
       , DEPARTAMENTO ID = (SELECT DEPARTAMENTO_ID FROM DEPARTAMENTOS WHERE
NOME_DEPARTAMENTO = 'Public Relations')

UPDATE (
SELECT *
  FROM departamentos )
   SET GERENTE ID =  21580140
 WHERE nome_departamento = 'Public Relations'

-- Programmer
UPDATE (
SELECT *
    FROM empregados
   WHERE nome like '%MARCOS%'
     AND EMPREGADO ID = 21005468)
     SET CARGO_ID = (SELECT CARGO_ID FROM CARGOS WHERE DESC_CARGO =
'Programmer')
       , SALARIO = (SELECT SALARIO_MAX FROM CARGOS WHERE DESC_CARGO =
'Programmer')
       , DEPARTAMENTO ID = (SELECT DEPARTAMENTO_ID FROM DEPARTAMENTOS WHERE
NOME_DEPARTAMENTO = 'IT')

UPDATE (
SELECT *
  FROM departamentos )
   SET GERENTE ID =  21005468
 WHERE nome_departamento = 'IT'


-- Human Resources Representative
UPDATE(
SELECT *
    FROM empregados
   WHERE nome like '%DANIEL%'
     AND EMPREGADO ID = 20882945)
     SET CARGO ID = (SELECT CARGO_ID FROM CARGOS WHERE DESC_CARGO = 'Human
Resources Representative')
       , SALARIO = (SELECT SALARIO_MAX FROM CARGOS WHERE DESC_CARGO =
'Human Resources Representative')
       , DEPARTAMENTO ID = (SELECT DEPARTAMENTO_ID FROM DEPARTAMENTOS WHERE
NOME_DEPARTAMENTO = 'Human Resources')

UPDATE (
SELECT *
  FROM departamentos )
   SET GERENTE ID =  20882945
 WHERE nome_departamento = 'Human Resources'

-- Stock Manager
UPDATE(
SELECT *
    FROM empregados
   WHERE nome like '%JULIANA%'
     AND EMPREGADO ID = 21302534)
     SET CARGO_ID = (SELECT CARGO_ID FROM CARGOS WHERE DESC_CARGO = 'Stock
Manager')
```

```
       , SALARIO = (SELECT SALARIO_MAX FROM CARGOS WHERE DESC_CARGO =
'Stock Manager')
       , DEPARTAMENTO ID = (SELECT DEPARTAMENTO_ID FROM DEPARTAMENTOS WHERE
NOME_DEPARTAMENTO = 'Shipping')

UPDATE (
SELECT *
  FROM departamentos )
   SET GERENTE ID =  21302534
 WHERE nome_departamento = 'Shipping'


-- Administration Assistant
UPDATE(
SELECT *
    FROM empregados
   WHERE nome like '%VANESSA%'
     AND EMPREGADO ID = 21457002)
     SET CARGO ID = (SELECT CARGO_ID FROM CARGOS WHERE DESC_CARGO =
'Administration Assistant')
       , SALARIO = (SELECT SALARIO_MAX FROM CARGOS WHERE DESC_CARGO =
'Administration Assistant')
       , DEPARTAMENTO ID = (SELECT DEPARTAMENTO_ID FROM DEPARTAMENTOS WHERE
NOME_DEPARTAMENTO = 'Administration')

UPDATE (
SELECT *
  FROM departamentos )
   SET GERENTE ID =  21457002
        WHERE nome_departamento = 'Administration'
```

ANEXO 3

```
DECLARE
   v total  NUMBER;
   v rateio NUMBER;
   TYPE tabela empregados IS TABLE OF empregados%ROWTYPE INDEX BY
BINARY INTEGER;
   tab emp            TABELA EMPREGADOS;
   v cargo id         EMPREGADOS.CARGO ID%TYPE;
   v departamento id EMPREGADOS.DEPARTAMENTO ID%TYPE;
   v gerente id       EMPREGADOS.GERENTE ID%TYPE;
   v_salario          EMPREGADOS.SALARIO%TYPE;

   CURSOR c cargos IS
select jobs.JOB TITLE
     , jobs.MIN SALARY
     , jobs.MAX SALARY
     , round(((1/max_salary) / 11) * 7500,2) rateio
  from hr.jobs
 where job id in (select job id
                    FROM hr.employees emp
                      JOIN hr.departments dept
                        USING (department_id)
                      JOIN hr.jobs
                        USING (job id)
                    -- não traz cargos de gerencia
                    WHERE job id NOT IN ( SELECT job id
                                            FROM hr.jobs
                                         NATURAL JOIN hr.employees emp
                                            JOIN hr.departments dept
                                              ON emp.EMPLOYEE_ID =
dept.MANAGER ID
                                           WHERE EXISTS ( SELECT 1
                                                            FROM
hr.employees emp1
                                                           WHERE
emp1.manager id = emp.employee id )
                                             AND JOB ID <> 'IT PROG')
                      AND job_title <> 'Administration Vice President')
ORDER BY MAX_SALARY;

BEGIN
   -- Busca o total de empregados
   SELECT COUNT(1)
     INTO v total
     FROM empregados;

   FOR r cargo IN c cargos LOOP
      v_rateio := r_cargo.rateio * v_total;

      -- busca empregados sem cargo e departamento dentro do rateio
      -- de forma aleatória
      SELECT *
        BULK COLLECT INTO tab_emp
        FROM( SELECT *
                FROM empregados
               WHERE cargo id IS NULL
            ORDER BY dbms random.value(1,100) )
       WHERE ROWNUM < v_rateio;

      -- Gera faixa de salário
      v salario :=
ROUND(DBMS_RANDOM.VALUE(r cargo.min_salary,r_cargo.max_salary));
      -- Busca id do cargo
      SELECT cargo id
        INTO v cargo id
        FROM cargos c
       WHERE c.desc_cargo = r_cargo.job_title;
```

```
      -- Busca id do departamento e do gerente
      BEGIN
         SELECT dept.departamento_id
              , dept.gerente id
           INTO v departamento_id
              , v gerente id
           FROM departamentos dept
          WHERE dept.nome departamento = (SELECT DISTINCT
                                                department name
                                           FROM hr.employees emp
                                           JOIN hr.jobs
                                          USING (job id)
                                           JOIN hr.departments dept
                                          USING (department id)
                                          WHERE jobs.job_title =
r cargo.job title);
      EXCEPTION
         WHEN NO DATA FOUND THEN
            v departamento id := NULL;
            v_gerente_id := NULL;
      END;

      -- Atualiza empregados.
      FOR r emp IN 1..tab emp.COUNT LOOP
         tab emp(r emp).cargo id := v_cargo_id;
         tab emp(r emp).salario  :=
ROUND(DBMS_RANDOM.VALUE(r cargo.min salary,r cargo.max salary));
         tab emp(r emp).departamento id := v departamento_id;
         tab_emp(r_emp).gerente_id := v_gerente_id;

      END LOOP r_emp;

      FORALL r emp IN 1..tab_emp.COUNT
         UPDATE empregados
            SET ROW = tab emp(r emp)
          WHERE empregado_id = tab_emp(r_emp).empregado_id;
      tab emp.DELETE;
   END LOOP r_cargo;
END;
```

ANEXO 4

```
-- Consulta 1 sem BULK COLLECT
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v_indice      NUMBER;

   CURSOR c empregados IS
      SELECT emp.nome
           , emp.sobrenome
           , dept.nome departamento departamento
           , cg.desc cargo cargo
           , emp.salario
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo id)
  CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 2) emp2
    ORDER BY emp.nome
           , emp.sobrenome;
BEGIN
   v indice := 1;
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   FOR r emp IN c empregados LOOP
      v empregados(v indice).nome := r emp.nome;
      v empregados(v indice).sobrenome := r emp.sobrenome;
      v empregados(v indice).departamento := r emp.departamento;
      v empregados(v indice).cargo := r emp.cargo;
      v empregados(v indice).salario := r_emp.salario;
      v indice := v_indice + 1;
   END LOOP r emp;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 1 sem BULK COLLECT - total:
'||v_indice,2001,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 2 sem BULK COLLECT
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v_indice      NUMBER;

   CURSOR c empregados IS
      SELECT emp.nome
           , emp.sobrenome
           , dept.nome departamento departamento
           , cg.desc cargo cargo
           , emp.salario
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo id)
  CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 3) emp2
    ORDER BY emp.nome
           , emp.sobrenome;
BEGIN
   v indice := 1;
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   FOR r emp IN c empregados LOOP
      v empregados(v indice).nome := r emp.nome;
      v empregados(v indice).sobrenome := r emp.sobrenome;
      v empregados(v indice).departamento := r emp.departamento;
      v empregados(v indice).cargo := r emp.cargo;
      v empregados(v indice).salario := r_emp.salario;
      v indice := v_indice + 1;
   END LOOP r emp;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 2 sem BULK COLLECT - total:
'||v_indice,2003,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 3 sem BULK COLLECT
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v_indice      NUMBER;

   CURSOR c empregados IS
      SELECT emp.nome
           , emp.sobrenome
           , dept.nome departamento departamento
           , cg.desc cargo cargo
           , emp.salario
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo id)
  CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 4) emp2
    ORDER BY emp.nome
           , emp.sobrenome;
BEGIN
   v indice := 1;
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   FOR r emp IN c empregados LOOP
      v empregados(v indice).nome := r emp.nome;
      v empregados(v indice).sobrenome := r emp.sobrenome;
      v empregados(v indice).departamento := r emp.departamento;
      v empregados(v indice).cargo := r emp.cargo;
      v empregados(v indice).salario := r_emp.salario;
      v indice := v_indice + 1;
   END LOOP r emp;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 3 sem BULK COLLECT - total:
'||v_indice,2005,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 4 sem BULK COLLECT
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v_indice      NUMBER;

   CURSOR c empregados IS
      SELECT emp.nome
           , emp.sobrenome
           , dept.nome departamento departamento
           , cg.desc cargo cargo
           , emp.salario
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo id)
  CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 5) emp2
    ORDER BY emp.nome
           , emp.sobrenome;
BEGIN
   v indice := 1;
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   FOR r emp IN c empregados LOOP
      v empregados(v indice).nome := r emp.nome;
      v empregados(v indice).sobrenome := r emp.sobrenome;
      v empregados(v indice).departamento := r emp.departamento;
      v empregados(v indice).cargo := r emp.cargo;
      v empregados(v indice).salario := r_emp.salario;
      v indice := v_indice + 1;
   END LOOP r emp;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 4 sem BULK COLLECT - total:
'||v_indice,2007,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 5 sem BULK COLLECT
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v_indice      NUMBER;

   CURSOR c empregados IS
      SELECT emp.nome
           , emp.sobrenome
           , dept.nome departamento departamento
           , cg.desc cargo cargo
           , emp.salario
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo id)
  CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 6) emp2
    ORDER BY emp.nome
           , emp.sobrenome;
BEGIN
   v indice := 1;
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   FOR r emp IN c empregados LOOP
      v empregados(v indice).nome := r emp.nome;
      v empregados(v indice).sobrenome := r emp.sobrenome;
      v empregados(v indice).departamento := r emp.departamento;
      v empregados(v indice).cargo := r emp.cargo;
      v empregados(v indice).salario := r_emp.salario;
      v indice := v_indice + 1;
   END LOOP r emp;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 5 sem BULK COLLECT - total:
'||v_indice,2009,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 1 com BULK COLLECT
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME_DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data_fim    TIMESTAMP;
BEGIN
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   SELECT emp.nome
        , emp.sobrenome
        , dept.nome departamento departamento
        , cg.desc cargo cargo
        , emp.salario
     BULK COLLECT INTO v_empregados
     FROM empregados emp
     JOIN departamentos dept
    USING (departamento_id)
     JOIN cargos cg
    USING (cargo id)
CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 2) emp2
 ORDER BY emp.nome
        , emp.sobrenome;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 1 com BULK COLLECT -
total:'||v_empregados.count,2002,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 2 com BULK COLLECT
begin
for c in 1..15 loop
DECLARE
   TYPE rec_empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data_fim    TIMESTAMP;
BEGIN
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   SELECT emp.nome
        , emp.sobrenome
        , dept.nome departamento departamento
        , cg.desc cargo cargo
        , emp.salario
     BULK COLLECT INTO v_empregados
     FROM empregados emp
     JOIN departamentos dept
    USING (departamento_id)
     JOIN cargos cg
    USING (cargo id)
CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 3) emp2
 ORDER BY emp.nome
        , emp.sobrenome;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 2 com BULK COLLECT -
total:'||v_empregados.count,2004,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 3 com BULK COLLECT
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data_fim    TIMESTAMP;
BEGIN
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   SELECT emp.nome
        , emp.sobrenome
        , dept.nome departamento departamento
        , cg.desc cargo cargo
        , emp.salario
     BULK COLLECT INTO v_empregados
     FROM empregados emp
     JOIN departamentos dept
    USING (departamento_id)
     JOIN cargos cg
    USING (cargo id)
CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 4) emp2
 ORDER BY emp.nome
        , emp.sobrenome;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 3 com BULK COLLECT -
total:'||v_empregados.count,2006,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 4 com BULK COLLECT
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data_fim    TIMESTAMP;
BEGIN
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   SELECT emp.nome
        , emp.sobrenome
        , dept.nome departamento departamento
        , cg.desc cargo cargo
        , emp.salario
     BULK COLLECT INTO v_empregados
     FROM empregados emp
     JOIN departamentos dept
    USING (departamento_id)
     JOIN cargos cg
    USING (cargo id)
CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 5) emp2
 ORDER BY emp.nome
        , emp.sobrenome;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 4 com BULK COLLECT -
total:'||v_empregados.count,2008,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 5 com BULK COLLECT
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data_fim    TIMESTAMP;
BEGIN
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   SELECT emp.nome
        , emp.sobrenome
        , dept.nome departamento departamento
        , cg.desc cargo cargo
        , emp.salario
     BULK COLLECT INTO v_empregados
     FROM empregados emp
     JOIN departamentos dept
    USING (departamento_id)
     JOIN cargos cg
    USING (cargo id)
CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 6) emp2
 ORDER BY emp.nome
        , emp.sobrenome;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 5 com BULK COLLECT -
total:'||v_empregados.count,2010,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 1 sem BULK COLLECT explícito
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v empregado  rec empregado;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v_indice      NUMBER;

   CURSOR c empregados IS
      SELECT emp.nome
           , emp.sobrenome
           , dept.nome departamento departamento
           , cg.desc cargo cargo
           , emp.salario
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo id)
  CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 2) emp2
    ORDER BY emp.nome
           , emp.sobrenome;
BEGIN
   v indice := 1;
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   OPEN c_empregados;
   LOOP
      FETCH c empregados INTO v empregado;
      EXIT WHEN c empregados%NOTFOUND;
      v empregados(v indice):= v_empregado;
      v indice := v_indice + 1;
   END LOOP r emp;
   CLOSE c empregados;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 1 sem BULK COLLECT cursor explícito - total:
'||v_indice,2011,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 2 sem BULK COLLECT explícito
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v empregado  rec empregado;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v_indice      NUMBER;

   CURSOR c empregados IS
      SELECT emp.nome
           , emp.sobrenome
           , dept.nome departamento departamento
           , cg.desc cargo cargo
           , emp.salario
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo id)
  CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 3) emp2
    ORDER BY emp.nome
           , emp.sobrenome;
BEGIN
   v indice := 1;
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   OPEN c_empregados;
   LOOP
      FETCH c empregados INTO v empregado;
      EXIT WHEN c empregados%NOTFOUND;
      v empregados(v indice):= v_empregado;
      v indice := v_indice + 1;
   END LOOP r emp;
   CLOSE c empregados;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 2 sem BULK COLLECT cursor explícito - total:
'||v_indice,2012,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 3 sem BULK COLLECT explícito
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v empregado  rec empregado;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v_indice      NUMBER;

   CURSOR c empregados IS
      SELECT emp.nome
           , emp.sobrenome
           , dept.nome departamento departamento
           , cg.desc cargo cargo
           , emp.salario
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo id)
  CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 4) emp2
    ORDER BY emp.nome
           , emp.sobrenome;
BEGIN
   v indice := 1;
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   OPEN c_empregados;
   LOOP
      FETCH c empregados INTO v empregado;
      EXIT WHEN c empregados%NOTFOUND;
      v empregados(v indice):= v_empregado;
      v indice := v_indice + 1;
   END LOOP r emp;
   CLOSE c empregados;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 3 sem BULK COLLECT cursor explícito - total:
'||v_indice,2013,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 4 sem BULK COLLECT explícito
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v empregado  rec empregado;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v_indice      NUMBER;

   CURSOR c empregados IS
      SELECT emp.nome
           , emp.sobrenome
           , dept.nome departamento departamento
           , cg.desc cargo cargo
           , emp.salario
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo id)
  CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 5) emp2
    ORDER BY emp.nome
           , emp.sobrenome;
BEGIN
   v indice := 1;
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   OPEN c_empregados;
   LOOP
      FETCH c empregados INTO v empregado;
      EXIT WHEN c empregados%NOTFOUND;
      v empregados(v indice):= v_empregado;
      v indice := v_indice + 1;
   END LOOP r emp;
   CLOSE c empregados;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 4 sem BULK COLLECT cursor explícito - total:
'||v_indice,2014,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 5 sem BULK COLLECT explícito
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v empregado  rec empregado;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v_indice      NUMBER;

   CURSOR c empregados IS
      SELECT emp.nome
           , emp.sobrenome
           , dept.nome departamento departamento
           , cg.desc cargo cargo
           , emp.salario
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo id)
  CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 6) emp2
    ORDER BY emp.nome
           , emp.sobrenome;
BEGIN
   v indice := 1;
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   OPEN c_empregados;
   LOOP
      FETCH c empregados INTO v empregado;
      EXIT WHEN c empregados%NOTFOUND;
      v empregados(v indice):= v_empregado;
      v indice := v_indice + 1;
   END LOOP r emp;
   CLOSE c empregados;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 5 sem BULK COLLECT cursor explícito - total:
'||v_indice,2015,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

ANEXO 5

```
-- consulta x consulta dinamica - consulta estática
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab_empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v data parc    TIMESTAMP;
   v data ini parc    TIMESTAMP;
   v_sql         VARCHAR2(4000);

   CURSOR c departamentos IS
      SELECT departamento id
        FROM departamentos;


BEGIN
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   -- Percorre todos os departamentos
   FOR r dept IN c departamentos LOOP
      v data ini parc := SYSTIMESTAMP;
      SELECT emp.nome
           , emp.sobrenome
           , dept.nome departamento departamento
           , cg.desc cargo cargo
           , emp.salario
BULK COLLECT INTO v empregados
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo id)
       WHERE departamento_id = r_dept.departamento_id
    ORDER BY emp.nome
           , emp.sobrenome;
      v data parc := SYSTIMESTAMP;
      SALVA TEMPO('consulta x consulta dinamica -
consulta',2016,(v data parc - v_data_ini_parc));
      v empregados.DELETE;
   END LOOP r dept;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('consulta x consulta dinamica - consulta',2017,(v_data_fim -
v data_inicio));
END;
end loop;
end;
```

```
begin
for c in 1..15 loop
-- consulta x consulta dinamica - consulta dinamica
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v data ini parc timestamp;
   v data parc    TIMESTAMP;
   v sql         VARCHAR2(4000);
   v_soma        NUMBER;

   CURSOR c departamentos IS
      SELECT departamento id
        FROM departamentos;

BEGIN
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   v soma := 0;
   -- Percorre todos os departamentos
   FOR r dept IN c departamentos LOOP
      -- Monta string com sql
      v sql := 'SELECT emp.nome
                     , emp.sobrenome
                     , dept.nome departamento departamento
                     , cg.desc cargo cargo
                     , emp.salario
                  FROM empregados emp
                  JOIN departamentos dept
                 USING (departamento_id)
                  JOIN cargos cg
                 USING (cargo id)
                 WHERE departamento_id = '||r_dept.departamento_id||'
              ORDER BY emp.nome
                     , emp.sobrenome';
      v data ini parc := SYSTIMESTAMP;
      EXECUTE IMMEDIATE v sql
      BULK COLLECT INTO v empregados;
      v data parc := SYSTIMESTAMP;
      SALVA TEMPO('consulta x consulta dinamica - consulta dinamica dep:
'||r dept.departamento id,6003,(v data parc - v_data_ini_parc));
      FOR r emp IN 1..v empregados.COUNT LOOP
         v soma := v soma + v_empregados(r_emp).salario;
      END LOOP r emp;
      v empregados.DELETE;
   END LOOP r dept;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('consulta x consulta dinamica - consulta dinamica:
'||v_soma,6004,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

ANEXO 6

```
-- Consulta 1 com BULK COLLECT consulta dinâmica
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v sql         VARCHAR2(4000);
BEGIN
   -- Grava hora inicio.
   v sql := 'SELECT emp.nome
        , emp.sobrenome
        , dept.nome departamento departamento
        , cg.desc cargo cargo
        , emp.salario
     FROM empregados emp
     JOIN departamentos dept
    USING (departamento_id)
     JOIN cargos cg
    USING (cargo id)
CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 2) emp2
 ORDER BY emp.nome
        , emp.sobrenome';
   v data inicio := SYSTIMESTAMP;
   EXECUTE IMMEDIATE v sql
   BULK COLLECT INTO v empregados;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 1 com BULK COLLECT -
total:'||v_empregados.count,5024,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 2 com BULK COLLECT consulta dinâmica
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v sql         VARCHAR2(4000);
BEGIN
   -- Grava hora inicio.
   v sql := 'SELECT emp.nome
        , emp.sobrenome
        , dept.nome departamento departamento
        , cg.desc cargo cargo
        , emp.salario
     FROM empregados emp
     JOIN departamentos dept
    USING (departamento_id)
     JOIN cargos cg
    USING (cargo id)
CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 3) emp2
 ORDER BY emp.nome
        , emp.sobrenome';
   v data inicio := SYSTIMESTAMP;
   EXECUTE IMMEDIATE v sql
   BULK COLLECT INTO v empregados;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 2 com BULK COLLECT -
total:'||v_empregados.count,5023,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 3 com BULK COLLECT consulta dinâmica
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v sql         VARCHAR2(4000);
BEGIN
   -- Grava hora inicio.
   v sql := 'SELECT emp.nome
        , emp.sobrenome
        , dept.nome departamento departamento
        , cg.desc cargo cargo
        , emp.salario
     FROM empregados emp
     JOIN departamentos dept
    USING (departamento_id)
     JOIN cargos cg
    USING (cargo id)
CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 4) emp2
 ORDER BY emp.nome
        , emp.sobrenome';
   v data inicio := SYSTIMESTAMP;
   EXECUTE IMMEDIATE v sql
   BULK COLLECT INTO v empregados;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 3 com BULK COLLECT -
total:'||v_empregados.count,5022,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 4 com BULK COLLECT consulta dinâmica
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v sql         VARCHAR2(4000);
BEGIN
   -- Grava hora inicio.
   v sql := 'SELECT emp.nome
        , emp.sobrenome
        , dept.nome departamento departamento
        , cg.desc cargo cargo
        , emp.salario
     FROM empregados emp
     JOIN departamentos dept
    USING (departamento_id)
     JOIN cargos cg
    USING (cargo id)
CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 5) emp2
 ORDER BY emp.nome
        , emp.sobrenome';
   v data inicio := SYSTIMESTAMP;
   EXECUTE IMMEDIATE v sql
   BULK COLLECT INTO v empregados;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 4 com BULK COLLECT -
total:'||v_empregados.count,5021,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
-- Consulta 5 com BULK COLLECT
begin
for c in 1..15 loop
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v sql         VARCHAR2(4000);
BEGIN
   -- Grava hora inicio.
   v sql := 'SELECT emp.nome
        , emp.sobrenome
        , dept.nome departamento departamento
        , cg.desc cargo cargo
        , emp.salario
     FROM empregados emp
     JOIN departamentos dept
    USING (departamento_id)
     JOIN cargos cg
    USING (cargo id)
CROSS JOIN (SELECT * FROM empregados WHERE ROWNUM <= 6) emp2
 ORDER BY emp.nome
        , emp.sobrenome';
   v data inicio := SYSTIMESTAMP;
   EXECUTE IMMEDIATE v sql
   BULK COLLECT INTO v empregados;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('Consulta 5 com BULK COLLECT -
total:'||v_empregados.count,5020,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

ANEXO 7

```
begin
for c in 1..15 loop
-- consulta x consulta dinamica - consulta dinamica
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v data ini parc timestamp;
   v data parc    TIMESTAMP;
   v sql         VARCHAR2(4000);
   v_soma        NUMBER;

   CURSOR c departamentos IS
      SELECT departamento id
        FROM departamentos;

BEGIN
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   v soma := 0;
   -- Percorre todos os departamentos
   FOR r dept IN c departamentos LOOP
      -- Monta string com sql
      v sql := 'SELECT emp.nome
                     , emp.sobrenome
                     , dept.nome departamento departamento
                     , cg.desc cargo cargo
                     , emp.salario
                  FROM empregados emp
                  JOIN departamentos dept
                 USING (departamento_id)
                  JOIN cargos cg
                 USING (cargo id)
                 WHERE departamento_id = '||r_dept.departamento_id||'
              ORDER BY emp.nome
                     , emp.sobrenome';
      v data ini parc := SYSTIMESTAMP;
      EXECUTE IMMEDIATE v sql
      BULK COLLECT INTO v empregados;
      v data parc := SYSTIMESTAMP;
      SALVA TEMPO('consulta x consulta dinamica - consulta dinamica dep:
'||r dept.departamento id,6003,(v data parc - v_data_ini_parc));
      FOR r emp IN 1..v empregados.COUNT LOOP
         v soma := v soma + v_empregados(r_emp).salario;
      END LOOP r emp;
      v empregados.DELETE;
   END LOOP r dept;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('consulta x consulta dinamica - consulta dinamica:
'||v_soma,6004,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
begin
for c in 1..15 loop
-- consulta x consulta dinamica - consulta dinamica bind
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v data ini parc timestamp;
   v data parc    TIMESTAMP;
   v sql         VARCHAR2(4000);
   v_soma        NUMBER;

   CURSOR c departamentos IS
      SELECT departamento id
        FROM departamentos;

BEGIN
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   v soma := 0;
   -- Percorre todos os departamentos
   FOR r dept IN c departamentos LOOP
      -- Monta string com sql
      v sql := 'SELECT emp.nome
                     , emp.sobrenome
                     , dept.nome departamento departamento
                     , cg.desc cargo cargo
                     , emp.salario
                  FROM empregados emp
                  JOIN departamentos dept
                 USING (departamento_id)
                  JOIN cargos cg
                 USING (cargo id)
                 WHERE departamento_id = :1
              ORDER BY emp.nome
                     , emp.sobrenome';
      v data ini parc := SYSTIMESTAMP;
      EXECUTE IMMEDIATE v sql
      BULK COLLECT INTO v empregados
                  USING r dept.departamento_id;
      v data parc := SYSTIMESTAMP;
      FOR r emp IN 1..v empregados.COUNT LOOP
         v soma := v_soma + v_empregados(r_emp).salario;
      END LOOP;
      SALVA TEMPO('consulta x consulta dinamica - consulta dinamica bind
dep: '||r dept.departamento_id,6001,(v_data_parc - v_data_ini_parc));
      v empregados.DELETE;
   END LOOP r dept;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('consulta x consulta dinamica - consulta dinamica bind:
'||v_soma,6002,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

```
begin
for c in 1..15 loop
-- consulta x consulta dinamica - consulta dinamica cursor explícito
DECLARE
   TYPE CURSOR EMPREGADOS IS REF CURSOR;
   c emp CURSOR EMPREGADOS;
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v data ini parc timestamp;
   v data parc    TIMESTAMP;
   v sql         VARCHAR2(4000);
   v indice      NUMBER;
   v_soma        NUMBER;

   CURSOR c departamentos IS
      SELECT departamento id
        FROM departamentos;

BEGIN
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   -- Percorre todos os departamentos
   v soma := 0;
   FOR r dept IN c departamentos LOOP
      -- Monta string com sql
      v sql := 'SELECT emp.nome
                     , emp.sobrenome
                     , dept.nome departamento departamento
                     , cg.desc cargo cargo
                     , emp.salario
                  FROM empregados emp
                  JOIN departamentos dept
                 USING (departamento_id)
                  JOIN cargos cg
                 USING (cargo id)
                 WHERE departamento_id = '||r_dept.departamento_id||'
              ORDER BY emp.nome
                     , emp.sobrenome';
      v indice := 1;
      v data ini parc := SYSTIMESTAMP;
      OPEN c_emp FOR v_sql;
      LOOP
         FETCH c emp INTO v empregados(v_indice);
         EXIT WHEN c emp%NOTFOUND;
         v soma := v soma + v empregados(v_indice).salario;
         v indice := v_indice + 1;
      END LOOP;
      v data parc := SYSTIMESTAMP;
      SALVA TEMPO('consulta x consulta dinamica - consulta dinamica cursor
explícito dept: '||r_dept.departamento_id,6007,(v_data_parc -
v data ini parc));
      v empregados.DELETE;
   END LOOP r dept;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('consulta x consulta dinamica - consulta dinamica cursor
explícito: '||v_soma,6008,(v_data_fim - v_data_inicio));
END;
end loop;end;
```

```
begin for c in 1..15 loop
-- consulta x consulta dinamica - consulta dinamica com cursor explícito
bind
DECLARE
   TYPE CURSOR EMPREGADOS IS REF CURSOR;
   c emp CURSOR EMPREGADOS;
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v data ini parc timestamp;
   v data parc    TIMESTAMP;
   v sql         VARCHAR2(4000);
   v indice      NUMBER;
   v_soma        NUMBER;

   CURSOR c departamentos IS
      SELECT departamento id
        FROM departamentos;

BEGIN
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   -- Percorre todos os departamentos
   v soma := 0;
   FOR r dept IN c departamentos LOOP
      -- Monta string com sql
      v sql := 'SELECT emp.nome
                     , emp.sobrenome
                     , dept.nome departamento departamento
                     , cg.desc cargo cargo
                     , emp.salario
                  FROM empregados emp
                  JOIN departamentos dept
                 USING (departamento_id)
                  JOIN cargos cg
                 USING (cargo id)
                 WHERE departamento_id = :1
              ORDER BY emp.nome
                     , emp.sobrenome';
      v indice := 1;
      v data ini parc := SYSTIMESTAMP;
      OPEN c emp FOR v sql
      USING r_dept.departamento_id;
      LOOP
         FETCH c emp INTO v empregados(v_indice);
         EXIT WHEN c emp%NOTFOUND;
         v soma := v soma + v empregados(v_indice).salario;
         v indice := v_indice + 1;
      END LOOP;
      v data parc := SYSTIMESTAMP;
      SALVA TEMPO('consulta x consulta dinamica - consulta dinamica cursor
explícito bind dept: '||r_dept.departamento_id,6005,(v_data_parc -
v data ini parc));
      v empregados.DELETE;
   END LOOP r dept;
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('consulta x consulta dinamica - consulta dinamica cursor
explícito bind: '||v_soma,6006,(v_data_fim - v_data_inicio));
END;
end loop; end;
```

ANEXO 8

```
begin
for c in 1..15 loop
-- consulta x consulta dinamica - consulta dinamica dbms_sql bind
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   v empregados tab empregados;
   v empregado  rec empregado;
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v data ini parc TIMESTAMP;
   v data parc    TIMESTAMP;
   v sql          VARCHAR2(4000);
   v indice       NUMBER;
   v soma         NUMBER;
   v cursor       NUMBER;
   v_cont         NUMBER;

   CURSOR c departamentos IS
      SELECT departamento id
        FROM departamentos;

BEGIN
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   -- Percorre todos os departamentos
   v soma := 0;
   FOR r dept IN c departamentos LOOP
      -- Monta string com sql
      v sql := 'SELECT emp.nome
                    , emp.sobrenome
                    , dept.nome departamento departamento
                    , cg.desc cargo cargo
                    , emp.salario
                 FROM empregados emp
                 JOIN departamentos dept
                USING (departamento_id)
                 JOIN cargos cg
                USING (cargo id)
                WHERE departamento_id = :departamento_id
             ORDER BY emp.nome
                    , emp.sobrenome';
      v data ini parc := SYSTIMESTAMP;
      -- Abre cursor
      v cursor := DBMS_SQL.OPEN_CURSOR;
      DBMS_SQL.PARSE(v cursor, v_sql, DBMS_SQL.NATIVE);
      -- Fornece variáveis bind

DBMS_SQL.BIND_VARIABLE(v cursor,'departamento_id',r_dept.departamento_id);
      -- Descreve definição de variáveis
      DBMS_SQL.DEFINE_COLUMN(v cursor, 1, v empregado.nome, 30);
      DBMS_SQL.DEFINE_COLUMN(v cursor, 2, v empregado.sobrenome, 30);
      DBMS_SQL.DEFINE_COLUMN(v cursor, 3, v empregado.departamento, 50);
      DBMS_SQL.DEFINE_COLUMN(v cursor, 4, v empregado.cargo, 50);
      DBMS_SQL.DEFINE_COLUMN(v_cursor, 5, v_empregado.salario);
      -- Executa cursor
      v cont := DBMS_SQL.EXECUTE(v_cursor);
      v indice := 1;
      LOOP
         IF DBMS_SQL.FETCH_ROWS(v_cursor) > 0 THEN
```

```
            -- Atribui valores
            DBMS_SQL.COLUMN_VALUE(v_cursor, 1, v_empregado.nome);
            DBMS_SQL.COLUMN_VALUE(v_cursor, 2, v_empregado.sobrenome);
            DBMS_SQL.COLUMN_VALUE(v_cursor, 3, v_empregado.departamento);
            DBMS_SQL.COLUMN_VALUE(v_cursor, 4, v_empregado.cargo);
            DBMS_SQL.COLUMN_VALUE(v_cursor, 5, v_empregado.salario);
            v_empregados(v_indice) := v_empregado;
            v_soma := v_soma + v_empregado.salario;
            v_indice := v_indice + 1;
         ELSE
            EXIT;
         END IF;
      END LOOP;
      DBMS_SQL.CLOSE_CURSOR(v_cursor);
      v_data_parc := SYSTIMESTAMP;
      SALVA_TEMPO('consulta x consulta dinamica - consulta dinamica cursor
dbms_sql bind dept: '||r_dept.departamento_id,6009,(v_data_parc -
v_data_ini_parc));
      v_empregados.DELETE;
   END LOOP r_dept;
   v_data_fim := SYSTIMESTAMP;
   SALVA_TEMPO('consulta x consulta dinamica - consulta dinamica cursor
dbms_sql bind: '||v_soma,6010,(v_data_fim - v_data_inicio));
END;
end loop;
end;
```

ANEXO 9

```
-- Insert cursor
BEGIN
FOR C IN 1..15 LOOP
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   --v empregados tab empregados;
   v info        VARCHAR2(1000);
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v_indice      NUMBER;

   CURSOR c empregados IS
      SELECT emp.nome
           , emp.sobrenome
           , dept.nome departamento departamento
           , cg.desc cargo cargo
           , emp.salario
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo id)
    ORDER BY emp.nome
           , emp.sobrenome;
BEGIN
   EXECUTE IMMEDIATE 'TRUNCATE TABLE PESQUISA';
   v indice := 1;
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   FOR r emp IN c empregados LOOP
      v info := 'Nome: '||r emp.nome||
                ' Sobrenome: '||r emp.sobrenome||
                ' Departamento:'||r emp.departamento||
                ' Cargo: '|| r emp.cargo||
                ' Salário: '||r_emp.salario;

      INSERT INTO PESQUISA
                  ( USUARIO
                  , sequencia
                  , texto01
                  , id
                  )
             VALUES
                  ( 'DML - INSERT - CURSOR'
                  , 7001
                  , v info
                  , SEQ_ID_PESQUISA.NEXTVAL
                  );
   END LOOP r emp;
   -- Grava hora q terminou de inserir os dados
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('DML - INSERT - CURSOR',7001,(v_data_fim - v_data_inicio));
   COMMIT;
END;
END LOOP;
END;
```

```
-- Insert forall
BEGIN
FOR C IN 1..15 LOOP
DECLARE
   v informacoes DBMS_SQL.VARCHAR2A;
   v data inicio TIMESTAMP;
   v_data_fim    TIMESTAMP;

BEGIN
   EXECUTE IMMEDIATE 'TRUNCATE TABLE PESQUISA';
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   SELECT 'Nome: '||emp.nome||
          ' Sobrenome: '||emp.sobrenome||
          ' Departamento:'||dept.nome departamento||
          ' Cargo: '|| cg.desc cargo||
          ' Salário: '||emp.salario
BULK COLLECT INTO v informacoes
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo id)
       ORDER BY emp.nome
           , emp.sobrenome;

   -- Popula vetor de informações
   FORALL r info IN 1..v informacoes.COUNT
      INSERT INTO PESQUISA
                  ( USUARIO
                  , sequencia
                  , texto01
                  , id
                  )
             VALUES
                  ( 'DML - INSERT - FORALL'
                  , 7002
                  , v informacoes(r info)
                  , SEQ_ID_PESQUISA.NEXTVAL
                  );
   -- Grava hora q terminou de inserir os dados
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('DML - INSERT - FORALL',7002,(v_data_fim - v_data_inicio));
   COMMIT;
END;
END LOOP;
END;
```

```
-- Insert direto
BEGIN
FOR C IN 1..15 LOOP
DECLARE
   v informacoes DBMS_SQL.VARCHAR2A;
   v data inicio TIMESTAMP;
   v_data_fim    TIMESTAMP;

BEGIN
   EXECUTE IMMEDIATE 'TRUNCATE TABLE PESQUISA';
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   INSERT INTO PESQUISA
                ( USUARIO
                , sequencia
                , texto01
                , id
                )
   SELECT USUARIO
        , SEQUENCIA
        , TEXTO
        , SEQ ID PESQUISA.NEXTVAL
     FROM ( SELECT 'DML - INSERT - DIRETO' USUARIO
                 , 7003                    SEQUENCIA
                 , 'Nome: '||emp.nome||
                   ' Sobrenome: '||emp.sobrenome||
                   ' Departamento:'||dept.nome departamento||
                   ' Cargo: '|| cg.desc cargo||
                   ' Salário: '||emp.salario  TEXTO
              FROM empregados emp
              JOIN departamentos dept
             USING (departamento_id)
              JOIN cargos cg
             USING (cargo id)
          ORDER BY emp.nome
                 , emp.sobrenome);
   -- Grava hora q terminou de inserir os dados
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('DML - INSERT - DIRETO',7003,(v_data_fim - v_data_inicio));
   COMMIT;
END;
END LOOP;
END;
```

```
-- Insert cursor sem order by
BEGIN
FOR C IN 1..15 LOOP
DECLARE
   TYPE rec empregado IS RECORD
   ( nome               EMPREGADOS.NOME%TYPE
   , sobrenome          EMPREGADOS.SOBRENOME%TYPE
   , departamento       DEPARTAMENTOS.NOME DEPARTAMENTO%TYPE
   , cargo              CARGOS.DESC CARGO%TYPE
   , salario            EMPREGADOS.SALARIO%TYPE
   );
   TYPE tab empregados IS TABLE OF rec_empregado INDEX BY BINARY_INTEGER;
   --v empregados tab empregados;
   v info        VARCHAR2(1000);
   v data inicio TIMESTAMP;
   v data fim    TIMESTAMP;
   v_indice      NUMBER;

   CURSOR c empregados IS
      SELECT emp.nome
           , emp.sobrenome
           , dept.nome departamento departamento
           , cg.desc cargo cargo
           , emp.salario
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo_id);
BEGIN
   EXECUTE IMMEDIATE 'TRUNCATE TABLE PESQUISA';
   v indice := 1;
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   FOR r emp IN c empregados LOOP
      v info := 'Nome: '||r emp.nome||
                ' Sobrenome: '||r emp.sobrenome||
                ' Departamento:'||r emp.departamento||
                ' Cargo: '|| r emp.cargo||
                ' Salário: '||r_emp.salario;

      INSERT INTO PESQUISA
                  ( USUARIO
                  , sequencia
                  , texto01
                  , id
                  )
            VALUES
                  ( 'DML - INSERT - CURSOR 2'
                  , 7006
                  , v info
                  , SEQ_ID_PESQUISA.NEXTVAL
                  );
   END LOOP r emp;
   -- Grava hora q terminou de inserir os dados
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('DML - INSERT - CURSOR 2',7006,(v_data_fim -
v data inicio));
   COMMIT;
END;
END LOOP;
END;
```

```
-- Insert forall sem order by
BEGIN
FOR C IN 1..15 LOOP
DECLARE
   v informacoes DBMS_SQL.VARCHAR2A;
   v data inicio TIMESTAMP;
   v_data_fim    TIMESTAMP;

BEGIN
   EXECUTE IMMEDIATE 'TRUNCATE TABLE PESQUISA';
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   SELECT 'Nome: '||emp.nome||
          ' Sobrenome: '||emp.sobrenome||
          ' Departamento:'||dept.nome departamento||
          ' Cargo: '|| cg.desc cargo||
          ' Salário: '||emp.salario
BULK COLLECT INTO v informacoes
        FROM empregados emp
        JOIN departamentos dept
       USING (departamento_id)
        JOIN cargos cg
       USING (cargo_id);

   -- Popula vetor de informações
   FORALL r info IN 1..v informacoes.COUNT
      INSERT INTO PESQUISA
                  ( USUARIO
                  , sequencia
                  , texto01
                  , id
                  )
             VALUES
                  ( 'DML - INSERT - FORALL 2'
                  , 7005
                  , v informacoes(r info)
                  , SEQ_ID_PESQUISA.NEXTVAL
                  );
   -- Grava hora q terminou de inserir os dados
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('DML - INSERT - FORALL 2',7005,(v_data_fim -
v data inicio));
   COMMIT;
END;
END LOOP;
END;
```

```
-- Insert direto sem order by
BEGIN
FOR C IN 1..15 LOOP
DECLARE
   v informacoes DBMS_SQL.VARCHAR2A;
   v data inicio TIMESTAMP;
   v_data_fim    TIMESTAMP;

BEGIN
   EXECUTE IMMEDIATE 'TRUNCATE TABLE PESQUISA';
   -- Grava hora inicio.
   v data inicio := SYSTIMESTAMP;
   INSERT INTO PESQUISA
                ( USUARIO
                , sequencia
                , texto01
                , id
                )
   SELECT 'DML - INSERT - DIRETO 2' USUARIO
        , 7004                  SEQUENCIA
        , 'Nome: '||emp.nome||
          ' Sobrenome: '||emp.sobrenome||
          ' Departamento:'||dept.nome departamento||
          ' Cargo: '|| cg.desc cargo||
          ' Salário: '||emp.salario  TEXTO
        , SEQ ID PESQUISA.NEXTVAL
     FROM empregados emp
     JOIN departamentos dept
    USING (departamento_id)
     JOIN cargos cg
    USING (cargo id);
   -- Grava hora q terminou de inserir os dados
   v data fim := SYSTIMESTAMP;
   SALVA TEMPO('DML - INSERT - DIRETO 2',7004,(v_data_fim -
v data inicio));
   COMMIT;
END;
END LOOP;
END;
```

ANEXO 10

```
CREATE OR REPLACE PACKAGE TCC.TESTE_COPY AS

   PROCEDURE EXECUTA_TESTE_SEM_NOCOPY;

   PROCEDURE EXECUTA_TESTE_COM_NOCOPY;

   PROCEDURE EXECUTA_TESTE_COM_PARAMETRO_IN;

END;
/
```

```
CREATE OR REPLACE PACKAGE BODY TCC.TESTE_COPY AS

   TYPE     t tab IS TABLE OF VARCHAR2(32767);
   g tab    t tab := t_tab();
   g_start  NUMBER;

   PROCEDURE TESTE SEM NOCOPY ( pio entrada    IN OUT T TAB
                              , pi cont        IN     NUMBER
                              , pi repeticoes  IN     NUMBER
                              , pio_cont_total IN OUT NUMBER
                              ) IS
      v cont NUMBER;
   BEGIN
      v cont := pio entrada.count;
      pio cont total := pio cont total + 1;
      IF pi cont < pi repeticoes THEN
         TESTE SEM NOCOPY( pio entrada => pio entrada
                         , pi cont => pi cont + 1
                         , pi repeticoes => pi repeticoes
                         , pio_cont_total => pio_cont_total
                         );
      END IF;
   END TESTE_SEM_NOCOPY;

   PROCEDURE TESTE COM NOCOPY ( pio entrada   IN OUT NOCOPY  t_tab
                              , pi cont       IN     NUMBER
                              , pi repeticoes IN     NUMBER
                              , pio_cont_total IN OUT NOCOPY NUMBER
                              ) IS
      v cont NUMBER;
   BEGIN
      v cont := pio entrada.count;
      pio cont total := pio cont total + 1;
      IF pi cont < pi repeticoes THEN
         TESTE COM NOCOPY( pio entrada => pio entrada
                         , pi cont => pi cont + 1
                         , pi repeticoes => pi repeticoes
                         , pio_cont_total => pio_cont_total
                         );
      END IF;
   END TESTE_COM_NOCOPY;

   PROCEDURE TESTE COM IN ( pio entrada   IN      t tab
                          , pi cont       IN      NUMBER
                          , pi repeticoes IN      NUMBER
                          , pio_cont_total IN OUT NOCOPY NUMBER
                           ) IS
      v cont NUMBER;
   BEGIN
      v cont := pio entrada.count;
      pio cont total := pio cont total + 1;
      IF pi cont < pi repeticoes THEN
         TESTE COM IN( pio entrada => pio entrada
                     , pi cont => pi cont + 1
                     , pi repeticoes => pi repeticoes
                     , pio_cont_total => pio_cont_total
                     );
      END IF;
   END TESTE_COM_IN;
```

```
FUNCTION STATUS BANCO  RETURN NUMBER AS
     l return  NUMBER;
   BEGIN
     SELECT ms.value
     INTO   l return
     FROM   v$mystat ms,
            v$statname sn
     WHERE  ms.statistic# = sn.statistic#
     AND    sn.name = 'session pga memory';
     RETURN l return;
   END STATUS_BANCO;

   PROCEDURE EXECUTA TESTE_SEM_NOCOPY AS
      v inicio NUMBER;
      v tempo inicio TIMESTAMP;
      v tempo fim    TIMESTAMP;
      v cont_total   NUMBER := 0;
   BEGIN
      -- inicializa collection com 1 milhão de linhas
      g tab.extend;
      g tab(1) := '12345678901234567890123456789012345678901234567890';
      g_tab.extend(999999, 1);

      v inicio := STATUS BANCO;
      v tempo inicio := SYSTIMESTAMP;
      FOR c IN 1..500 LOOP
         TESTE SEM_NOCOPY(g_tab,1,100,v_cont_total);
      END LOOP;
      v tempo fim := SYSTIMESTAMP;
      SALVA TEMPO('NOCOPY - SEM NOCOPY - USADO: '||(STATUS BANCO -
v inicio)||' total execuções: '||v_cont_total,8001,(v_tempo_fim -
v tempo inicio));
   END EXECUTA_TESTE_SEM_NOCOPY;

   PROCEDURE EXECUTA TESTE_COM_NOCOPY AS
      v inicio NUMBER;
      v tempo inicio TIMESTAMP;
      v tempo fim    TIMESTAMP;
      v cont_total   NUMBER := 0;
   BEGIN
      -- inicializa collection com 1 milhão de linhas
      g tab.extend;
      g tab(1) := '12345678901234567890123456789012345678901234567890';
      g_tab.extend(999999, 1);

      v inicio := STATUS BANCO;
      v tempo inicio := SYSTIMESTAMP;
      FOR c IN 1..500 LOOP
         TESTE COM_NOCOPY(g_tab,1,100,v_cont_total);
      END LOOP;
      v tempo fim := SYSTIMESTAMP;
      SALVA TEMPO('NOCOPY - COM NOCOPY - USADO: '||(STATUS BANCO -
v inicio)||' total execuções: '||v_cont_total,8002,(v_tempo_fim -
v tempo inicio));
   END EXECUTA_TESTE_COM_NOCOPY;
```

```
PROCEDURE EXECUTA TESTE_COM_PARAMETRO_IN AS
      v inicio NUMBER;
      v tempo inicio TIMESTAMP;
      v tempo fim    TIMESTAMP;
      v cont_total   NUMBER := 0;
   BEGIN
      -- inicializa collection com 1 milhão de linhas
      g tab.extend;
      g tab(1) := '12345678901234567890123456789012345678901234567890';
      g_tab.extend(999999, 1);

      v inicio := STATUS BANCO;
      v tempo inicio := SYSTIMESTAMP;
      FOR c IN 1..500 LOOP
         TESTE COM_IN(g_tab,1,100,v_cont_total);
      END LOOP;
      v tempo fim := SYSTIMESTAMP;
      SALVA TEMPO('NOCOPY - COM NOCOPY - IN: '||(STATUS BANCO -
v inicio)||' total execuções: '||v_cont_total,8003,(v_tempo_fim -
v tempo inicio));
   END EXECUTA_TESTE_COM_PARAMETRO_IN;

END;
/

Scripts dos testes:

conn tcc/tcc

exec teste_copy.EXECUTA_TESTE_SEM_NOCOPY;

conn tcc/tcc

exec teste_copy.EXECUTA_TESTE_COM_NOCOPY;

conn tcc/tcc

exec teste_copy.EXECUTA_TESTE_COM_PARAMETRO_IN;
```