UNIVERSIDADE FEEVALE

EDER MARLON MACHADO DOS SANTOS

ESTUDO COMPARATIVO ENTRE SISTEMAS GERENCIADORES DE BANCO DE DADOS EM MEMÓRIA

Novo Hamburgo
2019

EDER MARLON MACHADO DOS SANTOS

ESTUDO COMPARATIVO ENTRE SISTEMAS GERENCIADORES DE BANCO DE DADOS EM MEMÓRIA

Trabalho de Conclusão de Curso, apresentado como requisito parcial à obtenção do grau de Bacharel em Ciência da Computação pela Universidade Feevale

Orientador: Ms. Edvar Bergmann Araújo

Novo Hamburgo
2019

AGRADECIMENTOS

Gostaria de agradecer a todos os que, de alguma maneira, contribuíram para a realização desse trabalho de conclusão, em especial, aos meus familiares e amigos.

RESUMO

Desde o advento da Internet e o nascimento da era da informação no início da década de 90, o volume de dados armazenados passou a crescer exponencialmente. Mais dados foram produzidos nos últimos quatro anos do que em toda a história da humanidade. Estima-se que em 2020 este montante terá atingido dez vezes o tamanho que possuía em 2013, indo dos 4,4 trilhões de gigabytes para 44 trilhões, mais do que dobrando a cada ano, de acordo com o que estimou a *Industry Developments and Models* – IDC, em abril de 2014, em seu sétimo relatório do universo digital. Evidentemente, este cenário apresenta novos desafios à tecnologia da informação. Principalmente no que tange ao armazenamento, gerenciamento e processamento de tamanho volume de dados. Além disso, é também cada vez maior a necessidade em se obter a informação no menor tempo possível, o que para muitos sistemas representa obtê-la em tempo real. Os sistemas convencionais de gerenciamento de bancos de dados em disco não têm suprido algumas destas imposições e os bancos de dados em memória (*In-memory databases* – IMDBs) surgem como uma boa alternativa, especialmente, no que diz respeito à performance. O objetivo deste trabalho foi o de realizar a análise comparativa de algumas das principais características de duas plataformas de banco de dados relacionais em memória: um Oracle utilizando a funcionalidade *In-Memory* e um Oracle TimesTen. Foram analisadas algumas das estratégias empregadas em ambas as soluções no que diz respeito à indexação, durabilidade, armazenamento e *tuning* de instruções SQL. Além disso, foram realizados inúmeros experimentos para aferir a performance dos dois bancos de dados. O resultado desta pesquisa gerou informações que podem ser relevantes para apoiar a tomada de decisão ao se optar entre uma destas arquiteturas.

Palavras-chave: *In-memory database*. Banco de dados em memória. Sistemas Gerenciadores de Banco de Dados. Computação em memória. Desempenho.

ABSTRACT

Since the advent of the Internet and the birth of the information age in the early 90's, the volume of stored data has grown exponentially. More data has been produced in the last four years than in all of humankind's history and it is estimated that by 2020 this amount will have reached ten times the size it had in 2013. Going from 4.4 trillion gigabytes to 44 trillion, which doubles every year, according to Industry Development and Models - IDC, in April 2014, in its seventh report on the digital universe. Of course, this scenario presents new challenges to information technology. Particularly, regarding to the storage, management and processing of all this amount of data. Moreover, it is also increasingly necessary to obtain the information in the shortest possible time, which for many systems represents obtaining it in real time. Conventional disk database management systems have not met some of these constraints, and In-Memory Databases (IMDBs) appear as a good alternative, especially in terms of performance. The purpose of this work was to perform the comparative analysis of some of the main characteristics of two relational database platforms: an Oracle using In-Memory functionality and an Oracle TimesTen. It was analyzed some of the strategies applied in both architectures regarding to indexing, durability, storage, and tuning of SQL statements. In addition, numerous experiments were performed to measure the performance of the two databases. The results of this research generated relevant information to support decision making by choosing between one of these architectures.

Keywords: In-memory database. Database Management Systems. In-memory computing. Performance.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

ACID	Atomicidade, Consistência, Isolamento, Durabilidade
AMM	*Automatic Memory Management*
API	*Application Programming Interface*
CPU	*Central Processing Unit*
CU	*Compression Units*
DBA	*DataBase Administrator*
DIMM	*Dual Inline Memory Module*
DML	*Data Manipulation Language*
DRAM	*Dynamic Random-Access Memory*
E/S	Entrada e Saída
ER	Entidade Relacionamento
IM	*In-Memory*
IMCU	*In-Memory Compression Unit*
IMDB	*In-Memory DataBase*
IMEU	*In-Memory Expression Unit*
MMDB	*Main Memory DataBase*
NVDIMM	*DIMM Non-Volatile*
NVRAM	*Non-volatile Random Access Memory*
OLAP	*Online Analytical Processing*
OLTP	*Online Transaction Processing*
RAM	*Random-Access Memory*
RDBMS	*Relational Database Management System*
SGA	*System Global Area*
SGDB	Sistema de Gerenciamento de Banco de Dados
SMU	*Snapshot Metadata Unit*
SQL	*Structured Query Language*
SSD	*Solid-State Drive*
TCP/IP	*Transmission Control Protocol/Internet Protocol*
TI	Tecnologia da Informação

SUMÁRIO

1. INTRODUÇÃO

Diante do exponencial crescimento no volume de dados gerados atualmente pelo uso dos mais diversos dispositivos e aplicativos, os sistemas convencionais de gerenciamento de bancos de dados em disco têm apresentado deficiências no que diz respeito à suprir algumas das necessidades advindas deste cenário. Melhorar a eficiência e o acesso a grandes volumes de dados, facilitar a manipulação de grandes conjuntos de dados e flexibilizar a atual estrutura baseada em modelo de dados e metadados utilizada pelos bancos relacionais, são alguns dos desafios que se tornam evidentes com o crescimento no volume dados (TIWARI, 2011, apud PINTO, 2016). A complexidade envolvida no armazenamento, gerenciamento e processamento aumenta à medida que mais e mais dados são gerados e, além disso, a exigência por desempenho é cada vez mais pungente.

Para Nascimento (2017), este grande volume de dados, estruturados e não estruturados, que é gerado a cada segundo é o que tem se chamado de *big data*. De acordo com O'Reilly (tradução nossa, 2012, pág. 3):

> Big data é o dado que excede a capacidade de processamento dos sistemas de banco de dados convencionais. O dado é muito grande, se move muito rápido, ou não se encaixa nas restrições de suas arquiteturas de banco de dados. Para extrair valor destes dados, você precisa escolher maneiras alternativas para processá-lo.

Uma das alternativas ao modelo tradicional, são os bancos de dados em memória (*in-memory databases*, IMDBs), os quais visam oferecer maior velocidade, flexibilidade, volume e disponibilidade, quando comparados aos sistemas gerenciadores de bancos de dados em disco (SGDBs).

Em um banco de dados em memória os dados são totalmente carregados na memória principal (GARCIA-MOLINA; SALEM, 1992). Como um dos gargalos dos SGDBs convencionais é justamente a escrita e a leitura em disco, espera-se obter um ganho acentuado em termos de desempenho em um IMDB, já que ele acessa os dados diretamente na memória principal.

Conforme o Yu (2013, pág. 4):

> As velocidades de memória são de 100.000 a um milhão de vezes mais rápidas do que os discos rígidos mecânicos em termos de tempos de acesso. O custo da memória é cerca de 100 vezes maior do que os discos rígidos mecânicos. A certeza (1.000 a 10.000 vezes) é um ganho de desempenho considerável ao mudar para soluções baseadas *In-Memory*.

Yu (2013) destaca ainda alguns fatores que devem ser levados em consideração antes de se escolher uma plataforma específica ao se optar por um IMDB. São eles:

- Conformidade com ACID: nem todo IMDB está em conformidade com a ACID);
- Volume de dados e escalabilidade: algumas plataformas são mais escaláveis que outras;
- Compatibilidade com SQL: alguns IMDBs oferecem suporte apenas a um conjunto básico de SQL primitivos;
- Compressão: alguns IMDBs suportam compressão às custas do tempo de processamento da CPU;
- Custo: existem tanto plataformas abertas quanto comerciais, mas a escolha passará pelos requisitos mencionados acima.

Horn (2016) e Vargas (2017) já realizaram trabalhos práticos que tiveram por objetivo comparar o desempenho de plataformas de bancos de dados em disco e em memória, e em ambos os casos os IMDBs demostraram melhor performance que os SGDBs tradicionais. Entretanto, aspectos como o comportamento do IMDB no que concerne à recuperação de desastres, a análise da relação entre o tamanho do banco de dados em disco e do quanto ele ocupa quando é carregado em memória, e a verificação de como ocorre a utilização de índices nestas diferentes arquiteturas, são aspectos que não foram abordados de maneira mais aprofundada por não fazerem parte do escopo daqueles trabalhos. No intuito de gerar informação para futuras pesquisas de natureza aplicada e de apoiar a tomada de decisão no momento da escolha entre um destes modelos, este estudo buscou aprofundar a análise destes pontos.

Por fim, o objetivo principal deste trabalho foi o de efetuar um estudo comparativo entre um banco de dados Oracle utilizando a funcionalidade *In-Memory* e um Oracle TimesTen. O *Oracle Database In-Memory* é uma solução híbrida que permite que sejam definidos quais objetos serão mantidos em memória e preserve os demais em disco, exatamente como ocorre em um SGDB convencional. Em contrapartida, o Oracle TimesTen é um banco de dados totalmente em memória.

Os capítulos 2, 3 e 4, discorrem à respeito do referencial teórico, abordando o conceito de banco de dados em memória e trazendo as especificações do Oracle

TimesTen 11g e do Oracle 18c, com *Database In-Memory*. O capítulo 5, aborda as questões relacionadas a metodologia e ao ambiente de testes. Já no capítulo 6, com a comparação entre TimesTen e o *Database In-Memory*, iniciam-se as contribuições desta monografia, as quais seguem pelo capítulo 7, no qual são apresentados os experimentos de performance. Por fim, o capítulo 8 traz as conclusões extraídas deste trabalho.

2. BANCO DE DADOS EM MEMÓRIA

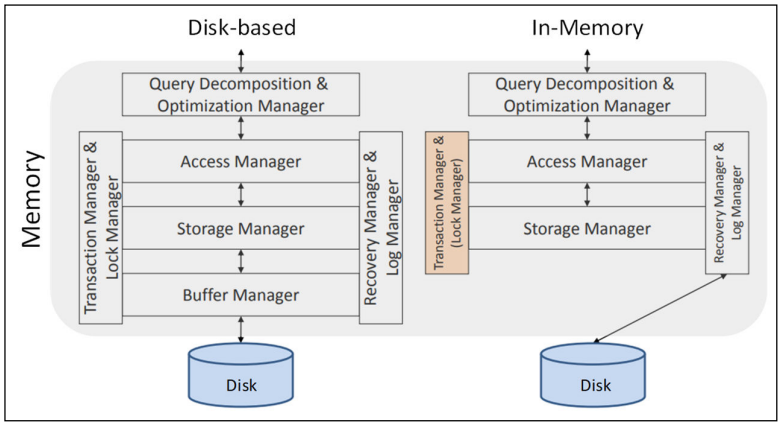
Segundo Garcia-Molina e Salem (apud Gupta; Verma; Verma, 2013), um *In-Memory Database* (IMDB), ou *Main Memory Database* (MMDB), é um tipo de sistema gerenciador de banco de dados (SGDBs) que armazena os dados inteiramente na memória principal ao invés de mantê-los no disco. Evidentemente, a utilização de discos passará pela forma como o IMDB irá garantir a durabilidade, mas o certo é que, ainda que permaneçam ocorrendo, as operações de entrada e saída (E/S) em disco sofrem uma dramática redução nos bancos de dados em memória. Isso também se deve ao fato de que em um IMDB os discos desempenham essencialmente a função de persistência e recuperação em caso de falhas.

Visto que os dados são armazenados em memória principal, a velocidade de escrita e leitura melhora drasticamente, considerando-se que são eliminadas as operações de entrada e saída em disco. Além disso, como os dados já estão em memória, não há a necessidade de implementação de lógicas complexas para *caching*, o que acaba também com a sobrecarga em função da utilização do *cache* (Gupta; Verma; Verma, 2013).

A Figura 1 apresenta uma comparação simplificada entre as arquiteturas de um SGDB tradicional e a de um IMDB. Nela podem ser observadas duas caraterísticas marcantes dos bancos de dados em memória:

- A inexistência do gerenciador de buffer;
- A mudança no acesso ao disco físico.

Figura 1 – Arquitetura básica de SGDB baseado em disco versus a de um IMDB

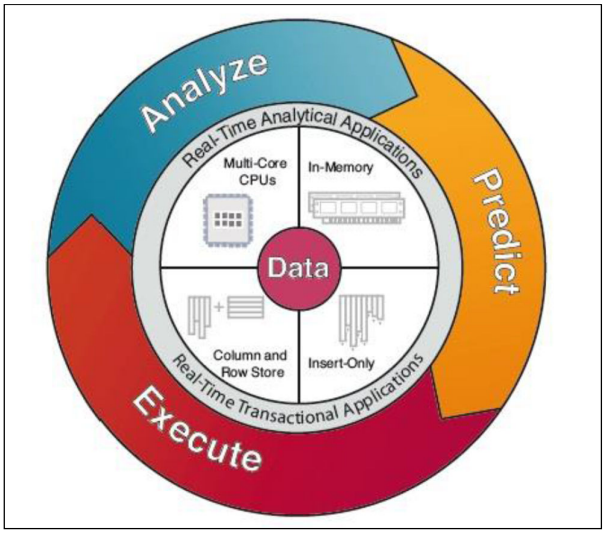


Fonte: Adaptado de Störl (2017)

De acordo com a Oracle (2018a), apesar de muitos afirmarem que os bancos de dados em memória utilizam uma arquitetura revolucionária, nenhuma das tecnologias empregadas por IMDBs é de fato nova. A inovação está, na verdade, na combinação, em um único produto, de tecnologias que até bem pouco tempo estavam disponíveis apenas em produtos separados.

Para sustentar esta afirmação, a Oracle (2018a) aponta que a manutenção em memória dos dados a serem lidos ou manipulados é algo que já é feito pelos SGDBs convencionais. Além disso, apesar dos bancos tradicionais armazenarem os dados em linhas, favorecendo o *Online Transaction Processing* (OLTP), já existem há muitos anos soluções colunares que buscam beneficiar o *Online Analytical Processing* (OLAP). Finalmente, a Oracle (2018a) menciona o fato de que alguns fornecedores oferecem a criação de procedimentos a serem armazenados no próprio banco de dados, as *stored procedures*. Certamente, isto também não representa uma inovação, já que a própria Oracle introduziu as *stored procedures* em seu banco de dados há mais de 20 anos. A Figura 2 ilustra alguns dos princípios que tornam possível a união de bancos de dados analíticos e transacionais em um só produto.

Figura 2 – Círculo de desempenho empresarial com *in-memory*



Fonte: Plattner; Zeier, 2011 (Apud Silva, 2013).

Conforme Störl (2017), as primeiras abordagens sobre bancos de dados em memória ocorreram ainda no início dos anos 80 e se mantiveram como um tópico dentro da comunidade de pesquisa sobre banco de dados. Com o passar dos anos a

memória principal se tornou mais barata e com maior capacidade, os processadores passaram a ser multicores, houve melhorias em software, como o armazenamento orientado a colunas e a melhora na compressão de dados. Isto apenas para listar alguns dos avanços tecnológicos que ocorreram nas últimas décadas envolvendo bancos de dados. Juntas, estas mudanças estão fazendo com que sistemas OLTP e OLAP, que historicamente foram separados por causa da contenção de recursos e de limitações de hardware, possam passar a conviver em um único SGDB sem que uma delas sofra degradação em relação à outra.

2.1. IMDB X SGDB BASEADO EM DISCO

Para a Oracle (2018b), por gerenciar todos os dados na memória e por ser otimizado para esse ambiente, um IMDB pode operar com muito mais eficiência e oferecer melhorias drásticas no desempenho. Como o IMDB assume que os dados residem na memória principal, pode usar rotas mais diretas, reduzindo o tamanho do código e simplificando o algoritmo e a estrutura. A Oracle (2018b) cita ainda algumas diferenças cruciais entre IMDBs e SGDBs baseados em disco. São elas:

1) Otimização de consultas: Como a E/S em disco é muito mais lenta do que o acesso à memória, as decisões de otimização do SGBDs tradicionais baseiam-se na suposição de que os dados residem principalmente no disco. Eles precisam reduzir ao máximo a probabilidade de acesso ao disco físico, por isso são otimizados para reduzir o gargalo do desempenho, o que faz com que o otimizador nem sempre produza o plano ideal para os dados que residem na memória principal. Por outro lado, um banco de dados em memória, otimiza suas consultas sob hipóteses mais simples, pois sabe que os dados residem na memória principal, o que faz com que suas estimativas de custo possam ser mais simples e consistentes;
2) Índices: diferentemente de SGDBs convencionais, o IMDB reduz muito o tamanho dos índices por não precisar armazenar valores chave na estrutura do índice, o que também resulta em uma implementação mais simples, pois evita a necessidade de gerenciar chaves de tamanho variável nos nós de índice;
3) Processamento de consultas: Com todo o dado em memória, o IMDB pode tirar vantagem disso. Melhorando, por exemplo, uma busca por índice, posto que

as entradas apontam diretamente para o endereço de memória onde os dados residem, fazendo com que a varredura aleatória por um índice seja tão barata quanto uma varredura sequencial;

4) Gerenciamento do *buffer pool*: Como os dados já estão todos na memória principal, não há mais a necessidade de *buffer pool*. A manutenção e o gerenciamento desta estrutura de memória, em conjunto com as cópias adicionais de dados, aumentam significativamente a carga original, o que não ocorre em um IMDB.

2.2. SUPORTE A ACID

Atomicidade, consistência, isolamento e durabilidade, também conhecidos pelo acrônimo ACID, são atributos que devem ser garantidos em transações de bancos de dados relacionais, a fim de assegurar a integridade dos dados. Em outras palavras, ACID são propriedades de conformidade que pressupõem que as transações de banco de dados são executadas de forma confiável (YU, 2013).

No que diz respeito ao presente trabalho, a propriedade que mais interessa é a durabilidade. Primeiro, porque ela é o atributo que costuma variar nas diferentes implementações de IMDB. E depois, pelo simples fato de que por si só ela gera um grande desafio a qualquer IMDB, tendo em mente que a memória principal é volátil e não mantém as informações em caso falta de energia. Segundo a Oracle (2018a, pág. 1, tradução nossa):

> “Dados em memória são voláteis. Para garantir, não apenas performance, mas também persistência, um armazenamento não-volátil (baseado em disco) precisa estar disponível como uma camada secundária de armazenamento.”

Para ITL (2010), durabilidade, também conhecida como persistência, implica que uma vez que uma transação é finalizada com sucesso, as mudanças feitas por esta transação persistem no banco de dados, mesmo se o sistema falhar.

As principais técnicas empregadas por IMDBs para garantir a durabilidade dos dados segundo Gorine (2004, apud Gupta; Verma; Verma, 2013) são:

1) Memória principal persistente: as últimas tecnologias na área de memória, como NVRAM (Non-Volatile RAM) ou memória principal com baterias, habilitam um cenário onde os dados na memória principal não são perdidos;

2) *Transaction logging:* com o log de transações, cada transação será registrada em um armazenamento persistente, o que permitirá que o mecanismo de *roll-forward* restaure o banco de dados em caso de falha de energia. Para evitar que a criação do log de transações se torne um gargalo no desempenho geral, os logs são escritos primeiro na memória estável (mais rápida do que a gravação em disco) e depois no disco, em processos assíncronos. Assim, as principais operações do banco de dados não vão esperar pela *logging* para concluírem;
3) Implementação de alta disponibilidade: alta disponibilidade significa replicar os dados em tempo real para outros nós disponíveis. Isso permitirá criar cópias dos dados quase que em tempo real, reduzindo a probabilidade de falha completa de dados. Caso, qualquer nó caia, outro nó assume a requisição e a processa.

O mecanismo mais utilizado é a regravação em um armazenamento persistente, o que acaba tendo de lidar com discos físicos. Para contornar este problema, a maior parte das soluções utiliza o que é chamado de gravação no cache ou na memória “preguiçosa” ou “imprecisa”. Isto é, basicamente, se poderá controlar a frequência com que o sistema irá gravar em disco o *buffer* de log gerado a partir das transações realizadas completamente em memória. A Microsoft (2016), por exemplo, permite duas configurações:

- Totalmente duráveis: são síncronas e relatam uma confirmação como bem-sucedida, devolvendo o controle ao cliente, somente após a gravação dos registros de log da transação em disco;
- Duráveis atrasadas: são assíncronas e relatam uma confirmação como bem-sucedida antes que os registros de log da transação sejam gravados no disco. As transações duráveis atrasadas tornam-se duráveis quando as entradas de log de transações são liberadas para o disco.

O primeiro caso é indicado sempre que o sistema não puder tolerar perda de dados, ou que não haja gargalo devido à latência de gravação do log de transações. Já a gravação atrasada pode ser a melhor alternativa quando o sistema puder tolerar alguma perda de dados, quando houver um gargalo em gravações de log de transações, ou quando suas cargas de trabalho tiverem uma taxa alta de contenção (MICROSOFT, 2016).

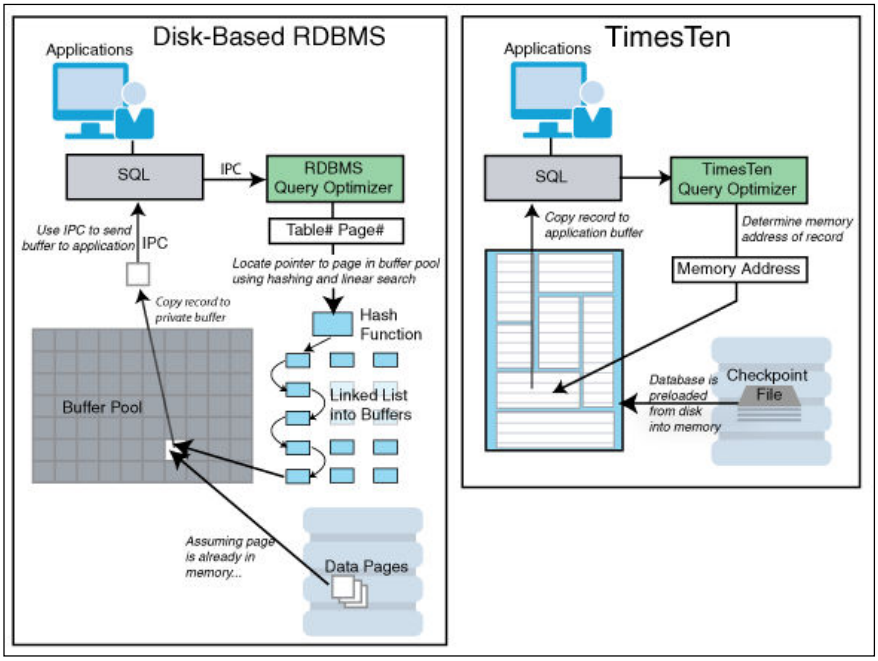
2.3. CONSIDERAÇÕES FINAIS

A bibliografia discute amplamente as características dos bancos de dados em memória e as diferenças entre IMDBs e os bancos de dados tradicionais baseados em disco. Os trabalhos de Horn (2016) e Vargas (2017) também abordaram tais pontos. Neste trabalho, o propósito foi apresentar tais temas de forma sintetizada e priorizar detalhes das implementações da Oracle baseadas em memória. Tais implementações serão apresentadas nos capítulos 3 e 4 e comparadas a partir do capítulo 6.

3. ORACLE TIMESTEN 11G

Conforme Oracle (2018b), o *TimesTen In-Memory Database* (TimesTen) é um banco de dados relacional otimizado para memória e para rápida resposta e taxa de transferência (Figura 3). O banco de dados reside inteiramente na memória principal em tempo de execução, o que faz dele um IMDB. É implantado normalmente na camada de aplicativo e, além disso, é persistente e recuperável, e permite o acesso por meio de interfaces SQL padrão.

Figura 3 – SGDB baseado em disco versus Oracle TimesTen clássico



Fonte: Oracle (2018b)

O TimesTen também permite a replicação entre servidores a fim de garantir a alta disponibilidade e o compartilhamento de carga. Suporta configurações ativo/passivo, ativo/ativo, transmissões síncrona e assíncrona, detecção e resolução de conflitos e *failover*, recuperação e ressincronização automáticas após a restauração de um servidor com falha (ORACLE, 2018b).

Este IMDB ainda pode ser utilizado como um cache de outro banco de dados Oracle na camada de aplicação, armazenando subconjuntos atualizáveis em tempo

real de um banco de dados Oracle e sincronizando automaticamente os dados entre o cache e o banco de dados.

3.1. MODOS DO ORACLE TIMESTEN

De acordo com a Oracle (2018b), o TimesTen suporta três modos de implementação:

- Clássico (*TimesTen Classic*): É o modo mais simples e tradicional do TimesTen e a alta disponibilidade é fornecida por meio da replicação transacional;
- Grid (*TimesTen Scaleout*): Neste modo há um *grid* de *hosts* interconectados que executam instâncias do *TimesTen Scaleout* e trabalham em conjunto para fornecer acesso rápido, tolerância a falhas e alta disponibilidade. Um grid pode conter um ou mais bancos de dados e cada banco de dados é distribuído em todas as instâncias do grid;
- Cache (*TimesTen Cache*): Esta opção permite o armazenamento em cache de subconjuntos críticos para o desempenho de um banco de dados Oracle. Tabelas de cache podem ser somente leitura ou atualizáveis e a sincronização de dados entre o cache e o banco de dados Oracle é realizada automaticamente. O modo cache oferece de maneira completa a generalidade e as funcionalidades de um banco de dados relacional, aliadas à manutenção transparente da consistência do cache e ao alto desempenho de um IMDB.

As três soluções oferecem o mesmo alto desempenho, pois todas usam o TimesTen como seu IMDB e realizam o gerenciamento dos dados na memória, otimizam as estruturas de dados e acessam algoritmos adequadamente, as operações do banco de dados são executadas com eficiência máxima, obtendo ganhos significativos em capacidade de resposta e taxa de transferência, mesmo em comparação a um RDBMS totalmente em cache (ORACLE, 2018b).

3.2. ARQUITETURA

A arquitetura para o banco de dados TimesTen é praticamente a mesma para o modo clássico e para o modo cache. A única diferença é que o modo cache possui

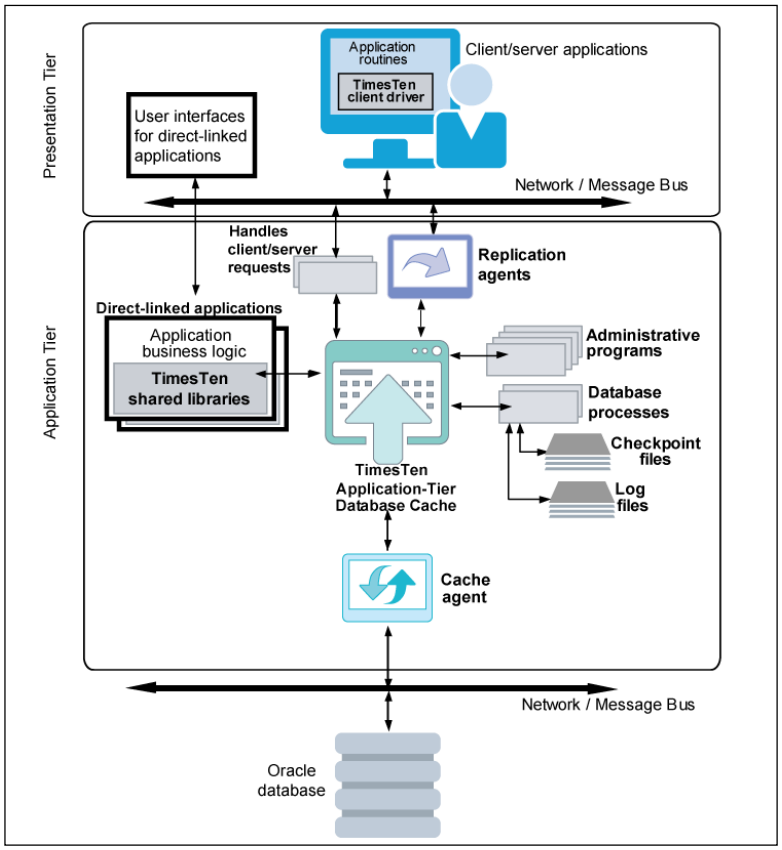
todos os componentes de um TimesTen *Classic*, mais um agente de cache e um outro banco de dados Oracle, o qual está sendo *cacheado*. De acordo com a Oracle (2018b), os principais componentes do TimesTen são:

- Bibliotecas compartilhadas: As rotinas que implementam a funcionalidade do TimesTen são incorporadas em um conjunto de bibliotecas compartilhadas. Estas bibliotecas são vinculadas aos aplicativos, que por sua vez às executam como parte de seu processo. Eles também podem usar uma conexão cliente/servidor, embora para a maioria dos casos o desempenho seja melhor com um aplicativo diretamente vinculado;
- Estruturas de dados residentes na memória: O TimesTen é mantido em regiões de memória compartilhada fornecidas pelo sistema operacional e contém todos os dados do usuário, índices, catálogos do sistema, *buffers* de log e espaço temporário;
- Processos de banco de dados: Para executar operações, o TimesTen atribui um processo de segundo plano separado a cada banco de dados;
- Programas administrativos: Os programas utilitários são explicitamente invocados por usuários, scripts ou aplicativos para executar serviços, como SQL interativo, cópia em massa, backup e restauração, migração de banco de dados e monitoramento do sistema;
- Arquivos de log de ponto de verificação e transação: Os arquivos do ponto de verificação contêm uma imagem do banco de dados no armazenamento persistente no disco;
- Dados armazenados em cache: Quando o TimesTen é usado para armazenar em cache partes de um banco de dados Oracle, um grupo de cache é criado para conter estes dados;
- Replicação: permite que você atinja a disponibilidade quase contínua ou a distribuição da carga de trabalho enviando atualizações entre dois ou mais *hosts*;
- Conexão: Os aplicativos podem se conectar ao TimesTen de uma das seguintes maneiras:
    - Conexão direta
    - Conexão cliente / servidor
    - Conexão do gerenciador de driver

3.3. CACHING

Quando há necessidade que os dados sejam capturados ou processados em tempo real, o TimesTen pode ser implantado em conjunto com um SGDB baseado em disco e funcionar como um cache deste banco de dados, com a integração entre ambos sendo customizada. Os aplicativos podem se conectar ao TimesTen e ao banco de dados de *back-end* e mover os dados por meio de solicitações regulares de API. Essa é a abordagem mais flexível, mas é também a menos transparente para os aplicativos e pode exigir uma complexa programação. A Figura 4 apresenta uma visão geral da arquitetura de um banco de dados Oracle TimesTen Cache.

Figura 4 – Arquitetura do TimesTen Cache



Fonte: Oracle (2018b)

Segundo a Oracle (2018b), uma solução de armazenamento em cache pré-integrada é por meio da opção de cache do banco de dados da camada do aplicativo

do Oracle TimesTen (TimesTen Cache). Essa opção permite que um aplicativo armazene em cache subconjuntos de linhas e colunas das tabelas do banco de dados Oracle no TimesTen. Os dados de *cache* podem ser lidos e atualizados e o TimesTen *Cache* sincroniza os dados entre os dois bancos automaticamente.

Quando TimesTen Cache é usado, um grupo de *cache* é criado para conter os dados armazenados em *cache*. Um grupo de *cache* é uma coleção de uma ou mais tabelas organizadas em uma hierarquia lógica usando os relacionamentos de chave primária e chave estrangeira. Cada tabela em um grupo de *cache* está relacionada a uma tabela do banco de dados Oracle. Uma tabela de *cache* pode conter todas as linhas e colunas ou um subconjunto das linhas e colunas na tabela de banco de dados Oracle relacionada.

Grupos de cache suportam os seguintes recursos:

- Os aplicativos podem ler e gravar em grupos de cache;
- Eles podem ser atualizados a partir do banco de dados Oracle, de forma automática ou manual;
- Atualizações em grupos de cache podem ser propagadas para as tabelas do banco de dados Oracle de forma automática ou manual;
- Alterações nas tabelas do banco de dados Oracle ou no grupo de cache podem ser rastreadas automaticamente.

Quando as linhas em um grupo de cache são atualizadas pelos aplicativos, as linhas correspondentes nas tabelas do banco de dados Oracle podem ser atualizadas de forma síncrona, como parte da mesma transação, ou assíncrona. A configuração assíncrona produz um rendimento significativamente mais alto e tempos de resposta mais rápidos.

As alterações originadas nas tabelas do banco de dados Oracle são atualizadas no cache pelo agente de cache.

3.4. DISPONIBILIDADE E INTEGRIDADE DE DADOS

Conforme a Oracle (2018b), o TimesTen assegura a disponibilidade, a durabilidade e a integridade dos dados através dos seguintes mecanismos:

- Log de transações;

- Ponto de verificação;
- Replicação.

3.4.1. Log de transações

Segundo a Oracle (2018b), o log de transações é gerado a partir de operações DML e é periodicamente armazenado em arquivos no disco. Ele é usado com os seguintes propósitos:

- Refazer transações em caso de falhas no sistema;
- Desfazer transações que sofrem *rollback;*
- Replicar alterações em outros bancos de dados do TimesTen;
- Quando usando o *TimesTen Cache*, replicar as alterações em um banco de dados Oracle;
- Para o *TimesTen Classic*, habilitar aplicações para monitorar as mudanças nas tabelas.

O TimesTen grava o conteúdo do *buffer* de log em memória nos arquivos de log de transação no disco em cada *commit* durável, em cada *checkpoint* e em outras ocasiões definidas pela implementação. Aplicações que não toleram a perda de transações confirmadas, devem solicitar um *commit* durável para cada transação que não seja somente leitura. Por outro lado, as aplicações que aceitam a perda de algumas transações que tenham sido confirmadas recentemente, podem melhorar significativamente seu desempenho, fazendo o *commit* de forma "*não-durável*" para algumas ou para todas as transações (ORACLE, 2018b).

3.4.2. Ponto de verificação

A Oracle (2018b, cap. 6, pág. 2, tradução nossa) descreve que:

> Pontos de verificação são usados para manter um *snapshot* do banco de dados. Se ocorrer uma falha no sistema, o TimesTen pode usar um arquivo de *checkpoint* com arquivos de log de transação para restaurar um banco de dados ao seu último estado transacional consistente. Apenas os dados que foram alterados desde a última operação de ponto de verificação são gravados no arquivo de *checkpoint*. A operação de *checkpoint* varre o banco de dados em busca de blocos que foram alterados desde o último ponto de verificação. Em seguida, atualiza o arquivo de *checkpoint* com as alterações e remove os arquivos de log de transações que não são mais necessários.

O TimesTen fornece dois tipos de pontos de verificação, conforme estabelece a Oracle (2018b):

- Sem bloqueios: o TimesTen inicia *checkpoints* sem bloqueio automaticamente e em segundo plano. Eles também são conhecidos como pontos de verificação *imprecisos*. A frequência desses *checkpoints* pode ser ajustada pela aplicação. Estes checkpoints não exigem nenhum bloqueio no banco de dados, portanto, várias aplicações podem, de maneira assíncrona, aplicar *commit* ou *rollback* em transações no mesmo banco de dados enquanto a operação de ponto de verificação está em andamento;
- Pontos de verificação com bloqueio: Ao usar o *TimesTen Classic*, uma aplicação pode iniciar um *checkpoint* com bloqueio a fim de construir um ponto de verificação consistente na transação. Enquanto esta operação estiver em andamento, quaisquer novas transações serão colocadas em uma fila atrás da transação de *checkpoint*. Se alguma transação for de longa duração, isso poderá fazer com que muitas outras transações sejam suspensas.

3.4.3. Replicação

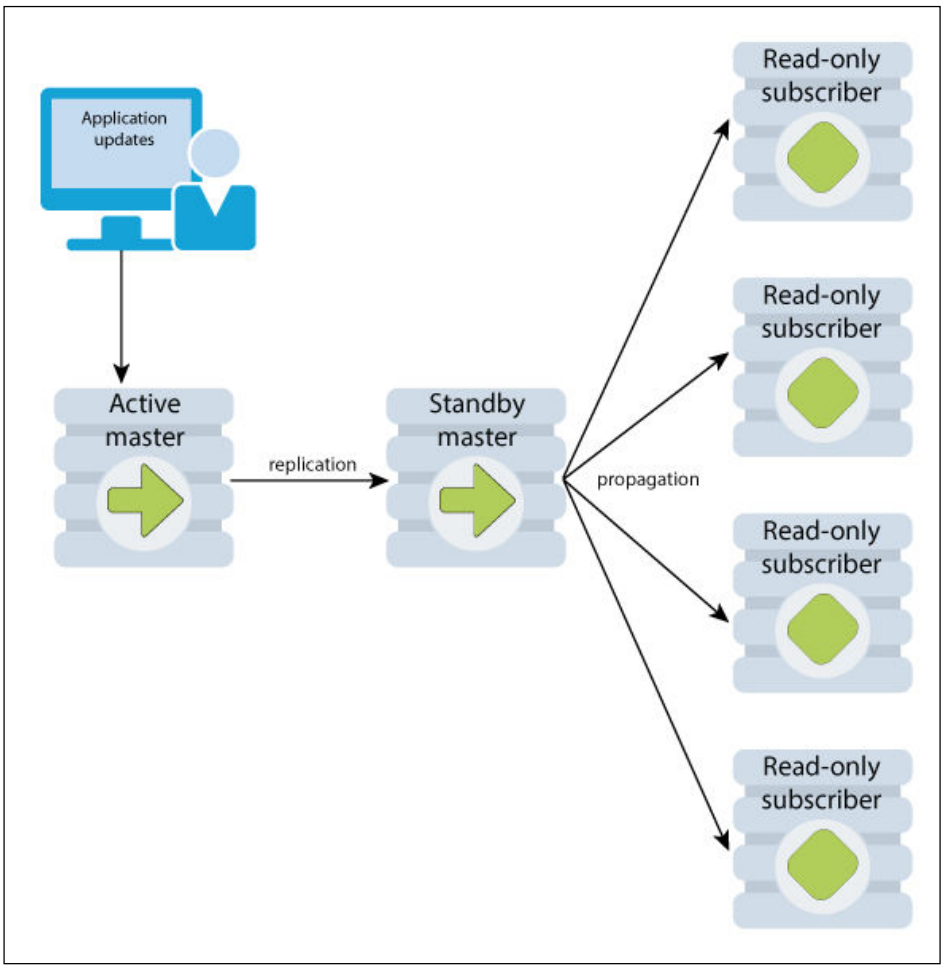
Embora a replicação tenha como principal motivação a garantia de alta disponibilidade com o mínimo de impacto na performance, ela também desempenha um papel importante na recuperação do banco em caso de falhas e na distribuição da carga entre múltiplos bancos de dados. “Replicação é o processo de copiar dados de um banco de dados mestre para um banco de dados assinante.” (ORACLE, 2018b, cap. 6, pág. 3, tradução nossa).

No TimesTen, a replicação para cada banco de dados principal e assinante é controlada por agentes de replicação que se comunicam através de soquetes de TCP/IP. O agente de replicação no banco de dados mestre lê os registros do log de transações do banco mestre e encaminha as alterações para o agente de replicação no banco de dados do assinante, que, por sua vez, aplica as atualizações no seu banco de dados. Se o agente do assinante não estiver em execução quando as atualizações forem encaminhadas, o mestre reterá as atualizações em seu log de transações até que elas possam ser aplicadas no assinante (ORACLE, 2018b).

Existem diferentes arquiteturas de replicação no TimesTen e a escolha de uma delas envolve aspectos não apenas relacionados à aplicação, mas também ao quanto se está disposto a investir.

A configuração recomendada pela Oracle para alta disponibilidade é a de “Par de espera ativa” (Figura 5), a qual inclui um banco de dados ativo, um banco de dados em espera, bancos de dados assinantes somente leitura (opcionais) e as tabelas e grupos de cache que compõem os bancos de dados.

Figura 5 – Par de espera ativa



Fonte: Oracle (2018b)

Este capítulo apresentou o TimesTen de acordo com a documentação da Oracle. As ponderações deste produto serão realizadas no capítulo 5, em conjunto com as do Oracle 18c, o qual será apresentado no capítulo 4. Tal comparação se configura como uma contribuição.

4. ORACLE 18C – DATABASE IN-MEMORY

De acordo com a Oracle (2018e), o Oracle *Database In-Memory* é um conjunto de recursos, introduzido no *Oracle Database 12c Release 1*, que melhora muito o desempenho do banco de dados para análises em tempo real e cargas de trabalho mistas. O *In-Memory Column Store* (*IM Column Store*, armazenamento colunar em memória) é o principal recurso do *Database In-Memory*.

O *Oracle Database In-Memory* inclui o *In-Memory Column Store*, otimizações avançadas de consulta e soluções de disponibilidade. Recursos que combinados aceleram as consultas em grande magnitude e sem sacrificar o desempenho ou a disponibilidade do OLTP (ORACLE, 2018e). Diferentemente do Oracle TimesTen, o *Oracle Database In-Memory* não é um SGDB, mas um recurso *built-in* que já está presente desde a versão 12c *Release* 1 do *Oracle Database*. O Quadro 1 apresenta as diretrizes para maximizar os benefícios do *Database In-Memory*.

Quadro 1 – Diretrizes para maximizar os benefícios do *Database In-Memory*

Regra 1	Processar dados no banco, não na aplicação	Calcular uma métrica como um total ou uma média, é mais eficiente se feito dentro do próprio banco de dados. Isso é especialmente verdade com o *Database In-Memory*, uma vez que os benefícios do processamento dentro do banco são ainda maiores.
Regra 2	Processar dados em conjuntos, não linha por linha	Esta regra aplica-se a qualquer carga de trabalho de análise: os custos de entrada e saída no banco de dados são amortizados pelo número de linhas processadas. Tendo em vista que o *Database In-Memory* pode processar bilhões de linhas por segundo, é importante dar à ele dados suficientes para processar em uma única chamada.
Regra 3	Usar estatísticas atualizadas para o otimizador	Planos diferentes podem provocar uma grande mudança na performance de uma consulta. Para garantir que se tenham os melhores planos, devem ser seguidas as práticas recomendadas pela Oracle para reunir um conjunto de estatísticas que seja representativo.
Regra 4	Quando possível, usar SQL com paralelismo	Com o *Database In-Memory*, gargalos de E/S são reduzidos e o tempo de CPU passa a dominar o perfil da execução. O paralelismo é essencial para maximizar a performance e utilizar todos os núcleos de CPU disponíveis para o processamento em memória.

Fonte: adaptado de Oracle (2015)

Para Colgan, gerente de produtos do *Oracle Database In-Memory* (Kyle, 2015, pág. 1, tradução nossa):

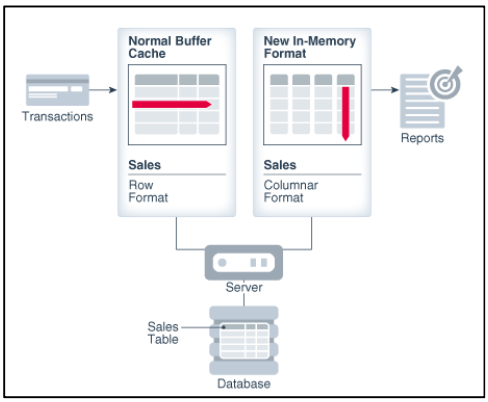
> O Oracle Database In-Memory fornece uma arquitetura de formato duplo exclusiva que permite que as tabelas do banco de dados Oracle sejam representadas no disco e na memória simultaneamente, usando um formato de linha tradicional e um novo formato de coluna na memória. O otimizador de consultas do Oracle Database encaminha automaticamente as consultas analíticas para o formato de coluna e o OLTP (processamento de transações on-line) para o formato de linha, fornecendo de maneira transparente o melhor desempenho nos dois mundos. O Oracle Database 12c mantém automaticamente a consistência transacional completa entre os formatos de linha e coluna, assim como mantém a consistência entre tabelas e índices hoje. O novo formato de coluna é um formato puro na memória e não é persistente no disco, portanto, não há custos adicionais de armazenamento ou problemas de sincronização de armazenamento.

Os princípios para se extrair o máximo de desempenho do *Database In-Memory* não são novos, mas eles são ainda mais importantes devido ao ganho que podem propiciar no acesso a dados analíticos.

4.1. FORMATO LINEAR X COLUNAR

Tradicionalmente, bancos de dados relacionais armazenam os dados em formato linear, ou colunar, mas não em ambos. Além disso, os dados são armazenados no mesmo formato tanto em memória, quanto em disco. O problema aqui é que o armazenamento em formato linear é a melhor abordagem para os sistemas transacionais ou OLTPs, enquanto o colunar beneficia sistemas analíticos ou OLAPs. Sendo assim, o ideal é que se tenha um SGDB capaz de armazenar os dados em ambos os formatos. A Figura 6 ilustra o duplo formato de armazenamento.

Figura 6 – Duplo formato de armazenamento



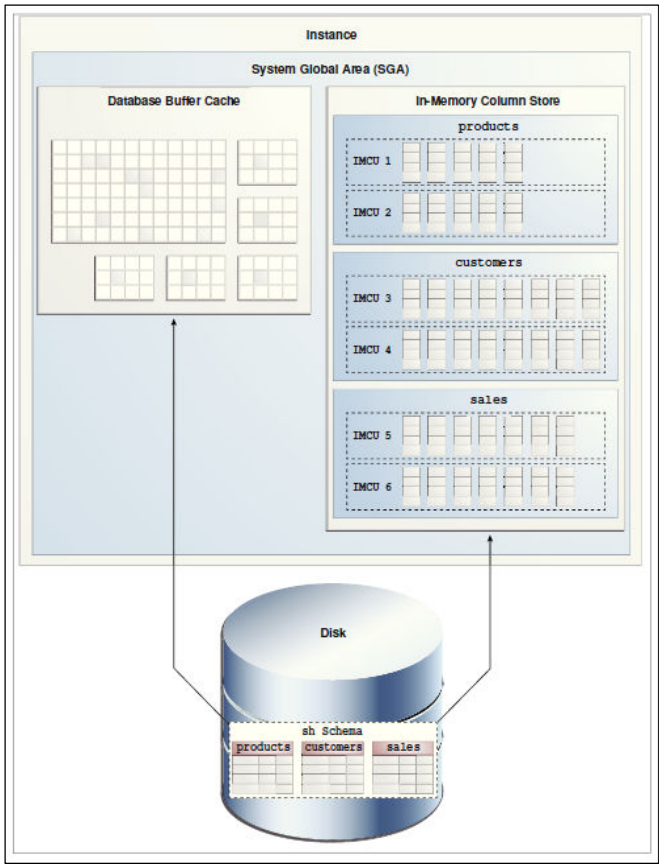
Fonte: Oracle (2018e)

"O *Oracle Database In-Memory* fornece o melhor dos dois mundos, permitindo que os dados sejam preenchidos simultaneamente em memória no formato linear (no *buffer cache*) e em um novo formato colunar: uma arquitetura de duplo formato." (ORACLE, 2018e, pág. 3, tradução nossa).

4.2. IN-MEMORY COLUMN STORE

Segundo a Oracle (2018e), o *Database In-Memory* usa o *In-Memory Column Store* (*IM Column Store*), que é um novo componente da Área Global do Sistema (*System Global Area* – SGA), também chamada de *In-Memory Area*. Os dados no *IM Column Store* não residem no formato de linha usado tradicionalmente pelo Oracle, mas sim em um formato colunar. O *IM Column Store* não substitui o *buffer cache*, mas age como um suplemento, para que os dados possam ser armazenados na memória em ambos os formatos, em linha e em coluna (Figura 7).

Figura 7 – *In-Memory Column Store*



Fonte: Oracle, 2018d

O banco de dados mantém a completa consistência transacional entre os formatos de linha e coluna, assim como mantém a consistência entre tabelas e índices. O que é realmente bom nessa abordagem é que há apenas uma cópia da tabela no armazenamento, portanto, não há custos adicionais de armazenamento nem problemas de sincronização (KYLE, 2015).

O formato colunar não afeta o formato dos dados armazenados nos arquivos de dados ou no *buffer cache*, nem afeta os dados de *undo* e de *redo log on-line*. Além disso, a arquitetura de formato duplo não duplica os requisitos de memória, pois o *buffer cache* é otimizado para ser executado com um tamanho muito menor que o tamanho do banco de dados (ORACLE, 2018e).

Para que um objeto seja carregado na *IM Column Store* ele precisa ter configurado o atributo *INMEMORY*, o qual pode ser especificado a nível de *tablespace*, tabela, partição, subpartição ou *view* materializada (ORACLE, 2018e).

Os objetos são populados na *IM Column Store* imediatamente após o banco ser aberto, de acordo com uma lista de prioridade, ou após eles serem acessados pela primeira vez. A ordem pela qual os objetos são carregados é controlada pela palavra-chave *PRIORITY*, que possui cinco diferentes níveis (Quadro 2).

Quadro 2 – Níveis de prioridade para a *IM Column Store*

PRIORIDADE	DESCRIÇÃO
CRITICAL	Objeto é populado imediatamente após o banco ser aberto
HIGH	Objeto é populado após todos os objetos CRITICAL terem sido populados, se houver espaço disponível na *IM Column Store*
MEDIUM	Objeto é populado após todos os objetos CRITICAL e HIGH terem sido populados, se houver espaço disponível na *IM Column Store*
LOW	Objeto é populado após todos os objetos CRITICAL, HIGH e MEDIUM terem sido populados, se houver espaço disponível na *IM Column Store*
NONE	Objetos são populados somente após serem acessados pela primeira vez (*Default*), se houver espaço disponível na *IM Column Store*

Fonte: adaptado de Oracle (2018e)

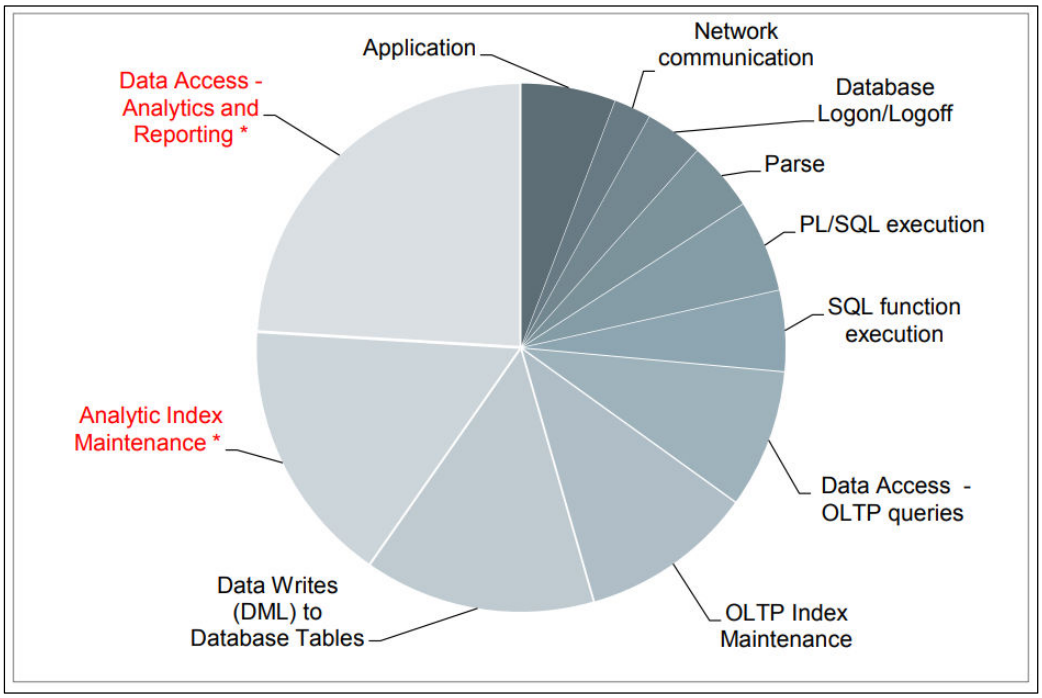
4.3. OPERAÇÕES BENEFICIADAS

De acordo com a Oracle (2015), os seguintes componentes de tempo de carga de trabalho podem se beneficiar do *Database In-Memory*:

- Acesso a dados para análises e relatórios (*Data access for analytics and reporting*): Este é o principal objetivo do *Database In-Memory*, habilitar acesso mais rápido para dados analíticos;
- Manutenção de índices analíticos (*Analytic index maintenance*): O *Database In-Memory* frequentemente permite que os índices analíticos sejam descartados, e eliminar a manutenção desses índices melhora o desempenho geral da aplicação.

A Figura 8 traz um gráfico que representa de maneira abstrata o tempo utilizado por alguns componentes de tempo de carga de trabalho em uma aplicação típica (ORACLE, 2015).

Figura 8 – Perfil de tempo para uma aplicação típica



Fonte: Oracle (2015)

O acesso a dados (*Data Access - Analytics and Reporting*) e a manutenção de índices analíticos (*Analytic Index Maintenance*) são os componentes que podem

tirar proveito do *Database In-Memory.* Os demais, por não estarem relacionados ao acesso de dados analíticos ou a qualquer aspecto da execução de SQL, não se beneficiam desta funcionalidade, sendo eles:

- Tempo de aplicação (*Application time*);
- Comunicação de rede entre cliente e banco de dados (*Network communication between client and database*);
- Logon e Logoff;
- Tempo de parse (Parse Time);
- PL/SQL e execução de funções SQL (*PL/SQL and SQL function execution*);
- Acesso a dados por *queries* OLTP (*Data access by OLTP queries*);
- Manutenção de índices OLTP (*OLTP index maintenance*);
- Escrita em tabelas do banco (*Writes to Database Tables*).

Com base na imagem, é possível também afirmar que o *Database In-Memory* é capaz de beneficiar duas das operações que mais oneram o banco de dados, sendo as quais: o acesso a dados analíticos e a relatórios e manutenção de índices analíticos.

5. METODOLOGIA E AMBIENTE DE TESTES

Como ocorre comumente, o trabalho iniciou com a pesquisa bibliográfica acerca do tema e de acordo com Fonseca (2002, apud UAB/UFRGS, 2009, pág. 37) a:

> A pesquisa bibliográfica é feita a partir do levantamento de referências teóricas já analisadas, e publicadas por meios escritos e eletrônicos, como livros, artigos científicos, páginas de web sites. Qualquer trabalho científico inicia-se com uma pesquisa bibliográfica, que permite ao pesquisador conhecer o que já se estudou sobre o assunto.

Superada a etapa de embasamento teórico, passou-se a criação do ambiente onde os testes viriam a ser realizados e, em seguida, teve início a fase da pesquisa experimental. Conforme Fonseca (2002, apud UAB/UFRGS, 2009, pág. 36):

> A pesquisa experimental seleciona grupos de assuntos coincidentes, submete-os a tratamentos diferentes, verificando as variáveis estranhas e checando se as diferenças observadas nas respostas são estatisticamente significantes. [...] Os efeitos observados são relacionados com as variações nos estímulos, pois o propósito da pesquisa experimental é apreender as relações de causa e efeito ao eliminar explicações conflitantes das descobertas realizadas.

5.1. METODOLOGIA DOS TESTES

O método usado para a busca dos resultados foi o indutivo, fundamentando as conclusões nos experimentos realizados. Experimentos estes que, em cada rodada, contaram com os seguintes passos:

1) Elaborar a *query*;
2) Executar a *query*;
3) Registrar o tempo de resposta.

A abordagem foi quantitativa, uma vez que recorreu à linguagem matemática para descrever as causas de um fenômeno e as relações entre as variáveis (FONSECA, 2002). Também por isso, os passos 2 e 3 foram repetidos por cinco vezes para cada uma das *queries* e, por fim, foi calculada a média e a mediana dos tempos destas execuções.

Foram também coletadas as imagens do plano de execução de cada uma das instruções SQL, visando a análise básica do plano escolhido pelo otimizador do banco de dados.

5.2. AMBIENTE DE TESTES

Os testes foram aplicados em servidores com a seguinte configuração:

1) Um servidor físico (*host*):
   - Processador: Intel(R) Xeon(R) CPU E5620 @ 2.40GHz;
   - Memória RAM: 63 GB;
   - Núcleos físicos: 4;
   - Sistema operacional: CentOS Linux 7 (Core) - version 3.10.0-327.36.1.el7.x86_64
2) Dois servidores virtuais (*guests*):
   - Processador: Intel(R) Core(TM) i7-4500U CPU @ 1.80GHz
   - Memória RAM: 58 GB;
   - Núcleos físicos: 2;
   - Sistema operacional: Oracle Linux 4.1.12-124.18.6.el6uek.x86_64.

Em uma das máquinas virtuais foi instalado o banco de dados Oracle Database 18c e, na outra, o Oracle TimesTen 11g. As instalações foram realizadas com as configurações padrão de cada banco de dados, dispensando maiores detalhes à respeito deste ponto.

A escolha de utilização de máquinas virtuais se deu por uma série de razões práticas. Havia apenas um servidor físico e com recursos um tanto quanto limitados se levar-se em consideração a natureza dos experimentos aqui pretendidos. Desta forma, a abordagem foi a de isolar os ambientes em duas máquinas virtuais distintas, mantendo apenas uma delas ativa por vez e utilizando o máximo de recursos possíveis para cada ambiente. O uso de máquinas virtuais também oferecia portabilidade, simplicidade de backup, conhecimento técnico prévio do sistema operacional, entre outros. Um aspecto determinante foi o fato do servidor físico não possuir acesso externo o que forçava a execução local dos testes. Com o uso de máquinas virtuais, algumas atividades da etapa de criação e configuração do ambiente puderam ser realizadas em locais distantes geograficamente do servidor físico que viria a hospedá-las durante os testes de performance.

A memória física foi configurada da seguinte maneira:

- Servidor físico (*host*) = total de 63GB de RAM

    - Sistema Operacional: 5GB
    - Máquinas virtuais: 58GB
- 18cVM = total de 58GB de RAM
    - Sistema Operacional: 3GB
    - Oracle 18c: 55GB
        - MEMORY_MAX_TARGET = 55GB
        - MEMORY_TARGET = 55G
        - INMEMORY_SIZE = 30GB
- ttVM = total de 58GB de RAM
    - Sistema Operacional: 18GB
    - Oracle TimesTen 11g: 40GB
        - PERM_ALLOCATED_SIZE: 40GB
        - PERM_IN_USE_SIZE: 30GB
        - TEMP_ALLOCATED_SIZE: 10GB

Aqui há um ponto que merece aprofundamento, enquanto na 18cVM foram disponibilizados 55GB de memória RAM para o banco de dados (valor do parâmetro MEMORY_TARGET), na ttVM o banco recebeu 40GB, que é o valor atribuído ao parâmetro PERM_ALLOCATED_SIZE, o qual equivale à soma dos parâmetros PERM_IN_USE_SIZE (30GB) e TEMP_ALLOCATED_SIZE (10GB). Esta configuração foi necessária devido às diferenças de arquitetura dos dois bancos de dados. O Oracle 18c é predominantemente um banco de dados em disco, mas que possui uma propriedade que permite trabalhar com alguns objetos em memória. Ou seja, ele ainda necessitará das áreas de memória convencionais, como SGA e PGA, para realizar as operações que envolvam os demais objetos, aqueles que não estão na INMEMORY, e para os procedimentos normais em um SGDB Oracle. O TimesTen 11g, por outro lado, é um IMDB e irá carregar todos os objetos em memória.

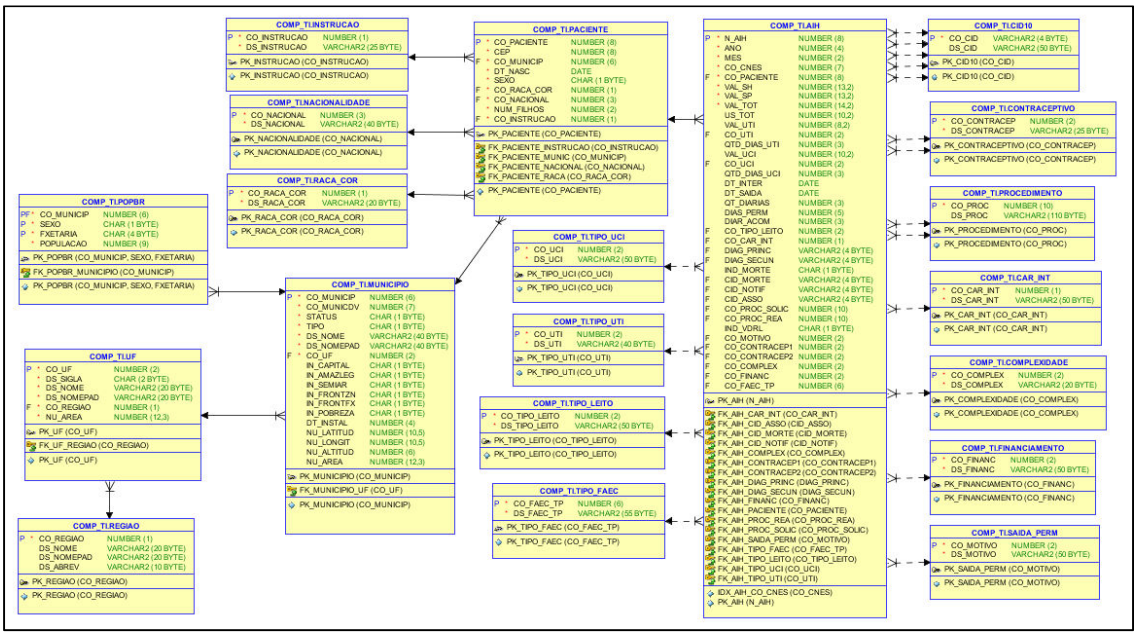
Sobre o Oracle 18c, é importante ainda ressaltar que a área de INMEMORY localiza-se dentro da SGA e, portanto, ocupa parte dos 55GB gerenciados pelo *Automatic Memory Management* (AMM). O AMM, contudo, não aumenta ou diminui a área de INMEMORY. Ele administra apenas as demais áreas dentro da SGA, como *buffer cache, redo log buffer, shared pool (library cache* e *data dictionary cache), large pool* e *java pool*.

Sendo assim, com o objetivo de garantir que ambos os bancos de dados contassem com a mesma quantidade de RAM para carregar os objetos em memória, respeitando o princípio de isonomia, foram dedicados 30GB para cada um deles. Os parâmetros que determinam isto são o INMEMORY_SIZE, no Oracle 18c, e o PERM_IN_USE_SIZE, no TimesTen 11g. Deve também ser mencionado que no Oracle 18c todas as tabelas foram configuradas com a prioridade alta (PRIORITY HIGH), fazendo com que sejam carregas em memória assim que o banco de dados é inicializado. Algo semelhante ao que ocorre no Oracle TimesTen, no qual a primeira conexão ao banco de dados dispara a carga do mesmo em memória. Mais uma vez, o objetivo foi o de criar os dois ambientes da forma mais semelhante possível.

5.3. BASE DE DADOS

A base de dados COMP_TI, utilizada para os testes, foi criada para armazenar dados disponibilizados publicamente no Portal da Saúde (http://www2.datasus.gov.br/DATASUS/index.php?area=0901) e foi cedida pelo professor da Universidade Feevale, Me. Edvar Bergmann Araújo. A Figura 9 apresenta o diagrama Entidade Relacionamento da base de dados COMP_TI.

Figura 9 – Diagrama ER da base COMP_TI



Fonte: adaptado de Araújo, 2018

O Quadro 3 apresenta uma descrição das tabelas e traz o total de registros de cada uma delas. Para que os testes fossem mais representativos, foram inseridas várias tuplas na tabela AIH, que passou de 16.999.033 para 54.741.842 de registros.

Quadro 3 – Resumo das tabelas do banco de dados COMP_TI

NOME DA TABELA	DESCRIÇÃO	REGISTROS
AIH	Autorização de Internação Hospitalar	54.741.842
CAR_INT	Caráter da Internação	6
CID10	Classificação Internacional de Doenças e Problemas Relacionados à Saúde	14.253
CNES	Cadastro Nacional de Estabelecimentos de Saúde	2.454
COMPLEXIDADE	Tabela de Complexidades de Atendimento	4
CONTRACEPTIVO	Tabela de Métodos Contraceptivos	13
FINANCIAMENTO	Tabela de Tipos de Financiamento	7
INSTRUCAO	Tabela de Graus de Instrução	5
MUNICIPIO	Tabela de Municípios	5.596
NACIONALIDADE	Tabela de Nacionalidades	332
PACIENTE	Tabela de Pacientes	12.191.575
POPBR	Tabela com a População dos Municípios	367.290
PROCEDIMENTO	Tabela de Procedimentos	5.028
RACA_COR	Tabela de Raça e Cor	6
REGIAO	Tabela de Regiões do Brasil	5
SAIDA_PERM	Tabela com Motivos de Saída e Permanência	28
TIPO_FAEC	Tabela com os subtipos de Financiamento FAEC	60
TIPO_LEITO	Tabela de Tipos de Leitos	15
TIPO_UCI	Tabela de Tipos de UCI (Unid. de Cuidados Intermediários)	4
TIPO_UTI	Tabela de Tipos de UTI (Unidade de Terapia Intensiva)	15
UF	Tabela de Unidades da Federação (UF)	27

Fonte: Araújo, 2018

6. TIMESTEN X DATABASE IN-MEMORY

A partir deste capítulo iniciam as contribuições deste trabalho, onde se buscou ir além da comparação de performance, para abordar também características e comportamentos destes dois sistemas gerenciadores de bancos de dados em memória. Embora o tema não seja tão recente, a maior parte do material que se destina à ele é elaborada por fornecedores e, muitas vezes, é parcial, imprecisa, ou omite informações sobre aspectos negativos de suas respectivas arquiteturas. Isso por si só já valoriza a existência deste trabalho, que vem para enriquecer ainda mais esta discussão. Neste capítulo foram realizadas análises e comparativos de funcionamento de forma teórica e prática, visando colocar lado a lado algumas das características mais importantes a serem consideradas quando se está pensando em migrar de um SGDB convencional para um IMDB.

O TimesTen e *Database In-Memory* são duas das soluções que a Oracle oferece para a utilização de bancos de dados em memória, mas enquanto o TimesTen é um IMDB totalmente em memória principal, que também pode ser utilizado com cache de outro banco de dados, o *Database In-Memory* é um recurso presente no banco de dados Oracle tradicional desde sua versão 12c. Portanto, o *Database In-Memory* pode ser considerado uma solução híbrida, tendo em vista que é uma propriedade de um SGDB baseado em disco, mas que carrega parte dos dados em memória e trabalha com estes dados com características próprias de um IMDB. É muito importante que fique claro que o *Database In-Memory* é um recurso presente no SGDB Oracle 18c e que ele não é um banco de dados, mas uma funcionalidade.

Além disso, também é preciso salientar que o termo *Database In-Memory* foi utilizado nesta monografia para referenciar o recurso do Oracle 18c. Já o *In-Memory Database,* foi empregado para se falar da arquitetura de banco de dados em memória – IMDB. Embora se reconheça que isto possa causar alguma confusão, é desta maneira que eles são tratados no mundo real. Assim sendo, para os fins de trabalho, os termos Oracle 18c e *Database In-Memory* funcionam como sinônimos, uma vez que aqui ambos representam o SGDB Oracle 18c com utilização do recurso *Database In-Memory.*

6.1. ARMAZENAMENTO EM MEMÓRIA

A questão do armazenamento foi aqui abordada sob o ponto de vista da memória principal. Isto é, buscou-se entender como estas duas arquiteturas mantém as informações em memória, quais os tipos de estruturas que são criadas e que estratégias são utilizadas no gerenciamento deste espaço. Deu-se pouco, ou quase nenhum, foco ao armazenamento em disco.

6.1.1. TimesTen

Assim como ocorre no *Database In-Memory*, no TimesTen as tabelas podem ser compactadas em nível de coluna. Esse mecanismo reduz a ocupação de espaço, eliminando o armazenamento redundante de valores duplicados nas colunas. Além disso, melhora o desempenho de consultas SQL que executam varreduras completas em tabelas.

O TimesTen gerencia o espaço do banco de dados, dentro de um espaço único e contíguo, usando duas regiões de memória: uma para dados permanentes e outra para dados temporários. Os dados permanentes são compostos por objetos como tabelas e índices. Quando o banco de dados é carregado em memória, o conteúdo da região de memória permanente é lido dos arquivos armazenados em disco e, durante as operações de *checkpoint*, seu conteúdo é escrito em disco. Já os dados temporários incluem bloqueios, cursores, comandos compilados e outras estruturas necessárias para a execução de comandos e para a avaliação de *queries*. A área de memória temporária é destruída quando o banco de dados é descarregado da memória, ou seja, as informações ali contidas não são persistentes (ORACLE, 2018b).

Segundo a Oracle (2018b), para certificar-se de que o tamanho máximo do segmento de memória compartilhada do sistema seja grande o suficiente para conter o banco de dados, deve ser usado o número máximo de conexões que espera-se que o banco receba e deve se garantir que o espaço definido seja maior que o seguinte:

TotMemSize = PermSize + TempSize + LogBufMB + 1 + (.043 * conexões)

Onde:

- TotMemSize: é o tamanho total da memória, em MB;
- PermSize: é o tamanho da região da memória permanente, em MB;
- TempSize: é o tamanho da região de memória temporária, em MB;
- LogBufMB: é o tamanho do buffer do log de transações interno, em MB.

Algo que também é citado pela Oracle (2018b) é que o *layout* das informações no TimesTen é otimizado para o processamento e armazenamento em memória. Enquanto em um SGDB baseado em disco as estruturas de disco e memória precisam ser quase idênticas, para o TimesTen isso não faz diferença, o que permite que as estruturas de memória sejam projetadas para minimizar as instruções de processamento. Entretanto, a Oracle não oferece um detalhamento aprofundado sobre como esta otimização ocorre.

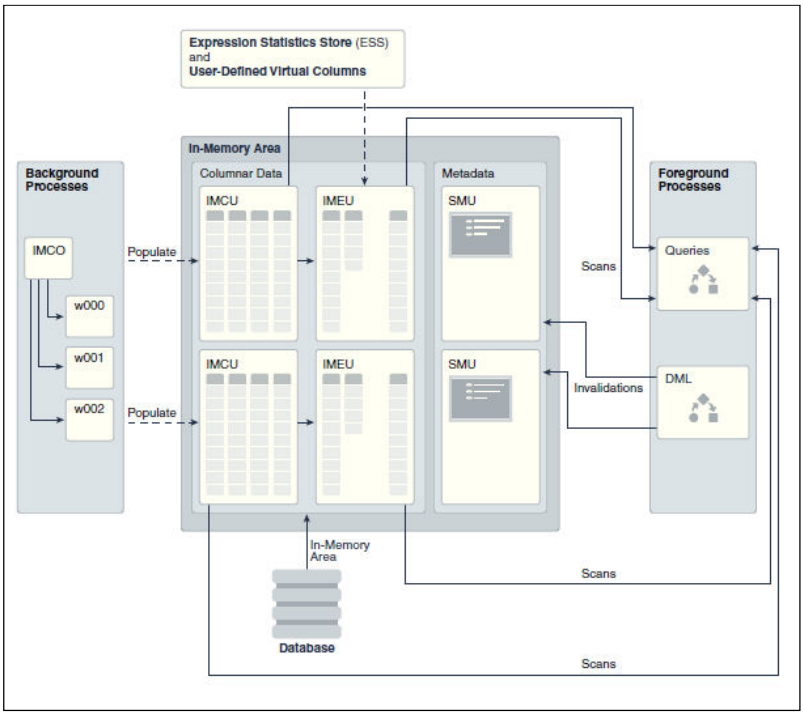
6.1.2. Database In-Memory

Conforme já foi abordado na seção 4.2, o *Database In-Memory* armazena os dados em formato colunar no *IM Column Store.* Deve ficar claro que isto ocorre somente em memória, pois em disco os dados são gravados em formato linear. Além disso, o *IM Column Store* gerencia dados e metadados em unidades de armazenamento otimizadas, não em blocos de dados tradicionais do Oracle, e usa formatos de compactação especiais otimizados para velocidade de acesso, e não para redução de armazenamento. Conforme a Oracle (2018d), esta compactação permite que as operações processem uma quantidade muito menor de dados, otimizando o desempenho das consultas. Algo mais, o banco descompacta os dados somente quando é necessário para o conjunto de resultados.

A Oracle (2018d) também garante que as instruções DML ocorrem da mesma maneira que em um SGBD baseado em disco, independentemente da *IM Column Store* estar ativa. O banco de dados usa um mecanismo interno para rastrear alterações e garantir que o *IM Column Store* esteja consistente com o restante da base.

Conforme a Oracle (2018d), as unidades de armazenamento do *IM Column Store* são (Figura 10) as *In-Memory Compression Units* (IMCUs), as *Snapshot Metadata Units* (SMUs) e as *In-Memory Expression Units* (IMEUs).

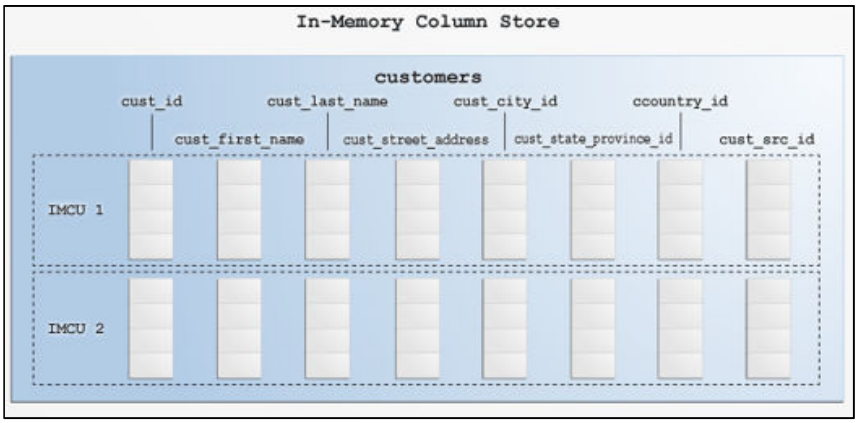
Figura 10 – *IM Column Store*: memória e processos



Fonte: Oracle (2018d)

A IMCU é uma unidade de armazenamento comprimida e somente leitura que contém dados de uma ou mais colunas. Pode ser comparada às *extents* de um *tablespace*, já que possui um conjunto de *Column Compression Units*, que são um armazenamento contíguo para uma única coluna em uma IMCU, e um cabeçalho que contém metadados (ORACLE, 2018d). Como ilustra a Figura 11, cada IMCU armazena dados em formato colunar para um único objeto.

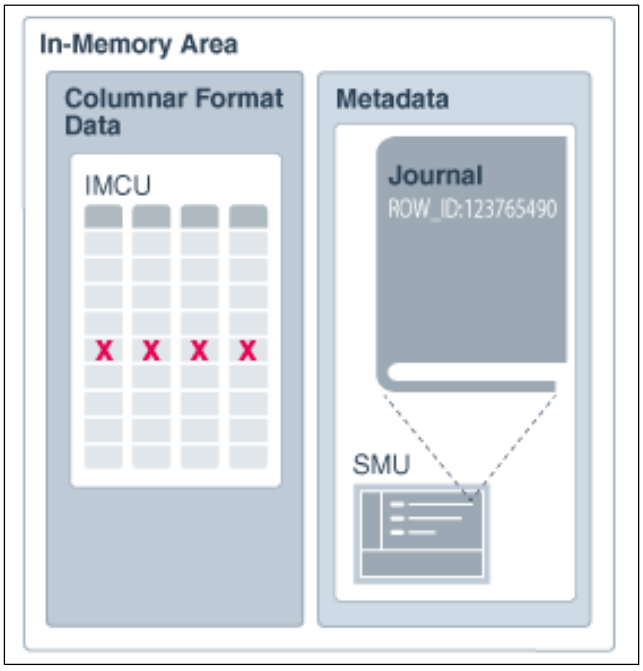
Figura 11 – IMCU



Fonte: Oracle (2018d)

De acordo com a Oracle (2018d), uma SMU contém metadados e informação transacional de uma IMCU associado. Cada IMCU aponta para uma SMU própria, logo, deverão haver tantas SMUs quanto hajam IMCUs. As SMUs armazenam muitos tipos de metadados, incluindo o número de objetos, o número de colunas, informação de mapeamento de linhas, entre outros. Cada SMU contém um diário de transações (Figura 12) que é usado para manter a IMCU consistente. Por exemplo, se uma instrução *UPDATE* modificar uma linha em uma IMCU o banco de dados adicionará o *rowid* da linha modificada ao diário de transações e o marcará como obsoleto no SCN da instrução DML. Se uma consulta precisar acessar a nova versão da linha, o banco de dados a obterá do *buffer cache*. O banco de dados atinge a consistência de leitura mesclando o conteúdo da coluna, o diário de transações e o *buffer cache.*

Figura 12 – Diário de transações



Fonte: Oracle (2018d)

"Uma *In-Memory Expression Unit* (IMEU) é um contêiner de armazenamento para *In-Memory Expressions* materializadas (expressões IM) e colunas virtuais definidas pelo usuário." (ORACLE, 2018d, cap. 2, pág. 9, tradução nossa). Uma IMEU é uma extensão lógica de uma IMCU. Cada IMEU aponta para uma IMCU e contém os resultados de expressões para os dados seu respectivo IMCU. Quando o IMCU é preenchido, o IMEU associado também é preenchido (ORACLE, 2018d).

6.1.3. Análise prática sobre armazenamento

O objetivo nesta seção foi o de realizar uma análise prática para avaliar como os dois sistemas se comportaram em relação ao tamanho dos objetos em memória. Com base no trabalho de Horn (2016), já era esperado que o TimesTen ocupasse mais espaço que o Oracle 18c, mas a questão aqui era identificar em que proporção isso ocorreu e quais as possíveis causas para este crescimento. Além disso, também foi avaliado o comportamento do *Database In-Memory* neste quesito.

6.1.3.1. TimesTen

Conforme a própria Oracle (2018g, tradução nossa), ao mover tabelas de um banco de dados Oracle para o TimesTen, normalmente é observado algum grau de "inflação" no armazenamento e isso ocorre devido às diferentes arquiteturas de armazenamento interno dos bancos de dados. Entretanto, o que ocorreu com a base COMP_TI foi muito mais do que uma simples "inflação". Os mesmos objetos que no 18c ocuparam pouco menos de 14 GB, chegaram a aproximadamente 31 GB no TimesTen. Ou seja, o TimesTen mais do que dobrou a ocupação de espaço em relação ao Oracle 18c.

Em um fórum de discussões da comunidade Oracle, um dos usuários cita pelo menos dois pontos que podem justificar um maior uso de espaço pelo TimesTen. Segundo Oracle Developer Community (2012, tradução nossa):

- No Oracle as linhas são armazenadas com tamanho variável, enquanto no TimesTen elas possuem um comprimento fixo;
- No Oracle uma coluna definida como NUMBER ocupa apenas o espaço necessário para armazenar o respectivo valor. No TimesTen uma coluna NUMBER sempre ocupa espaço para armazenar a máxima precisão possível e, portanto, ocupa 21 bytes. É possível reduzir isso explicitamente usando NUMBER (n) ou NUMBER (n, p).

Tanto no Oracle 18c, quanto no TimesTen, a estrutura e os dados do banco original foram importados sem que os tipos fossem modificados manualmente. Entretanto, o que pode ser observado na Figura 13 é que o TimesTen alterou implicitamente os tipos de algumas colunas durante a importação. Vale ressaltar que

as colunas da tabela AIH, a maior da base COMP_TI, com mais de 54 milhões de registros, são predominantemente do tipo NUMBER no banco de origem, bem como no Oracle 18c, e este tipo foi o que mais sofreu conversões.

Figura 13 – Tipos das colunas na tabela AIH

	COLUNA	ORACLE 18C	TIMESTEN
1	N_AIH	NUMBER(8,0)	TT_INTEGER
2	ANO	NUMBER(4,0)	TT_SMALLINT
3	MES	NUMBER(2,0)	TT_SMALLINT
4	CO_CNES	NUMBER(7,0)	TT_INTEGER
5	CO_PACIENTE	NUMBER(8,0)	TT_INTEGER
6	VAL_SH	NUMBER(13,2)	NUMBER(13,2)
7	VAL_SP	NUMBER(13,2)	NUMBER(13,2)
8	VAL_TOT	NUMBER(14,2)	NUMBER(14,2)
9	US_TOT	NUMBER(10,2)	NUMBER(10,2)
10	VAL_UTI	NUMBER(8,2)	NUMBER(8,2)
11	CO_UTI	NUMBER(2,0)	TT_SMALLINT
12	QTD_DIAS_UTI	NUMBER(3,0)	TT_SMALLINT
13	VAL_UCI	NUMBER(10,2)	NUMBER(10,2)
14	CO_UCI	NUMBER(2,0)	TT_SMALLINT
15	QTD_DIAS_UCI	NUMBER(3,0)	TT_SMALLINT
16	DT_INTER	DATE	DATE
17	DT_SAIDA	DATE	DATE
18	QT_DIARIAS	NUMBER(3,0)	TT_SMALLINT
19	DIAS_PERM	NUMBER(5,0)	TT_INTEGER
20	DIAR_ACOM	NUMBER(3,0)	TT_SMALLINT
21	CO_TIPO_LEITO	NUMBER(2,0)	TT_SMALLINT
22	CO_CAR_INT	NUMBER(1,0)	TT_SMALLINT
23	DIAG_PRINC	VARCHAR2(4 BYTE)	VARCHAR2(4 BYTE)
24	DIAG_SECUN	VARCHAR2(4 BYTE)	VARCHAR2(4 BYTE) inline
25	IND_MORTE	CHAR(1 BYTE)	CHAR(1 BYTE) inline
26	CID_MORTE	VARCHAR2(4 BYTE)	VARCHAR2(4 BYTE)
27	CID_NOTIF	VARCHAR2(4 BYTE)	VARCHAR2(4 BYTE) inline
28	CID_ASSO	VARCHAR2(4 BYTE)	VARCHAR2(4 BYTE) inline
29	CO_PROC_SOLIC	NUMBER(10,0)	TT_BIGINT inline
30	CO_PROC_REA	NUMBER(10,0)	TT_BIGINT
31	IND_VDRL	CHAR(1 BYTE)	CHAR(1 BYTE)
32	CO_MOTIVO	NUMBER(2,0)	TT_SMALLINT
33	CO_CONTRACEP1	NUMBER(2,0)	TT_SMALLINT
34	CO_CONTRACEP2	NUMBER(2,0)	TT_SMALLINT
35	CO_COMPLEX	NUMBER(2,0)	TT_SMALLINT
36	CO_FINANC	NUMBER(2,0)	TT_SMALLINT
37	CO_FAEC_TP	NUMBER(6,0)	TT_INTEGER

Fonte: do autor, 2019

De acordo com Oracle (2018c), os tipos TT_SMALLINT, TT_INTEGER e TTBIGINT, são mais compactos e mais performáticos que NUMBER. Os tipos TT

utilizam 2, 4 e 8 bytes, respectivamente, para o armazenamento, já NUMBER, tem tamanho variável e pode chegar a 21 bytes. Ou seja, no caso do banco COMP_TI se deve descartar ao menos uma das justificativas encontradas em Oracle Developer Community (2012), pois, como visto, o TimesTen otimizou os tipos das colunas ao importar a base.

Outro aspecto do TimesTen que pode provocar o uso de mais espaço quando comparado com o Oracle 18c, é a maneira como ele armazena os dados em memória. De acordo com Oracle (2009), colunas de comprimento variável cujo comprimento da coluna declarada é maior que 128 bytes, são armazenadas OUT OF LINE, já as que são menores ou iguais a 128 bytes são armazenadas IN LINE. Sigalaev (2012) afirma que o armazenamento IN LINE, onde o armazenamento é contiguo, é empregado por beneficiar a performance, uma vez que é ligeiramente mais rápido que o OUT OF LINE, quando a linha não é armazenada de forma contigua. Por outro lado, o armazenamento IN LINE pode causar o aumento desnecessário da base e ser responsável por desperdício de espaço. Ainda em Sigalaev (2012), podem ser vistos alguns testes que comprovam a relação entre esta estratégia de armazenamento e o desperdício de área.

Quando a base COMP_TI foi importada para o TimesTen, utilizou-se o ttImportFromOracle para gerar os arquivos da importação e todas as suas opções foram mantidas com valores padrões. Contudo, é possível executá-lo de modo que ele indique o mapeamento para o melhor tipo de dado a ser usado em uma coluna, bem como, se é exequível utilizar compressão em uma tabela. Originalmente, não se importou a base COMP_TI com compressão e nem tampouco se indicou se o TimesTen deveria ou não converter os tipos de dados, o que já se sabe que ele fez.

Isto posto, o ttImportFromOracle foi reexecutado, mas desta vez com as opções *typemap* e *compression* setadas para seus valores mais “agressivos” (Oracle, 2018g), isto é, para aqueles que podem reduzir ao mínimo o espaço utilizado pelas tabelas. O intuito era saber o quanto se poderia reduzir o tamanho do banco COMP_TI no TimesTen. O ttImportFromOracle recomendou uma série de conversões de tipos de dados e três tabelas se qualificaram para compressão. O Quadro 5 apresenta os resultados da compressão para estas tabelas e, se estes valores estiverem corretos, seria possível reduzir o tamanho do banco no TimesTen para pouco mais da metade da área que ele utilizou com a importação padrão.

Quadro 4 – Tabelas que se qualificam para a compressão

TABELA	REGISTROS	TAMANHO	TAMANHO COM COMPRESSÃO	TAXA DE COMPRESSÃO
AIH	54.741.842	12046,68 MB	5575,70 MB	0,46
PACIENTE	12.191.575	634,74 MB	538,80 MB	0,85
POPBR	367.290	16,19 MB	13,78 MB	0,85

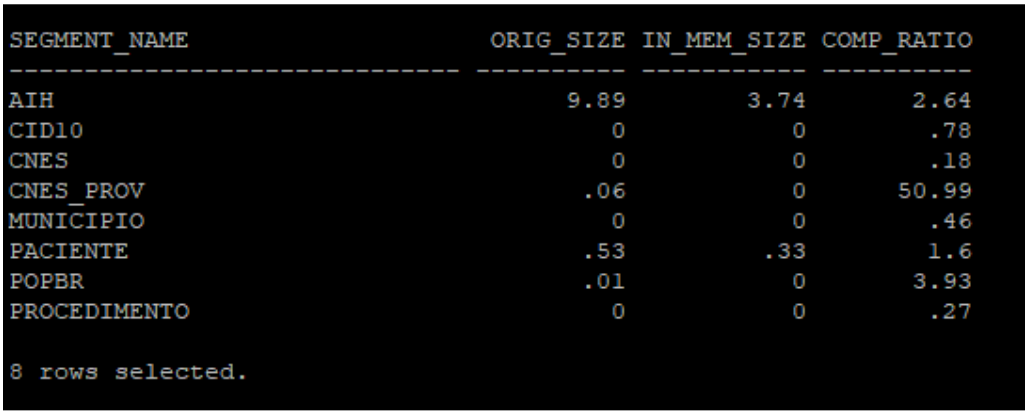
Fonte: do autor, 2019

Para concluir, foram apontadas nesta seção algumas das características que fazem com que o TimesTen acabe ocupando mais espaço do que o Oracle 18c. Todavia, há carência de fundamentação teórica e de um maior aprofundamento neste tema. O certo é que este é um tópico importantíssimo quando se está pensando em utilizar uma das duas soluções e este tema pode ser objeto de um estudo no qual o escopo seja justamente avaliar as causas desta diferença de tamanho e encontrar meios de otimizar a utilização de espaço. A conversão de tipos e a compressão provaram ser eficientes no que tange a utilização de espaço, mas precisariam ser checadas também no que diz respeito à performance.

6.1.3.2. Database In-Memory

Nestes experimentos os objetos do Oracle 18c ocuparam menos espaço em memória do que em disco e isso se deve ao fato do MEMCOMPRESS, uma subclasse do atributo INMEMORY, utilizar por padrão o tipo de compressão QUERY LOW (ORACLE, 2018e). Em tese, este tipo é otimizado para ganho de performance, mas o que pode ser observado é que houve também um grande ganho em termos de espaço. A Figura 14, apresenta dados sobre a taxa de compressão das tabelas do banco COMP_TI e, para tanto, traz quatro colunas: SEGMENT_NAME (nome da tabela), ORIG_SIZE (tamanho original da tabela em GB), IN_MEM_SIZE (tamanho da tabela em memória em GB) e COMP_RATIO (taxa de compressão). Analisando os dados é possível ver, por exemplo, que a tabela AIH teve uma taxa de compressão de 2,64 em relação ao seu tamanho em disco.

Figura 14 – Taxa de compressão das tabelas no 18c



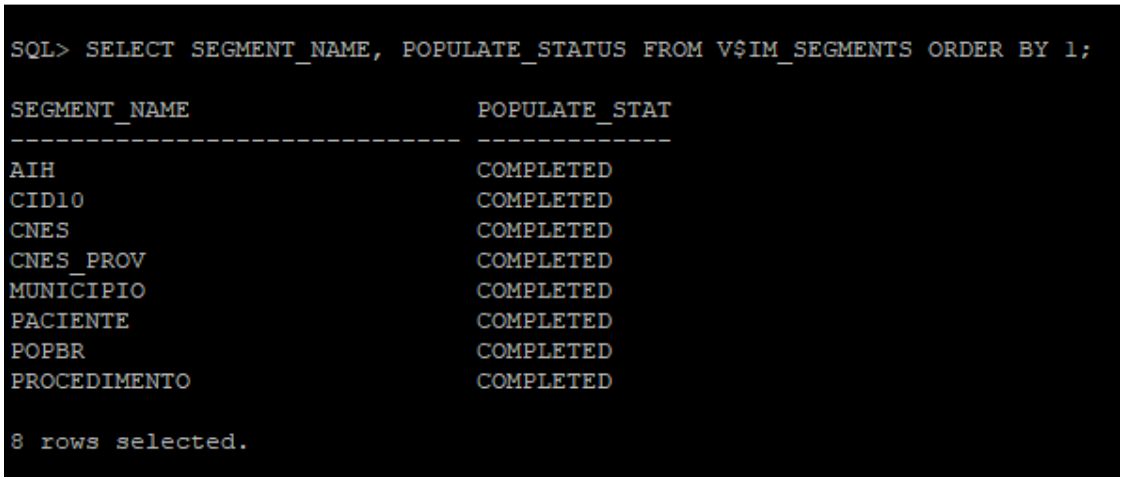
```
SEGMENT_NAME                    ORIG_SIZE IN_MEM_SIZE COMP_RATIO
------------------------------ ---------- ----------- ----------
AIH                                  9.89        3.74       2.64
CID10                                   0           0        .78
CNES                                    0           0        .18
CNES_PROV                             .06           0      50.99
MUNICIPIO                               0           0        .46
PACIENTE                              .53         .33        1.6
POPBR                                 .01           0       3.93
PROCEDIMENTO                            0           0        .27

8 rows selected.
```

Fonte: do autor, 2019

Outro aspecto do Oracle 18c é que ele não populou tabelas pequenas em memória, ainda que elas estivessem configuradas para isso. De acordo com a Oracle (2017b), a *view* V$IM_SEGMENTS exibe dados apenas de segmentos que estão em memória e se um segmento está marcado para o *IM column store*, mas não foi populado, não haverá uma linha correspondente à ele nesta visão. Como pode ser observado na Figura 15, apenas oito tabelas, de um total de vinte e duas, foram carregadas em memória.

Figura 15 – Tabelas populadas em memória no 18c



```
SQL> SELECT SEGMENT_NAME, POPULATE_STATUS FROM V$IM_SEGMENTS ORDER BY 1;

SEGMENT_NAME                   POPULATE_STAT
------------------------------ -------------
AIH                            COMPLETED
CID10                          COMPLETED
CNES                           COMPLETED
CNES_PROV                      COMPLETED
MUNICIPIO                      COMPLETED
PACIENTE                       COMPLETED
POPBR                          COMPLETED
PROCEDIMENTO                   COMPLETED

8 rows selected.
```

Fonte: do autor, 2019

No Quadro 6 são listadas as tabelas que não foram carregas e a quantidade de registros que cada uma possui. Como pode visto, trata-se de tabelas muito

pequenas e objetos que são menores que 64 KB não são carregados na memória, porque desperdiçam uma quantidade considerável de espaço dentro do *IM Column Store*, pois a memória é alocada em blocos de 1 MB. (ORACLE, 2018e, tradução nossa)

Quadro 5 – Tabelas que não foram carregadas em memória

NOME DA TABELA	REGISTROS
CAR_INT	6
COMPLEXIDADE	4
CONTRACEPTIVO	13
FINANCIAMENTO	7
INSTRUCAO	5
NACIONALIDADE	332
RACA_COR	6
REGIAO	5
SAIDA_PERM	28
TIPO_FAEC	60
TIPO_LEITO	15
TIPO_UCI	4
TIPO_UTI	15
UF	27

Fonte: do autor, 2019

Finalizando, pode se dizer que, comparado ao TimesTen, o 18c utilizou melhor o espaço em memória. Pelo menos, é nesse sentido que apontam os indícios coletados durante os experimentos.

Mais uma vez, recomenda- se que novos testes sejam feitos, simulando, por exemplo, outros tipos de compactação. Aliás, estes experimentos não foram realizados nesta monografia porque ela tem um caráter mais amplo e busca dar uma

visão geral sobre diversos aspectos relacionados aos IMDBs, servindo de base para trabalhos futuros que possam explorar melhor algumas destas peculiaridades.

O tempo necessário para que fossem feitas diversas importações para o TimesTen, simulando diferentes cenários, ou para que fossem utilizados diferentes parâmetros de compactação no Oracle 18c, não permitiria que este trabalho abordasse outras características dos IMDBs que fazem parte de seu escopo.

6.2. DURABILIDADE

O conceito e algumas das características da durabilidade já foram abordados na seção 2.2., mas o objetivo desta seção era o de identificar como TimesTen e *Database In-Memory* asseguram esta propriedade ACID. Conforme será visto em 6.2.1. e 6.2.2., nenhuma das duas arquiteturas apresenta grandes inovações neste quesito, mas o mais importante é que ambas garantem a durabilidade. Pode se dizer inclusive, que elas o fazem praticamente da mesma forma: através do uso de logs transacionais.

6.2.1. TimesTen

O TimesTen oferece consistência e durabilidade através de uma combinação de *checkpoint* e registro de transações. “Uma operação de *checkpoint* grava a imagem atual do banco de dados na memória em um arquivo de *checkpoint* no disco, o que tem o efeito de tornar consistentes e duráveis todas as transações que foram confirmadas no momento da operação do *checkpoint*.” (ORACLE, 2018b, tradução nossa).

Segundo a Oracle (2018b), todas as transações são registradas em memória em um *buffer* de log de transações que é gravado em disco em uma das seguintes maneiras:

- Síncrono (durabilidade garantida através de um *commit* durável): Qualquer recuperação usará a última imagem de *checkpoint* em conjunto com o log de transação para reconstruir o banco de dados para o seu estado consistente mais recente. O controle retorna à aplicação somente após os dados do log

de transações terem sido gravados de forma durável no disco e mesmo em caso de falha do sistema, uma transação com *commit* durável não será perdida (ORACLE, 2018b);

- Assíncrono (atrasos de durabilidade por meio de um *commit* não durável): Como no modo de durabilidade garantida, cada transação insere registros no log de transações na memória. No entanto, quando uma transação é confirmada no modo de durabilidade atrasada, ela não espera que o log de transações seja gravado no disco antes de retornar o controle para a aplicação. Assim, uma transação não durável pode ser perdida no caso de uma falha no banco de dados, por outro lado, executa mais rápido do que transações duráveis. As transações são liberadas para o disco pelo processo de sub-rotina do banco de dados ou quando o *buffer* de log da memória está cheio (ORACLE, 2018b).

Conforme a Oracle (2011, cap. 1, pág. 5, tradução nossa):

> O TimesTen mantém a versão residente no disco do banco de dados com uma operação de *checkpoint* que ocorre em segundo plano e tem pouco impacto nos aplicativos de banco de dados. Essa operação é chamada de *checkpoint* "*fuzzy*" e é executada automaticamente. TimesTen também têm um *checkpoint* de bloqueio que não requer arquivos de log de transações para recuperação. Os *checkpoints* de bloqueio devem ser iniciados pela aplicação. O TimesTen mantém dois arquivos de *checkpoint* caso ocorra uma falha no meio do *checkpoint*. Os arquivos de *checkpoint* devem residir em discos separados dos logs de transação para minimizar o impacto do *checkpoint* na atividade da aplicação.

6.2.2. Database In-Memory

O Oracle 18c garante a durabilidade da mesma maneira que as outras versões de SGDBs tradicionais da Oracle, fazendo uso de logs transacionais, ou como a própria Oracle chama: *redo logs*. De acordo com Legatti; Furushima (2015, pág. 1), para assegurar a durabilidade dos dados:

> "...um SGBD de modelo transacional clássico (banco de dados relacional) mantém um registro de log para operações de escritas no banco de dados, nomeado em contexto Oracle, como *redo logs*. Essa estrutura é responsável por garantir as propriedades de atomicidade (A) e durabilidade (D) de modo que se o banco de dados sofrer uma eventual queda antes que os dados alterados sejam escritos de forma persistente em disco, o log será utilizado para restaurar essas informações quando o sistema for normalizado."

Como o 18c não difere das versões anteriores do Oracle no que tange à durabilidade e como ele utiliza a *IM Column Store* apenas para as operações de leitura, talvez o mais interessante aqui seja a forma como ele garante o sincronismo entre o que foi persistido e os dados na *In-Memory Area*. Isso foi discutido em maiores detalhes na seção 6.1.2, mas vale lembrar que a consistência de leitura é alcançada por um mecanismo que une o conteúdo da coluna, o diário de transações e o *buffer cache.*

Enfim, no que diz respeito à durabilidade o Oracle 18c age exatamente como qualquer outro SGDB baseado em disco.

6.2.3. Experimento prático de recuperação de desastre

Neste teste o que se pretendia aferir era qual dos dois sistemas seria capaz de recuperar-se mais rapidamente após um desastre, como em caso de falta de energia ou qualquer outra situação capaz de provocar um *shutdown* inesperado do servidor onde o banco de dados reside.

As etapas seguidas para a simulação de desastre foram as seguintes:

1) Conectou-se ao banco de dados;
2) Aplicou-se um *insert* com *commit* (Figura 16);

Figura 16 – *Insert* com *commit*

```
1  INSERT INTO COMP_TI.AIH
2    (N_AIH, ANO,MES, CO_CNES, CO_PACIENTE, VAL_SH, VAL_SP, VAL_TOT)
3  VALUES (COMP_TI.AIH_SEQ_II.NEXTVAL,2019,05,2223589,622737,10,10,10);
4  COMMIT;
```

Fonte: do autor, 2019

3) Aplicou-se outro *insert*, desta vez sem *commit* (Figura 17);

Figura 17 – *Insert* sem *commit*

```
1  INSERT INTO COMP_TI.AIH
2    (N_AIH, ANO,MES, CO_CNES, CO_PACIENTE, VAL_SH, VAL_SP, VAL_TOT)
3  VALUES (COMP_TI.AIH_SEQ_II.NEXTVAL,2019,05,2223589,656172,20,20,20);
```

Fonte: do autor, 2019

4) Aplicou-se *shutdown* no servidor em que está o banco de dados;
5) Aplicou-se *start* no servidor em que está o banco de dados;
6) Aplicou-se *start* no banco de dados;
7) O Banco de dados abriu para transações.

6.2.3.1. TimesTen

O TimesTen levou aproximadamente 6 minutos para abrir o banco. Veja na Figura 18 que ele persistiu as duas instruções mesmo não havendo um *commit* explícito para o segundo *insert*. Isto ocorre porque, o TimesTen utilizada por *default* a opção de *autocommit*.

Figura 18 – Persistência após desastre – TimesTen

```
Command> SELECT N_AIH, ANO,MES, CO_CNES, CO_PACIENTE, VAL_SH, VAL_SP, VAL_TOT
       > FROM COMP_TI.AIH
       > WHERE VAL_TOT = 10;
< 100150041, 2019, 5, 2223589, 622737, 10, 10, 10 >
1 row found.
Command> SELECT N_AIH, ANO,MES, CO_CNES, CO_PACIENTE, VAL_SH, VAL_SP, VAL_TOT
       > FROM COMP_TI.AIH
       > WHERE VAL_TOT = 20;
< 100150042, 2019, 5, 2223589, 656172, 20, 20, 20 >
1 row found.
```

Fonte: do autor, 2019

6.2.3.2. Database In-Memory

Ainda que necessitasse de mais algum tempo para que todas as tabelas fossem carregadas no *IM Column Store*, passado 1 minuto do comando de *start,* o Oracle 18c já estava apto a receber transações. Como já se sabe, o 18c mantém toda a parte do SGDB convencional o que faz com que ele esteja disponível mesmo antes do carregamento em memória haver concluído. Além disso, a transação que recebeu *commit* foi corretamente persistida e a outra sofreu *rollback* como era esperado (Figura 19);

Figura 19 – Persistência após desastre – 18c

```
SQL> SELECT N_AIH, ANO,MES, CO_CNES, CO_PACIENTE, VAL_SH, VAL_SP, VAL_TOT
FROM COMP_TI.AIH
WHERE VAL_TOT = 10;  2    3

     N_AIH        ANO        MES    CO_CNES CO_PACIENTE     VAL_SH     VAL_SP    VAL_TOT
---------- ---------- ---------- ---------- ---------- ---------- ---------- ----------
 100870792       2019          5    2223589      622737         10         10         10

SQL> SELECT N_AIH, ANO,MES, CO_CNES, CO_PACIENTE, VAL_SH, VAL_SP, VAL_TOT
FROM COMP_TI.AIH
WHERE VAL_TOT = 20;  2    3

no rows selected
```

Fonte: do autor, 2019

O que se percebe ao avaliar os resultados é que não houve alterações significativas entre os tempos que ambos os bancos de dados levaram para abrir após um *shutdown* normal ou após o desastre. Contudo, não há dúvidas de que o tempo para recuperação deve variar de acordo com o tamanho do banco de dados, bem como, com o volume de transações que necessitará ser recuperada para que o banco retome um estado consistente. Além disso, é importante considerar que o TimesTen necessita de um tempo maior para abrir o banco, mesmo em condições normais. Nestes experimentos ele precisou de cinco a seis vezes mais tempo para estar disponível.

6.3. INDEXAÇÃO

A Oracle (2018d) afirma que é necessária a criação de estruturas de acesso para melhorar o desempenho de consultas analíticas quando os dados são armazenados em linhas. A abordagem padrão é a criação de índices analíticos, visões materializadas e cubos OLAP, mas, embora essas técnicas melhorem o desempenho de consultas analíticas, elas diminuem o desempenho de transações OLTP. Isso ocorre porque para inserir uma linha em uma tabela é necessário modificar todos os índices que pertencem a ela e, à medida que o número de índices aumenta, a velocidade de inserção diminui.

Apesar de algumas diferenças, os tipos de índices utilizados pelo TimesTen e pelo *Database In-Memory* não apresentam grandes inovações em relação aos de um SGDB Oracle baseado em disco. Novamente, é importante lembrar que o *Database*

In-Memory não é um SGDB, mas um recurso presente no Oracle Database e, como tal, faz uso dos mesmos tipos de índices desta plataforma, além de um tipo de índice novo, o *In-Memory Storage Index*, sobre o qual será tratado na seção 6.3.2.. Talvez as maiores contribuições do TimesTen e do *Database In-Memory,* no que tange à índices, ficam por conta da redução no número de índices analíticos, que ocorre em ambas as soluções, e da redução no tamanho de cada índice, no caso do TimesTen.

Alguns testes práticos relacionados à indexação estarão presentes no capítulo 7, onde foram abordados os experimentos de performance. Além disso, trabalhos futuros poderiam testar se de fato os IMDBs permitem o descarte de estruturas de acesso analítico, o que reduziria a quantidade de índices e traria benefícios também a transações OLTP. Tendo em vista que identificar a possibilidade de remover um índice não é algo trivial, conforme é abordado de maneira sucinta na sessão 6.3.2., não foi possível a realização deste tipo de experimento no presente trabalho.

6.3.1. TimesTen

Segundo a Oracle (2018b), o TimesTen executa correspondências exatas rápidas através de índices de *hash*, índices de *bitmap* e pesquisas pelo *rowid,* já as correspondências de intervalo ocorrem por meio de índices de *range*. As dicas do *otimizador* podem ser usadas para permitir ou impedir que o *otimizador* considere determinados métodos de verificação ao escolher um plano de consulta. Uma varredura completa da tabela examinará todas as linhas e, como é a maneira menos eficiente de se avaliar um predicado de consulta, só será usada quando nenhum outro método estiver disponível.

Tal qual um SGDB Oracle baseado em disco, o TimesTen atribui uma identidade exclusiva, chamada *rowid*, a cada linha armazenada em uma tabela. As pesquisas pelo *rowid* são mais rápidas que as que utilizam índices (ORACLE, 2018b).

A Oracle (2018b) afirma que o TimesTen possui três diferentes tipos de índices:

- *Range*: aplicável a predicados de correspondência exata, ou a predicados de intervalo, desde que a coluna usada no predicado tenha um índice de intervalo

definido sobre ela. Se um índice de intervalo for definido em várias colunas, ele poderá ser usado para predicados de várias colunas;

- *Hash:* localiza linhas com uma correspondência exata em uma ou mais colunas. Essas pesquisas são aplicáveis a pesquisas de igualdade em uma ou mais colunas especificadas;
- *Bitmap:* usado para localizar linhas que satisfaçam um predicado de igualdade e são apropriados para colunas com poucos valores exclusivos. São particularmente úteis na avaliação de vários predicados, cada um dos quais pode usar uma pesquisa de índice de *bitmap* porque os predicados combinados podem ser avaliados eficientemente por meio de operações de bit nos próprios índices.

De acordo com a Oracle (2018b), cada entrada de índice em um SGDB baseado em disco geralmente contém uma chave e um identificador de registro, que permite ao SGDB localizar o registro no disco. Entretanto, em um IMDB como o TimesTen, as chaves não precisam ser armazenadas nos índices e identificadores de registro podem ser implementados como ponteiros de registro, os quais apontam para o registro que contém a chave. Os principais benefícios ao se evitar a duplicação de valores-chave na estrutura do índice são a redução no tamanho de cada índice e a simplificação ao se implementar um índice, uma vez que se evita a necessidade de gerenciar chaves de tamanho variável dentro dos nós.

6.3.2. Database In-Memory

De acordo com a Oracle (2018d), ao carregar os dados na *IM Column Store* podem ser descartadas estruturas de acesso analítico. Essa técnica reduz o espaço de armazenamento e a sobrecarga de processamento porque são necessários menos índices, visões materializadas e cubos OLAP. Entretanto, embora a *IM Column Store* melhore o desempenho de consultas analíticas e cargas de trabalho de *data warehouse*, ela beneficia menos os bancos de dados OLTP que executam transações curtas usando pesquisas por índices. O desempenho dos seguintes tipos de consultas não é beneficiado pela *IM Column Store*:

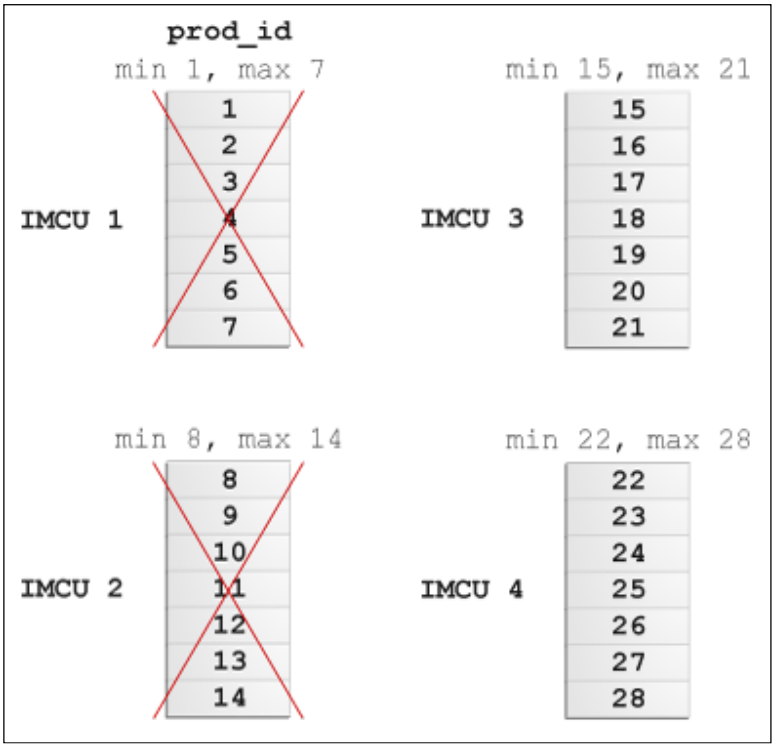
- Uma consulta com predicados complexos;
- Uma consulta que seleciona um grande número de colunas;

– Uma consulta que retorna um grande número de linhas.

Idealmente, os índices de relatórios analíticos podem ser descartados ao usar o *Database In-Memory*, mas muitas vezes pode ser difícil determinar quais índices são analíticos e quais estão sendo usados para consultas do tipo OLTP. Por esta razão, tornar os índices invisíveis é uma abordagem mais segura do que simplesmente removê-los. Se estiverem invisíveis ao *otimizador*, não serão utilizados, mas se for constatado que há necessidade de utilização, é muito simples torná-los visíveis novamente. Após executar alguns ciclos com os índices invisíveis e nenhum problema de desempenho ocorrer em razão da falta deles, poderão ser descartados com segurança (ORACLE, 2017).

Além dos índices convencionais, o *Database In-Memory* também faz uso de um tipo de índice em memória, o *In-Memory Storage Index*. Cada cabeçalho de um IMCU automaticamente cria e gerencia *In-Memory Storage Indexes*. Estes índices armazenam os valores mínimo e máximo para cada coluna de um IMCU, permitindo que o banco de dados execute as consultas considerando somente os IMCUs que satisfazem a um determinado predicado (Figura 20).

Figura 20 – *In-Memory Storage Index*



Fonte: Oracle (2018d)

6.4. TUNING DE INSTRUÇÕES SQL

De acordo com Prado (2014, pág. 1):

> Em TI, *tuning* refere-se basicamente ao conceito de propor e aplicar mudanças visando otimizar o desempenho na recuperação ou atualização de dados. Em curtas palavras, *tuning* (em TI) é sinônimo de otimização. Atualmente existem muitas técnicas e dicas de *tuning* que podem ser aplicadas para otimizar os sistemas corporativos, compreendo-os desde o nível do sistema operacional, até o nível do seu código-fonte.

Prado (2014) afirma ainda que podem realizadas três tipos de atividades de *tuning* em um banco de dados:

1) Planejamento de performance: Definição e configuração do ambiente em que o SGDB será instalado. Deve se considerar itens como *hardware*, *software*, sistema operacional e infraestrutura de rede;
2) *Tuning* de instância e de banco de dados: Ajuste de parâmetros e configurações do SGDB;
3) SQL *tuning*: Otimização de instruções SQL.

Neste trabalho foram abordadas essencialmente as estratégias de otimização que dizem respeito a instruções SQL. Contudo, isto não impede que em alguns momentos tenham sido citadas técnicas que possam fazer parte do planejamento de performance ou do *tuning* de banco de dados. No capítulo 7, onde são apresentados os experimentos de performance, foram realizados uma série de testes práticos relacionados ao *tuning* de instruções SQL, como:

- Criação de índices;
- Utilização de índice de chave estrangeira;
- Consultas aninhadas;
- *Joins* com filtros *Bloom;*
- Entre outros.

Além disso, foram coletadas as estatísticas de ambos os bancos de dados antes de serem iniciados os testes de desempenho e no *Database In-Memory* as tabelas foram compactadas na *IM Column Store*, o que é o comportamento padrão, conforme já citado na seção 6.1.3.2..

6.4.1. TimesTen

Muitas das estratégias de *tuning* em termos de instruções SQL recomendadas pela Oracle para o TimesTen, são as mesmas empregadas em um SGDB baseado em disco. A lista a seguir apresenta estas recomendações e o impacto estimado para cada uma delas no desempenho do banco de dados (ORACLE, 2018d):

- Ajustar as declarações e utilizar índices (grande);
- Coletar e avaliar a amostragem de tempos de execução para instruções SQL (variável);
- Selecionar adequadamente os índices de *hash*, *range* ou *bitmap* (variável);
- Dimensionar os índices de *hash* apropriadamente (variável);
- Usar a restrição de chave estrangeira apropriadamente (variável);
- Calcular estatísticas exatas ou estimadas (grande);
- Coletar estatísticas para tabelas grandes em paralelo (grande);
- Criar script para coletar as estatísticas atuais do banco de dados (variável);
- Controlar a invalidação de comandos no cache do comando SQL (variável);
- Evitar ALTER TABLE (variável);
- Evitar consultas aninhadas (variável);
- Preparar as declarações antecipadamente – usar variáveis *bind* sempre que possível (variável);
- Evitar operações de preparação desnecessárias (grande);
- Armazenar os dados de forma eficiente com a compactação de tabelas (grande);
- Controlar a otimização de leitura durante operações de gravação concorrentes (variável).

6.4.2. Database In-Memory

Pode se dizer que o *Database In-Memory* é uma solução híbrida, uma vez que mantém todas as características de um SGDB convencional, somadas à utilização da *IM Column Store*. Por esta razão, um banco de dados utilizando o *Database In-Memory* deve seguir as mesmas recomendações de *tuning* de SQL que se aplicam a um banco baseado em disco. E, além disso, deve empregar estratégias de otimização

para as consultas que utilizam dados *in-memory*. Em Oracle (2018d) são listadas quatro estratégias: (I) *IM Expressions*; (II) *Join Groups*; (III) *IM Aggregation*; (IV) Otimizar o repovoamento da *IM Column Store*.

6.4.2.1. IM Expressions

Segundo a Oracle (2018d), no contexto da *IM Column Store*, uma expressão é uma combinação de um ou mais valores, operadores e funções SQL ou PL/SQL que resolvem para um valor. As expressões avaliadas com mais frequência são automaticamente armazenadas pelo *Expression Statistics Store* (ESS). Por exemplo, se *last_name* for uma coluna armazenada na *IM Column Store*, *UPPER (last_name)* poderia ser uma *IM Expression*.

> Uma IM Expression é materializada como uma coluna virtual oculta, mas é acessada da mesma forma que uma coluna não virtual. Para armazenar as expressões materializadas, o repositório de colunas de mensagens instantâneas usa formatos de compactação especiais, como vetores de largura fixa e codificação de dicionário com códigos de largura fixa. (ORACLE, 2018d, cap. 1, pág. 5, tradução nossa)

As *IM Expressions* aceleram consultas de grandes conjuntos de dados pré-computando expressões computacionalmente intensas. Elas beneficiam especialmente as junções de tabelas, projeções e avaliações de predicado executadas com frequência (ORACLE, 2018d). Conforme a Oracle (2018d), dentre as vantagens das *IM Expressions,* estão:

- O fato de que uma consulta não precisa recalcular as expressões todas as vezes que é executada;
- A possibilidade de o banco de dados tirar proveito de recursos como o processamento vetorial SIMD e a remoção de IMCU;
- E o próprio banco, e não o usuário, rastrear as expressões mais ativas.

Tanto as *IM Expressions* quanto as visões materializadas tentam evitar a avaliação repetida de expressões. Todavia, as *IM Expressions* podem capturar dados que não são armazenados persistentemente, qualquer consulta que contenha uma *IM Expression* pode se beneficiar dela e o próprio banco de dados às identifica e às cria automaticamente (ORACLE, 2018d).

6.4.2.2. Join Groups

Um *Join Group* é um objeto de dicionário criado pelo usuário que lista duas colunas que podem ser unidas de forma significativa. A Oracle (2018d) o define como “um conjunto de colunas nas quais um conjunto de tabelas é frequentemente associado. O conjunto de colunas contém uma ou mais colunas e o conjunto de tabelas contém uma ou mais tabelas. As colunas no *Join Group* podem estar na mesma tabela ou em tabelas diferentes.” Por outro lado, uma coluna só pode ser membra de um *Join Group*.

A Oracle (2018d) atesta que os seguintes *joins* são executados mais rapidamente quando as tabelas estão na *IM Column Store*:

- *Joins* que podem utilizar filtros *Bloom* (um tipo de filtro que verifica a probabilidade de um elemento fazer parte de um conjunto);
- *Joins* de várias tabelas de dimensões pequenas com uma tabela de fatos;
- *Joins* entre duas tabelas que possuem um relacionamento de chave primária e chave estrangeira.

De acordo com a Oracle (2018d), em determinadas consultas, os *Join Groups* eliminam a sobrecarga de desempenho para descompactar e indexar os valores de uma coluna. Sem *Join Groups*, se o otimizador usar um *hash join*, mas não puder utilizar um filtro *Bloom*, ou se o filtro *Bloom* não filtrar as linhas efetivamente, o banco de dados deverá descompactar as IMCUs e usar um *hash join* com um custo alto.

Quando há um *Join Group*, o banco de dados armazena códigos para cada valor de coluna do *join* em um dicionário comum. O banco de dados cria um array de *Join Groups* usando códigos de dicionário. Cada elemento da matriz aponta para uma linha do lado da construção armazenada na área de hash. Durante o verificação, cada linha verificada possui um código associado à chave de junção. O banco de dados usa esse código para pesquisar a matriz para determinar se existe um ponteiro no elemento da matriz. Se um ponteiro existe, então há uma correspondência; caso contrário, não há correspondência (ORACLE, 2018d). A Oracle (2018d) sustenta que os principais benefícios do uso de *Join Groups* são:

- O banco de dados opera em dados compactados;

- O banco de dados evita fazer o *hashing* na chave de *join* e examinar a tabela de *hash*;
- Os códigos do dicionário são densos e têm um comprimento fixo, o que os torna eficientes em termos de espaço;
- Otimizar uma consulta com um *Join Group* é possível às vezes, mesmo quando não é possível usar um filtro *Bloom*.

6.4.2.3. IM Aggregation

A *IM Aggregation* (ou *Vector Aggregation*) otimiza os blocos de consulta que envolvem agregações e *joins* de uma tabela grande com várias tabelas pequenas. Antes do *Oracle Database* 12c, as únicas operações de *GROUP BY* eram *HASH* e *SORT*. O *VECTOR GROUP BY* é uma transformação adicional baseada em custo que transforma um *join* em um filtro. O banco de dados pode aplicar esse filtro durante a varredura da tabela de fatos. Os joins usam vetores-chave, que são semelhantes aos filtros *Bloom*, e a agregação usa um *VECTOR GROUP BY* (ORACLE 2018d).

Conforme a Oracle (2018d), a utilização da *IM Aggregation* permite que os *joins* de vetor e as operações de *GROUP BY* ocorram simultaneamente à varredura da tabela de fatos. Assim, essas operações são agregadas à medida que são verificadas e não precisam esperar pela conclusão das varreduras de tabelas e das operações de *join*.

A agregação *VECTOR GROUP BY* não beneficia o desempenho nos seguintes cenários:

- Quando o *join* ocorre entre duas tabelas muito grandes;
- Quando as dimensões contêm mais de 2 bilhões de linhas;
- Quando o sistema não possui memória suficiente.

6.4.2.4. Otimizando o repovoamento da IM Column Store

“A atualização automática dos dados colunares após modificações significativas é chamada de repovoamento.” (ORACLE, 2018d, cap. 8, pág. 1, tradução nossa)

Uma IMCU é uma estrutura somente leitura que não modifica os dados quando ocorre um DML na tabela. Em vez disso, a SMU associada a cada IMCU controla as modificações de linha em um diário de transação (*transaction journal*). Se uma consulta acessa os dados e descobre linhas modificadas, ela pode obter do diário de transações os *rowids* correspondentes e, em seguida, recuperar as linhas modificadas do *buffer cache*. Entretanto, à medida que o número de modificações aumenta, também aumenta o tamanho das SMUs e a quantidade de dados que devem ser pesquisados no diário de transações ou no *buffer cache*. Para evitar a degradação do desempenho da consulta, os processos em segundo plano recarregam os objetos modificados (ORACLE, 2018d).

O repovoamento automático, de acordo com a Oracle (1028c), assume duas formas: repovoamento baseado em *threshold* e repovoamento *trickle*. A primeira forma depende da porcentagem de entradas antigas no diário de transações de uma IMCU. A segunda, suplementa o repovoamento baseado em *threshold*, atualizando periodicamente os dados obsoletos, mesmo quando o limite não foi atingido.

A Oracle (2018d) estabelece que os fatores que mais afetam o repovoamento da *IM Column Store* são:

- Taxa de operações DML;
- Tipo de operações DML;
- Localização das linhas modificadas dentro do bloco de dados;
- Nível de compressão aplicado aos objetos em memória;
- Número de processos *worker* ativos.

Por fim, vale ressaltar que o repovoamento ocorre automaticamente por padrão, mas é possível controlar sua frequência e até mesmo desativá-lo por completo.

6.5. COMPARAÇÃO

O Quadro 6 traz a comparação entre várias características das duas tecnologias e através dele é possível perceber algumas das principais diferenças e semelhanças entre os dois bancos de dados objetos deste estudo.

Quadro 6 – TimesTen x *Database In-Memory*

	ORACLE TIMESTEN 11g	ORACLE 18c – IN-MEMORY
TIPO	IMDB	Híbrido
FOCO	OLTP e OLAP	OLAP (OLTP em disco)
LAYOUT DE ARMAZENAMENTO	Linear e colunar	Colunar (dados em memória)
MEMÓRIA	Área específica	*System Glogal Area* (*In-Memory area*)
OBJETOS EM MEMÓRIA	O banco inteiro precisa caber em memória, ou ser usado como cache de outro banco de dados	Os objetos que serão armazenados em memória são eleitos
ÍNDICES	Reduz tamanho dos índices e quantidade de índices analíticos	Reduz quantidade de índices analíticos
ACID	Sim	Sim
COMPACTAÇÃO	Sim	Sim
TAMANHO EM MEMÓRIA	20% a mais (duplo *layout*)	Menor
BUFFER CACHE	Não existe	Está presente
LAYER	Recomendado implementar na mesma camada da aplicação	Camada de banco de dados
IMPLANTAÇÃO	Mais complexa	Simples (se já houver um SGBD Oracle baseado em disco)
CURVA DE APRENDIZAGEM	Média (se já utilizar um SGBD Oracle baseado em disco)	Praticamente nenhuma (se já utilizar um SGBD Oracle baseado em disco)
TEMPO DE INICIALIZAÇÃO	Nestes experimentos foi até 5 vezes maior que o 18c	Normal (pode carregar parcialmente em memória)
SUPORTE A SQL E PL/SQL	Sim, mas com algumas poucas restrições	Exatamente como em um SGDB Oracle convencional

Fonte: do autor, 2018

7. EXPERIMENTOS DE PERFORMANCE

Neste capítulo abordou-se a realização dos experimentos de performance e cada seção foi dedicada à um grupo de testes. Além disso, também foram avaliados aspectos da indexação e decisões do otimizador do banco de dados. A estrutura empregada nas seções foi a seguinte:

1) Parágrafo conciso descrevendo o teste;
2) Figura(s) da(s) query(ies) executada(s);
3) Figuras dos planos de execução para o 18c e para o TimesTen;
4) Gráfico comparativo com os resultados das execuções nos dois bancos de dados;
5) Breve discussão dos resultados do teste.

Cada uma das *queries* testadas foi executada por cinco vezes sem que houvesse a concorrência de outros SQLs de usuário. Os tempos de cada execução foram registrados, a média e a mediana entre as execuções calculadas e, finalmente, foi gerado o gráfico comparativo entre os dois SGDBs.

Não foram excluídos o maior e o menor valor coletados porque entende-se que em um ambiente real eles também estão presentes e impactam na performance do banco de dados. É notório, por exemplo, que a primeira execução de uma *query* em um banco de dados Oracle é, em geral, mais demorada do que as que a seguem. Isso ocorre porque o banco de dados realiza o *hard parsing* para determinar o plano de execução da *query* e esta é uma tarefa custosa. Entretanto, para as execuções posteriores, caso a declaração SQL ainda permaneça em memória, o Oracle reutiliza o código e faz apenas o que é chamado de *soft parsing*, processando a query de maneira mais rápida (ORACLE, 2013). Em um ambiente de produção, é comum que muitas *queries* não consigam tirar proveito desta situação, seja porque são executadas uma única vez, porque utilizam valores literais ao invés de *binds*, ou por alguma outra razão. Por isso, o entendimento de que é relevante para estes experimentos considerar o tempo da primeira execução, ainda que, em alguns casos, ele seja muito superior ao das execuções seguintes.

7.1. CONSULTAS CRIADAS PELO AUTOR

Aqui foram testadas uma série de consultas que são muito comuns a grande maioria dos bancos de dados relacionas, sejam eles OLTP ou OLAP. Essas *queries* tiveram como alvo principal a tabela AIH e isso se deve ao fato de que ela é a maior tabela no banco de dados COMP_TI, contendo cerca de 55 milhões de registros. As consultas não foram criadas, portanto, pensando na qualidade da informação extraída, mas sim, na carga que poderiam causar no banco de dados e no tempo que ele poderia levar para processá-las.

7.1.1. Consulta com filtro pela *primary key*

Operação típica que visa obter um único registro em uma tabela através de sua chave primária. Como utiliza o índice da PK, é geralmente muito rápida.

Figura 21 – Consulta por *Primary Key*

```
1  SELECT *
2  FROM COMP_TI.AIH
3  WHERE N_AIH=18965658;
```

Fonte: do autor, 2019

Figura 22 – Plano de execução no 18c – PK

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	3
TABLE ACCESS	COMP_TI.AIH	BY INDEX ROWID	1	3
INDEX	COMP_TI.PK_AIH	UNIQUE SCAN	1	2
Access Predicates				
N_AIH=18965658				
Other XML				

Fonte: do autor, 2019

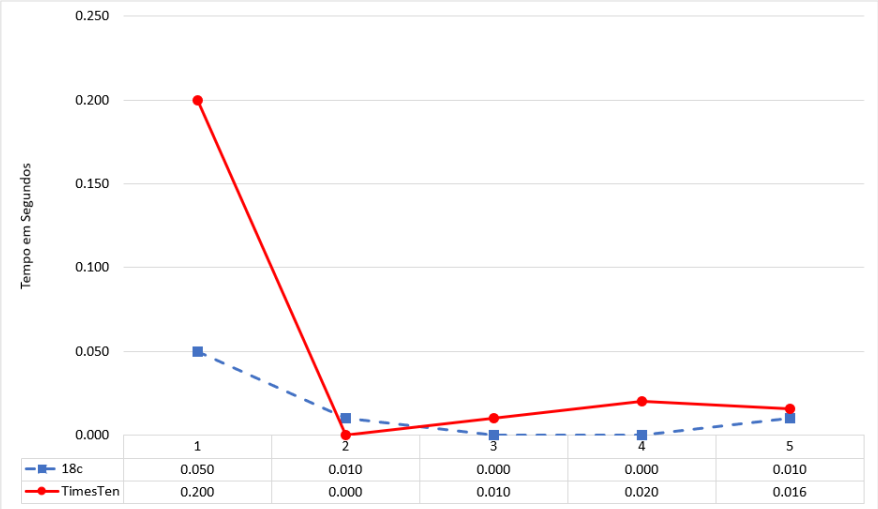
Figura 23 – Plano de execução no TimesTen – PK

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
RowLkRangeScan	COMP_TI.AIH	PK_AIH	AIH.N_AIH = 18965658	

Fonte: do autor, 2019

Figura 24 – Consulta por *Primary Key* – Resultados

	18c	TimesTen
1	0.050	0.200
2	0.010	0.000
3	0.000	0.010
4	0.000	0.020
5	0.010	0.016
Média	0.014	0.049
Mediana	0.010	0.016
Desvio Padrão	0.019	0.076



Fonte: do autor, 2019

Ao analisar a Figura 22, é possível perceber que o Oracle 18c não utilizou a tabela carregada em memória, mas, ao invés disso, optou por usar o índice da chave primária para acessar a tabela em disco. Este não era o comportamento esperado, já que o acesso à memória é mais rápido que ao disco físico. Por isso, presume-se que algo fez com que o otimizador do Oracle concluísse que, neste cenário, acessar o disco via índice seria mais rápido. E, corroborando com esta decisão, o 18c apresentou resultados melhores do que o TimesTen mesmo acessando a tabela em disco.

7.1.2. Consulta sem índice

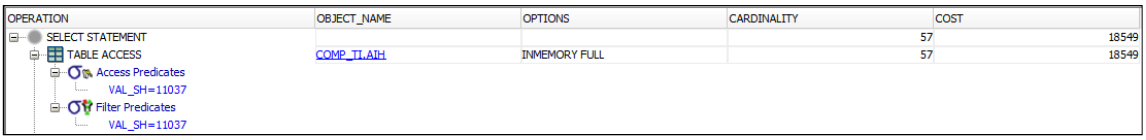
Esta é outra consulta que busca em uma tabela por um único registro. Entretanto, ao contrário da *query* da Figura 21, este *select* utiliza um predicado que não possui índice. Isso tende a provocar uma varredura completa da tabela e, consequentemente, um maior tempo de execução da consulta. Em muitos destes casos, ao se criar o índice a consulta obterá um ganho de desempenho considerável.

Figura 25 – Consulta sem índice

```
1  SELECT *
2  FROM COMP_TI.AIH
3  WHERE VAL_SH=11037;
```

Fonte: do autor, 2019

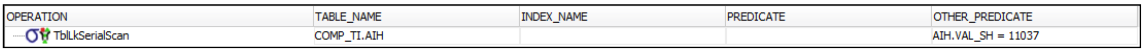
Figura 26 – Plano de execução Oracle 18c – SI



OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			57	18549
TABLE ACCESS	COMP_TI.AIH	INMEMORY FULL	57	18549
Access Predicates				
VAL_SH=11037				
Filter Predicates				
VAL_SH=11037				

Fonte: do autor, 2019

Figura 27 – Plano de execução TimesTen – SI

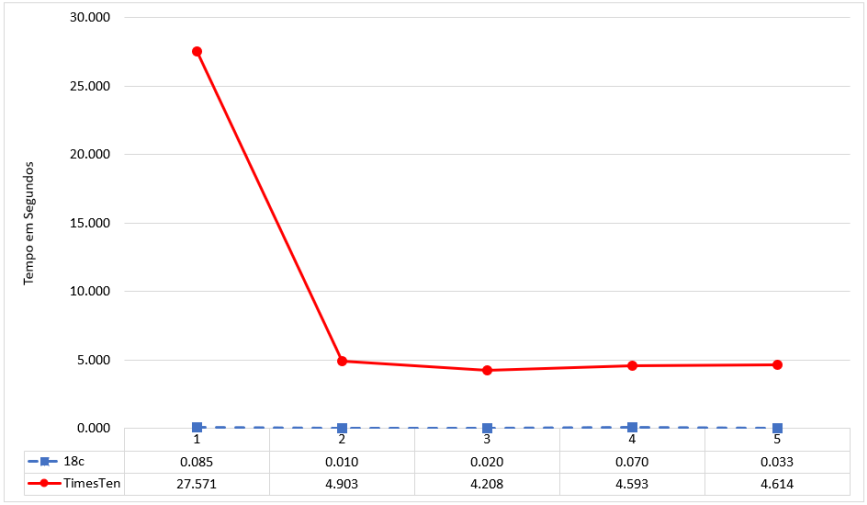


OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
TblLkSerialScan	COMP_TI.AIH			AIH.VAL_SH = 11037

Fonte: do autor, 2019

Figura 28 – Consulta sem índice – Resultados

	18c	TimesTen
1	0.085	27.571
2	0.010	4.903
3	0.020	4.208
4	0.070	4.593
5	0.033	4.614
Média	0.044	9.178
Mediana	0.033	4.614
Desvio Padrão	0.029	9.199



Fonte: do autor, 2019

Como era de se esperar, ambos os bancos de dados fizeram uma varredura completa na tabela e desta vez o 18c também acessou o objeto em memória. Porém, o que chama a atenção é que a média das execuções do 18c foi quase 205 vezes melhor do que a TimesTen.

7.1.3. Consulta com *full scan* e ordenação

A consulta da Figura 29 realiza um *full scan* e apresenta todos os registros da tabela AIH ordenados de maneira ascendente. A ordenação é uma operação custosa para o banco de dados.

Figura 29 – Consulta com *full scan* e ordenação

```
1  SELECT *
2  FROM COMP_TI.AIH
3  ORDER BY VAL_SH;
```

Fonte: do autor, 2019

Figura 30 – Plano de execução Oracle 18c – FS/Ordenação

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			54741842	1591232
SORT		ORDER BY	54741842	1591232
TABLE ACCESS	COMP_TI.AIH	INMEMORY FULL	54741842	18465

Fonte: do autor, 2019

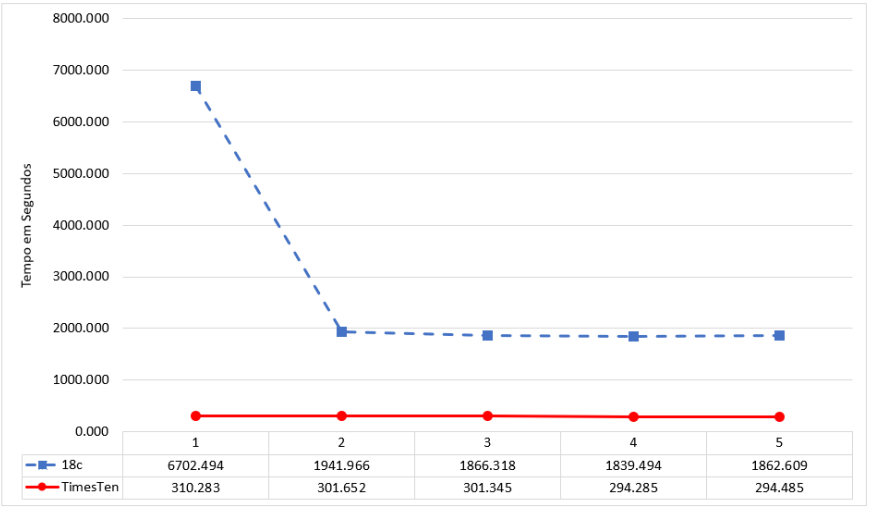
Figura 31 – Plano de execução TimesTen – FS/Ordenação

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
OrderBy				
TblLkRangeScan	COMP_TI.AIH	PK_AIH		

Fonte: do autor, 2019

Figura 32 – Consulta com *full scan* e ordenação – Resultados

	18c	TimesTen
1	6702.494	310.283
2	1941.966	301.652
3	1866.318	301.345
4	1839.494	294.285
5	1862.609	294.485
Média	2842.576	300.410
Mediana	1866.318	301.345
Desvio Padrão	1930.267	5.874



Fonte: do autor, 2019

Nesta consulta a surpresa negativa ficou por conta do Oracle 18c. A etapa de ordenação utilizou disco e foi necessário aumentar a *tablespace* temporária dos 6GB da instalação padrão para 32GB. Com muitos eventos do tipo *direct path write*, o *select full* com ordenação na COMP_TI.AIH, tabela que possui quase 55 milhões de registros, levou aproximadamente 1 hora e 52 minutos para finalizar sua primeira execução. No TimesTen este tempo foi de apenas 5 minutos e 10 segundos.

7.1.4. Consulta usando função de agrupamento (50 grupos)

O *group by*, ou agrupamento, é outra operação que tende a ser custosa para o banco de dados e neste teste se buscou agrupar por uma coluna que resultasse em muitos grupos.

Figura 33 – Consulta usando função de agrupamento (50 grupos)

```
1  SELECT COUNT(*)
2  FROM COMP_TI.AIH
3  GROUP BY VAL_SH;
```

Fonte: do autor, 2019

Figura 34 – Plano de execução Oracle 18c – *Group By* 50

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			2	13420
HASH		GROUP BY	2	13420
TABLE ACCESS	COMP_TI.AIH	INMEMORY FULL	54741842	13419

Fonte: do autor, 2019

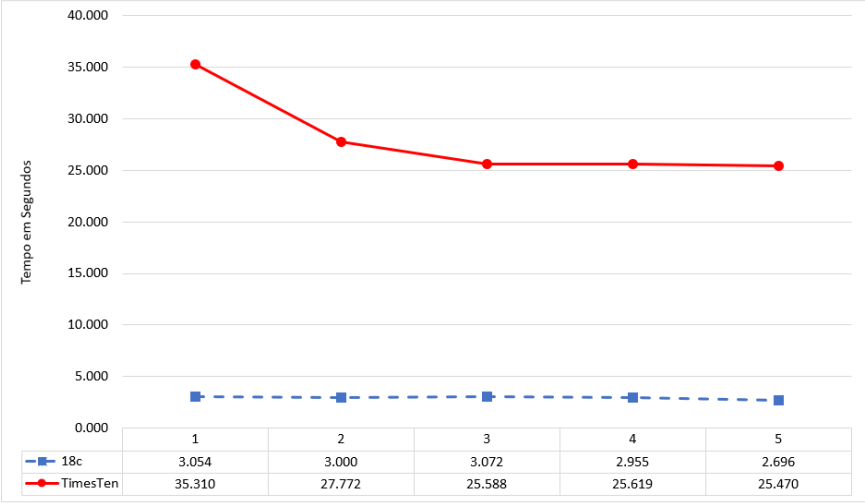
Figura 35 – Plano de execução TimesTen – *Group By* 50

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
GroupBy				
TblLkSerialScan	COMP_TI.AIH			

Fonte: do autor, 2019

Figura 36 – Consulta usando função de agrupamento (50 grupos) – Resultados

	18c	TimesTen
1	3.054	35.310
2	3.000	27.772
3	3.072	25.588
4	2.955	25.619
5	2.696	25.470
Média	2.955	27.952
Mediana	3.000	25.619
Desvio Padrão	0.136	3.778



Fonte: do autor, 2019

Para a consulta com agrupamento, mais uma vez o 18c apresentou tempos significativamente melhores que os do Timesten. Na média, o 18c chegou a ser nove vezes mais rápido.

7.1.5. Consulta usando função de agrupamento (2 grupos)

Aqui, do mesmo modo que na seção anterior, empregou-se o agrupamento. Porém, na consulta da Figura 37 apenas dois grupos serão retornados.

Figura 37 – Consulta usando função de agrupamento (2 grupos)

```
1  SELECT COUNT(*)
2  FROM COMP_TI.AIH
3  GROUP BY IND_MORTE;
```

Fonte: do autor, 2019

Figura 38 – Plano de execução Oracle 18c – 2 grupos

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			2	145400
HASH		GROUP BY	2	145400
TABLE ACCESS	COMP_TI.AIH	INMEMORY FULL	54741842	144805

Fonte: do autor, 2019

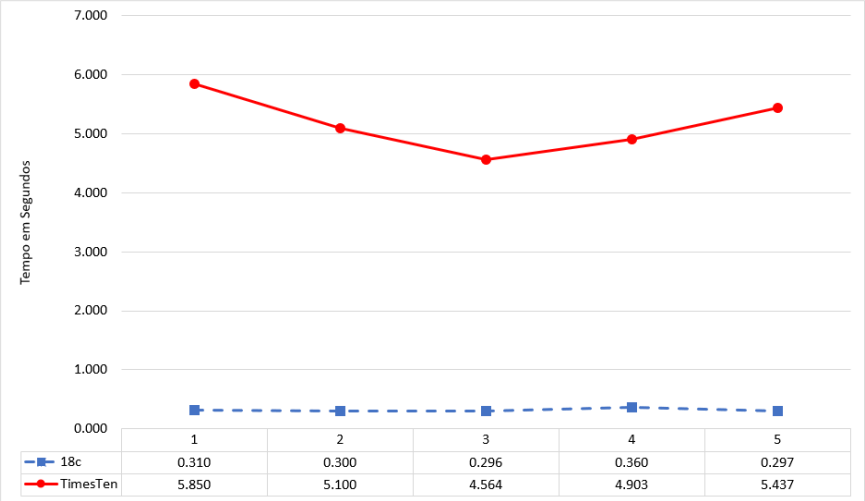
Figura 39 – Plano de execução TimesTen – 2 grupos

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
GroupBy				
TblLkSerialScan	COMP_TI.AIH			

Fonte: do autor, 2019

Figura 40 – Consulta usando função de agrupamento (2 grupos) – Resultados

	18c	TimesTen
1	0.310	5.850
2	0.300	5.100
3	0.296	4.564
4	0.360	4.903
5	0.297	5.437
Média	0.313	5.171
Mediana	0.300	5.100
Desvio Padrão	0.024	0.442



Fonte: do autor, 2019

Além dos tempos nos dois bancos de dados terem sido melhores com a formação de menos grupos, ficou evidente mais uma vez a superioridade do 18c para algumas consultas, chegando a ser 16,5 vezes mais rápido que o TimesTen em média.

7.1.6. Consulta usando uma *Foreign Key* sem índice

Outro tipo de *select* muito comum são aqueles que utilizam alguma chave estrangeira como predicado. Nestes casos, há situações onde a inexistência de um índice na tabela pai, referenciando a FK, pode causar graves problemas de performance no banco de dados. Este foi justamente o tipo de situação que se quis testar aqui.

Figura 41 – Consulta usando uma *Foreign Key* sem índice

```
1  SELECT *
2  FROM COMP_TI.MUNICIPIO M, COMP_TI.PACIENTE P
3  WHERE M.CO_MUNICIP = P.CO_MUNICIP
4    AND M.DS_NOME = 'Porto Alegre';
```

Fonte: do autor, 2019

Figura 42 – Plano de execução Oracle 18c – FK sem índice

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			2743	1030
HASH JOIN			2743	1030
Access Predicates				
M.CO_MUNICIP=P.CO_MUNICIP				
JOIN FILTER	:BF0000	CREATE	1	2
TABLE ACCESS	COMP_TI.MUNICIPIO	INMEMORY FULL	1	2
Access Predicates				
M.DS_NOME='Porto Alegre'				
Filter Predicates				
M.DS_NOME='Porto Alegre'				
JOIN FILTER	:BF0000	USE	12191575	998
TABLE ACCESS	COMP_TI.PACIENTE	INMEMORY FULL	12191575	998
Access Predicates				
SYS_OP_BLOOM_FILTER(:BF0000,P.CO_MUNICIP)				
Filter Predicates				
SYS_OP_BLOOM_FILTER(:BF0000,P.CO_MUNICIP)				

Fonte: do autor, 2019

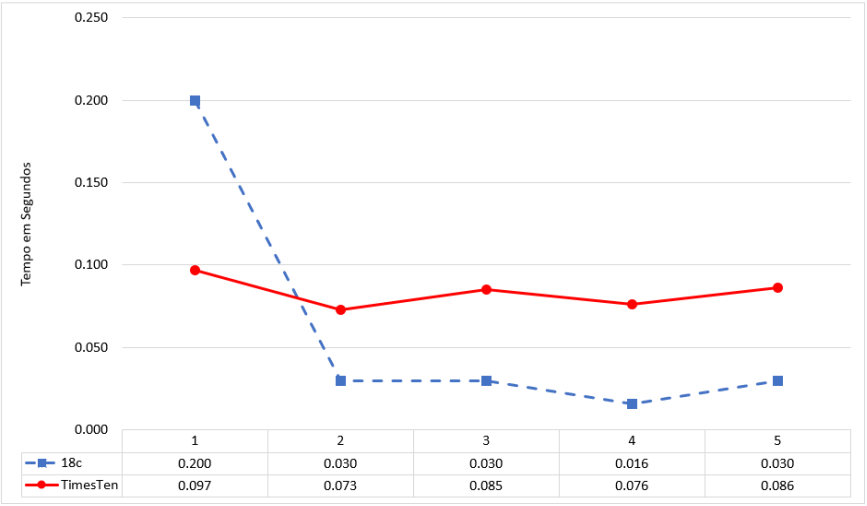
Figura 43 – Plano de execução TimesTen – FK sem índice

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
NestedLoop				
TmpRangeScan	COMP_TI.MUNICIPIO		M.DS_NOME = 'Porto Alegre'	
RowLkRangeScan	COMP_TI.PACIENTE	FK_PACIENTE_MUNIC	P.CO_MUNICIP = M.CO_MUNICIP	

Fonte: do autor, 2019

Figura 44 – Consulta usando uma *Foreign Key* sem índice – Resultados

	18c	TimesTen
1	0.200	0.097
2	0.030	0.073
3	0.030	0.085
4	0.016	0.076
5	0.030	0.086
Média	0.061	0.083
Mediana	0.030	0.085
Desvio Padrão	0.070	0.008



Fonte: do autor, 2019

O TimesTen criou automaticamente o índice da FK e não permitiu que ele fosse dropado justamente porque há uma chave estrangeira que o utiliza. Assim sendo, o teste não pode ser repetido da mesma maneira nos dois bancos. Todavia, os resultados demonstraram que o TimesTen foi mais lento mesmo tendo a provável vantagem de acessar os dados da tabela pai via índice. Outro fato interessante que pode ser observado neste teste é o de que o 18c utilizou um filtro *Bloom* para acessar a tabela pai (Figura 42). Filtros Bloom foram abordados na seção 6.4.2.2. e fazem parte da estratégia de *tuning* de *Join Groups* para o 18c.

7.1.7. Consulta usando uma *Foreign Key* com índice

Exatamente, o mesmo teste da seção 7.1.6, mas agora com o índice criado.

Figura 45 – Consulta usando uma *Foreign Key* com índice

```
1  CREATE INDEX COMP_TI.IX_PACIENTE_CO_MUNICIP ON COMP_TI.PACIENTE(CO_MUNICIP);
2
3  SELECT *
4  FROM COMP_TI.MUNICIPIO M, COMP_TI.PACIENTE P
5  WHERE M.CO_MUNICIP = P.CO_MUNICIP
6    AND M.DS_NOME = 'Porto Alegre';
7
8  DROP INDEX COMP_TI.IX_PACIENTE_CO_MUNICIP;
```

Fonte: do autor, 2019

Figura 46 – Plano de execução Oracle 18c – FK com índice

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			2743	1030
HASH JOIN			2743	1030
Access Predicates				
M.CO_MUNICIP=P.CO_MUNICIP				
JOIN FILTER	:BF0000	CREATE	1	2
TABLE ACCESS	COMP_TI.MUNICIPIO	INMEMORY FULL	1	2
Access Predicates				
M.DS_NOME='Porto Alegre'				
Filter Predicates				
M.DS_NOME='Porto Alegre'				
JOIN FILTER	:BF0000	USE	12191575	998
TABLE ACCESS	COMP_TI.PACIENTE	INMEMORY FULL	12191575	998
Access Predicates				
SYS_OP_BLOOM_FILTER(:BF0000,P.CO_MUNICIP)				
Filter Predicates				
SYS_OP_BLOOM_FILTER(:BF0000,P.CO_MUNICIP)				

Fonte: do autor, 2019

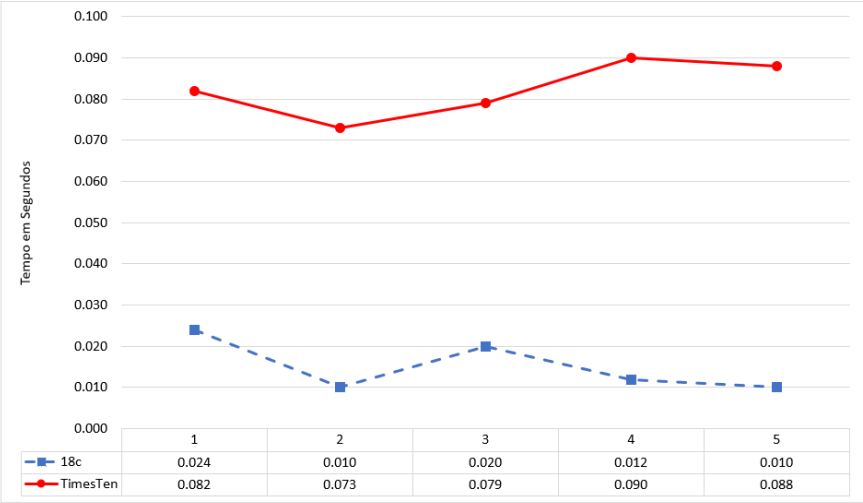
Figura 47 – Plano de execução TimesTen – FK com índice

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
NestedLoop				
TmpRangeScan	COMP_TI.MUNICIPIO		M.DS_NOME = 'Porto Alegre'	
RowLkRangeScan	COMP_TI.PACIENTE	FK_PACIENTE_MUNIC	P.CO_MUNICIP = M.CO_MUNICIP	

Fonte: do autor, 2019

Figura 48 – Consulta usando uma *Foreign Key* com índice – Resultados

	18c	TimesTen
1	0.024	0.082
2	0.010	0.073
3	0.020	0.079
4	0.012	0.090
5	0.010	0.088
Média	0.015	0.082
Mediana	0.012	0.082
Desvio Padrão	0.006	0.006



Fonte: do autor, 2019

O TimesTen não apresentou alterações, até mesmo porque ele já havia utilizado um índice no experimento da seção 6.1.6. O Oracle 18c, por outro lado, teve uma sensível melhora em sua performance e, embora o plano de execução tenha sido o mesmo utilizado na consulta da seção 7.1.6., ele foi cerca de quatro vezes mais rápido após a criação do índice na tabela pai.

7.2. CONSULTAS BASEADAS EM ORACLE, 2015

Este grupo de *queries* foi baseada nos SQLs que aparecem em Oracle, 2015, página 7, e mais uma vez, a tabela AIH é o objeto principal das consultas. À exemplo do que a Oracle faz em seu *white paper,* cada seção secundária irá comparar um par de *queries*, e a meta é constatar se de fato uma consulta se beneficia mais dos IMDBs do que a outra.

7.2.1. Número de colunas selecionadas

Segundo Oracle (2015), o aumento do número de colunas selecionada por uma query faz com que o custo de processamento por coluna passe a dominar o tempo de execução da query, reduzindo o benefício do IMDB.

Figura 49 – Consulta retornando todas as colunas

```
1  SELECT *
2  FROM COMP_TI.AIH;
```

Fonte: do autor, 2019

Figura 50 – Plano de execução Oracle 18c – Todas as colunas

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			54741842	18465
TABLE ACCESS	COMP_TI.AIH	INMEMORY FULL	54741842	18465

Fonte: do autor, 2019

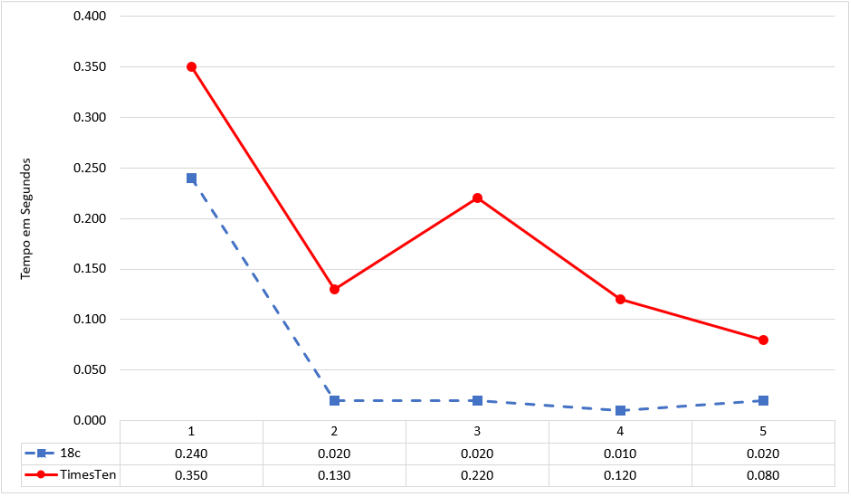
Figura 51 – Plano de execução TimesTen – Todas as colunas

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
TblLkRangeScan	COMP_TI.AIH	PK_AIH		

Fonte: do autor, 2019

Figura 52 – Número de colunas selecionadas – Todas as colunas

	18c	TimesTen
1	0.240	0.350
2	0.020	0.130
3	0.020	0.220
4	0.010	0.120
5	0.020	0.080
Média	0.062	0.180
Mediana	0.020	0.130
Desvio Padrão	0.089	0.097



Fonte: do autor, 2019

Para este SQL o Oracle 18c foi em média três vezes mais rápido que o TimesTen. Além disso, se comparadas apenas as suas execuções, é possível observar que os tempos da segunda à quinta execução foram cerca de doze vezes menores à primeira, o que reforça o benefício do *caching*.

Figura 53 – Consulta retornando apenas uma coluna

```
1  SELECT VAL_TOT
2  FROM COMP_TI.AIH;
```

Fonte: do autor, 2019

Figura 54 – Plano de execução Oracle 18c – Uma coluna

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			54741842	13419
TABLE ACCESS	COMP_TI.AIH	INMEMORY FULL	54741842	13419

Fonte: do autor, 2019

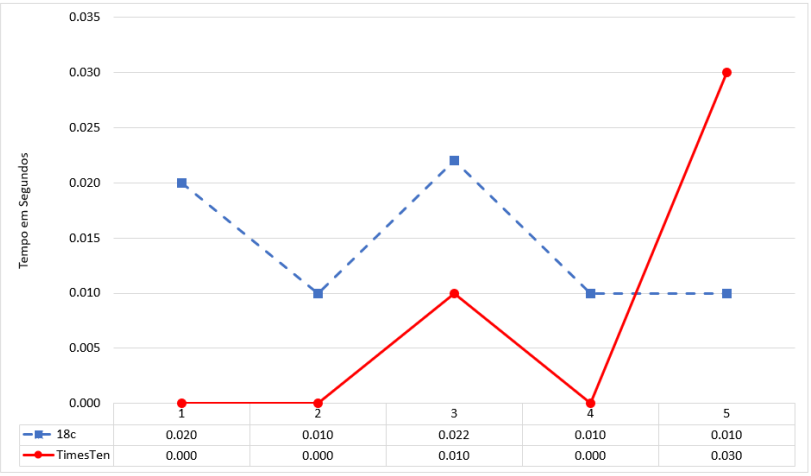
Figura 55 – Plano de execução TimesTen – Uma coluna

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
TblLkRangeScan	COMP_TI.AIH	PK_AIH		

Fonte: do autor, 2019

Figura 56 – Número de colunas selecionadas – Uma coluna

	18c	TimesTen
1	0.020	0.000
2	0.010	0.000
3	0.022	0.010
4	0.010	0.000
5	0.010	0.030
Média	0.014	0.008
Mediana	0.010	0.000
Desvio Padrão	0.005	0.012



Fonte: do autor, 2019

Este é um dos poucos casos em que a média de tempo do TimesTen sobrepujou a alcançada pelo Oracle 18c e, embora a diferença não seja tão significativa quanto em experimentos anteriores, o resultado acaba sendo importante, porque o que foi visto até aqui é um domínio do 18c no que tange à melhores tempos.

Os resultados dos dois testes reiteram o que afirma Oracle (2015) em relação ao número de colunas selecionadas, uma vez que os dois bancos de dados levaram mais tempo para exibir o resultado da *query* com maior número de colunas.

7.2.2. Quantidade de valores retornados

"Quanto maior o número de valores retornados por uma query, menor é o benefício do IM, porque o retorno dos dados para o cliente irá dominar os custos." (ORACLE, 2015, tradução nossa).

Figura 57 – Consulta retornando inúmeras linhas

```
1  SELECT VAL_TOT
2  FROM COMP_TI.AIH;
```

Fonte: do autor, 2019

Figura 58 – Plano de execução Oracle 18c – Inúmeras linhas

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			54741842	13419
TABLE ACCESS	COMP_TI.AIH	INMEMORY FULL	54741842	13419

Fonte: do autor, 2019

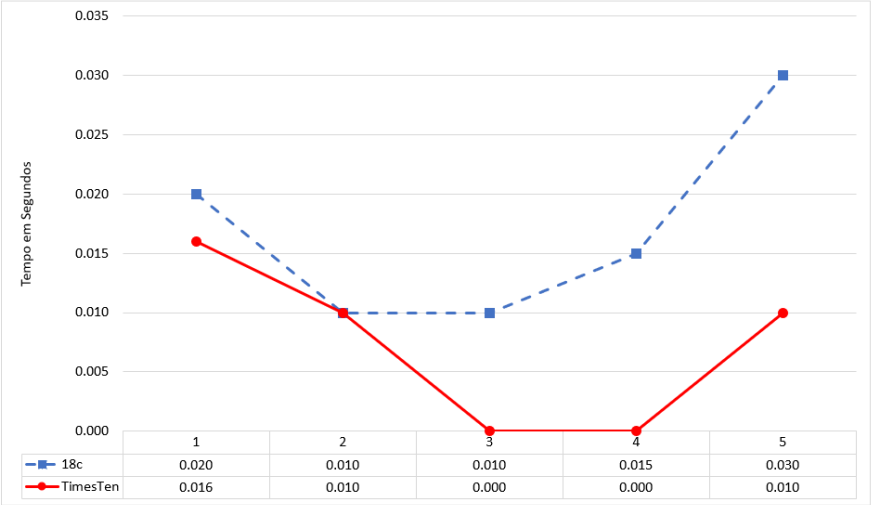
Figura 59 – Plano de execução TimesTen – Inúmeras linhas

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
TblLkRangeScan	COMP_TI.AIH	PK_AIH		

Fonte: do autor, 2019

Figura 60 – Quantidade de valores retornados – Inúmeras linhas

	18c	TimesTen
1	0.020	0.016
2	0.010	0.010
3	0.010	0.000
4	0.015	0.000
5	0.030	0.010
Média	0.017	0.007
Mediana	0.015	0.010
Desvio Padrão	0.007	0.006



Fonte: do autor, 2019

Este foi mais um teste em que o TimesTen conseguiu uma pequena vantagem em relação ao 18c.

Figura 61 – Consulta retornando um só valor

```
1  SELECT SUM(VAL_TOT)
2  FROM COMP_TI.AIH;
```

Fonte: do autor, 2019

Figura 62 – Plano de execução Oracle 18c – Um valor

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	13419
SORT		AGGREGATE	1	
TABLE ACCESS	COMP_TI.AIH	INMEMORY FULL	54741842	13419

Fonte: do autor, 2019

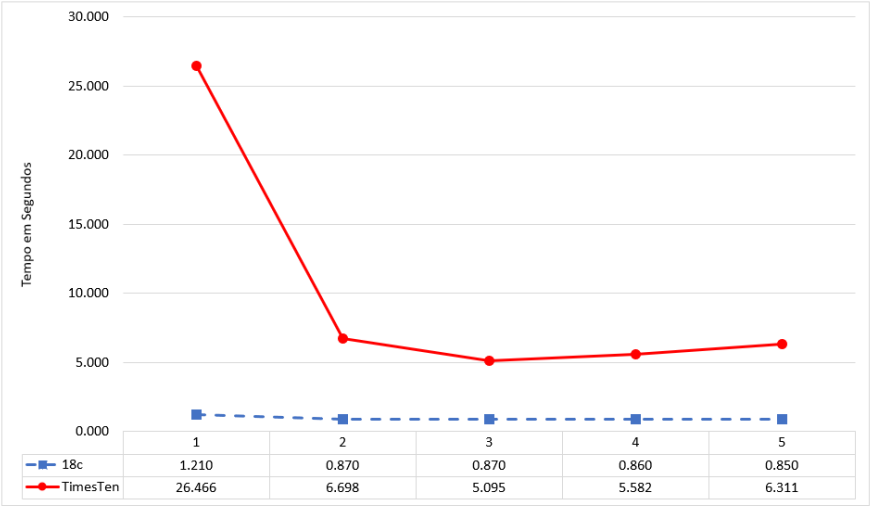
Figura 63 – Plano de execução TimesTen – Um valor

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
OneGroupGroupBy				
TblLkSerialScan	COMP_TI.AIH			

Fonte: do autor, 2019

Figura 64 – Quantidade de valores retornados – Um valor

	18c	TimesTen
1	1.210	26.466
2	0.870	6.698
3	0.870	5.095
4	0.860	5.582
5	0.850	6.311
Média	0.932	10.030
Mediana	0.870	6.311
Desvio Padrão	0.139	8.237



Fonte: do autor, 2019

Aqui o 18c voltou a apresentar tempos muito melhores que os alcançados pelo TimesTen.

Os resultados destes testes em ambos os IMDBs não comprovaram o que afirma a Oracle (2015), pois os tempos para a *query* que retornou apenas um valor acabaram sendo maiores nos dois casos.

7.2.3. Seletividade da coluna de predicado

“Uma coluna de predicado mais seletivo permite filtrar mais os resultados e reduz o volume de dados que as queries do plano intermediário precisam processar.

Além disso, predicados mais seletivos podem potencializar os índices armazenados em memória, acelerando ainda mais o acesso aos dados.”

Figura 65 – Consulta mais seletiva

```
1  SELECT MEDIAN(VAL_TOT)
2  FROM COMP_TI.AIH
3  WHERE VAL_TOT > 1000;
```

Fonte: do autor, 2019

Figura 66 – Plano de execução Oracle 18c – Mais seletiva

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	13504
SORT		GROUP BY	1	
TABLE ACCESS	COMP_TI.AIH	INMEMORY FULL	54516037	13504
Access Predicates				
VAL_TOT>1000				
Filter Predicates				
VAL_TOT>1000				

Fonte: do autor, 2019

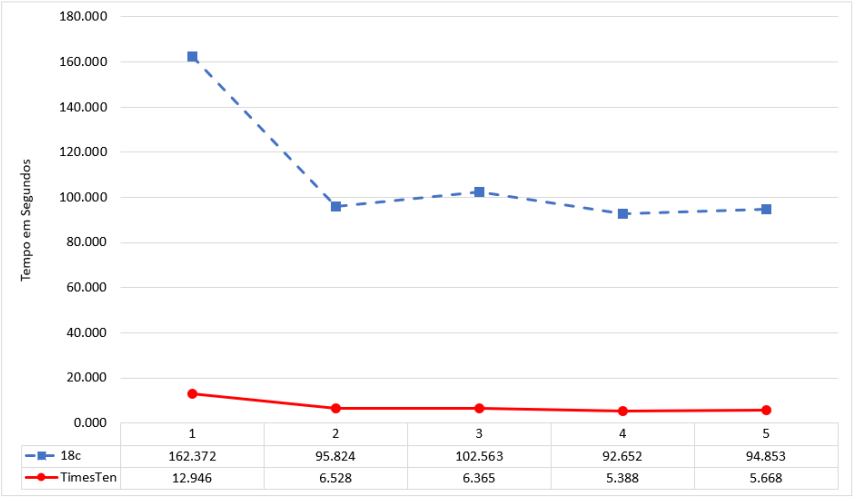
Figura 67 – Plano de execução TimesTen – Mais seletiva

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
OneGroupGroupBy				
TblLkSerialScan	COMP_TI.AIH			AIH.VAL_TOT > 1000

Fonte: do autor, 2019

Figura 68 – Seletividade da coluna de predicado – Mais seletiva

	18c	TimesTen
1	162.372	12.946
2	95.824	6.528
3	102.563	6.365
4	92.652	5.388
5	94.853	5.668
Média	109.653	7.379
Mediana	95.824	6.365
Desvio Padrão	26.566	2.816



Fonte: do autor, 2019

Para esta consulta os resultados do TimesTen foram melhores e com diferenças mais expressivas para o 18c, que teve um desempenho, em média, na casa de quatorze vezes pior.

Figura 69 – Consulta menos seletiva

```
1  SELECT MEDIAN(VAL_TOT)
2  FROM COMP_TI.AIH
3  WHERE VAL_TOT < 1000;
```

Fonte: do autor, 2019

Figura 70 – Plano de execução Oracle 18c – Menos seletiva

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			1	13504
SORT		GROUP BY	1	
TABLE ACCESS	COMP_TI.AIH	INMEMORY FULL	225805	13504
Access Predicates				
VAL_TOT<1000				
Filter Predicates				
VAL_TOT<1000				

Fonte: do autor, 2019

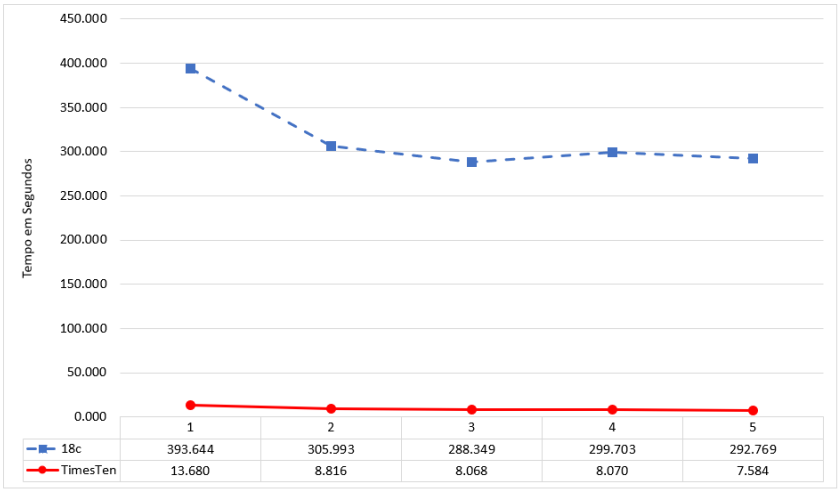
Figura 71 – Plano de execução TimesTen – Menos seletiva

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
OneGroupGroupBy				
TblLkSerialScan	COMP_TI.AIH			AIH.VAL_TOT < 1000

Fonte: do autor, 2019

Figura 72 – Seletividade da coluna de predicado – Menos seletiva

	18c	TimesTen
1	393.644	13.680
2	305.993	8.816
3	288.349	8.068
4	299.703	8.070
5	292.769	7.584
Média	316.092	9.244
Mediana	299.703	8.070
Desvio Padrão	39.239	2.253



Fonte: do autor, 2019

Mais uma vez o TimesTen foi superior, chegando a uma média de tempo trinta e cinco vezes melhor do que o Oracle 18c.

Os resultados destes testes corroboraram para o que declara a Oracle (2015). Os tempos para a query com um predicado menos seletivo foram realmente superiores, embora, no caso do TimesTen as diferenças não tenham sido tão grandes quanto para o 18c.

7.2.4. Seletividade das condições de *Join*

Para Oracle (2015), uma condição de *join* mais seletiva irá produzir um resultado menor e fazer com que menos dados tenham que ser processados pela query.

Figura 73 – *Join* menos seletivo

```
1  SELECT A.N_AIH, P.DT_NASC
2  FROM COMP_TI.AIH A, COMP_TI.PACIENTE P
3  WHERE A.CO_PACIENTE = P.CO_PACIENTE;
```

Fonte: do autor, 2019

Figura 74 – Plano de execução Oracle 18c – *Join* menos seletivo

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			54741842	91747
HASH JOIN			54741842	91747
Access Predicates				
A.CO_PACIENTE=P.CO_PACIENTE				
TABLE ACCESS	COMP_TI.PACIENTE	INMEMORY FULL	12191575	779
TABLE ACCESS	COMP_TI.AIH	INMEMORY FULL	54741842	13560

Fonte: do autor, 2019

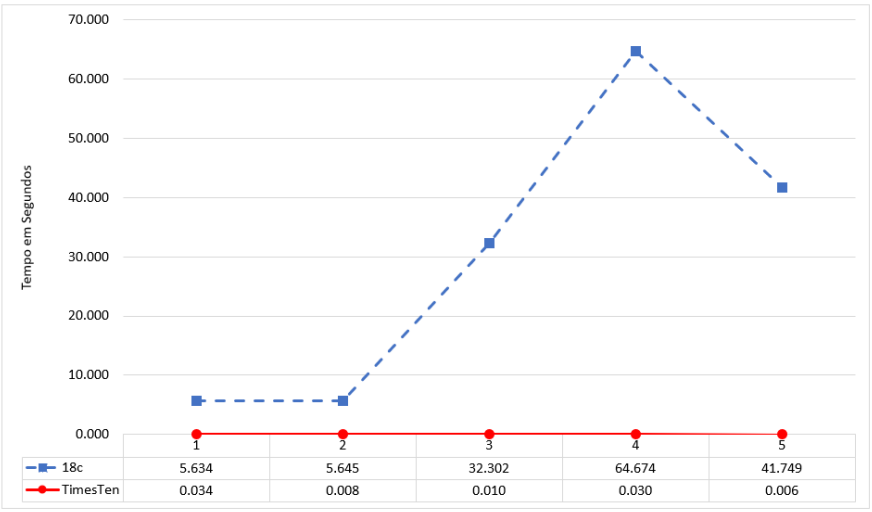
Figura 75 – Plano de execução TimesTen – *Join* menos seletivo

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
MergeJoin			P.CO_PACIENTE = A.CO_PACIENTE	
TblLkRangeScan	COMP_TI.PACIENTE	PK_PACIENTE		
TblLkRangeScan	COMP_TI.AIH	FK_AIH_PACIENTE	A.CO_PACIENTE >= P.CO_PACIENTE	

Fonte: do autor, 2019

Figura 76 – Seletividade das condições de *Join* – Menos seletivo

	18c	TimesTen
1	5.634	0.034
2	5.645	0.008
3	32.302	0.010
4	64.674	0.030
5	41.749	0.006
Média	30.001	0.018
Mediana	32.302	0.010
Desvio Padrão	22.506	0.012



Fonte: do autor, 2019

O TimesTen foi trinta vezes mais rápido, mas o que mais chamou a atenção aqui foram as diferenças entre os tempos obtidos pelo 18c. As duas primeiras execuções levaram 5,6 segundos, a terceira foi seis vezes mais lenta e a quarta chegou aos 64,6 segundos. A quinta e última baixou para 41,7 segundos. Este comportamento está longe do padrão para um ambiente que não está sofrendo a concorrência de outros processos. Além disso, durante estes testes não houve falta de recursos no sistema operacional e o consumo de memória e CPU não apresentaram comportamentos anormais.

Figura 77 – *Join* mais seletivo

```
1  SELECT A.N_AIH, P.DT_NASC
2  FROM COMP_TI.AIH A, COMP_TI.PACIENTE P
3  WHERE A.CO_PACIENTE = P.CO_PACIENTE AND NUM_FILHOS = 4;
```

Fonte: do autor, 2019

Figura 78 – Plano de execução Oracle 18c – *Join* mais seletivo

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			5001246	78326
HASH JOIN			5001246	78326
Access Predicates				
A.CO_PACIENTE=P.CO_PACIENTE				
JOIN FILTER	:BF0000	CREATE	1108325	863
TABLE ACCESS	COMP_TI.PACIENTE	INMEMORY FULL	1108325	863
Access Predicates				
NUM_FILHOS=4				
Filter Predicates				
NUM_FILHOS=4				
JOIN FILTER	:BF0000	USE	54741842	13560
TABLE ACCESS	COMP_TI.AIH	INMEMORY FULL	54741842	13560
Access Predicates				
SYS_OP_BLOOM_FILTER(:BF0000,A.CO_PACIENTE)				
Filter Predicates				
SYS_OP_BLOOM_FILTER(:BF0000,A.CO_PACIENTE)				

Fonte: do autor, 2019

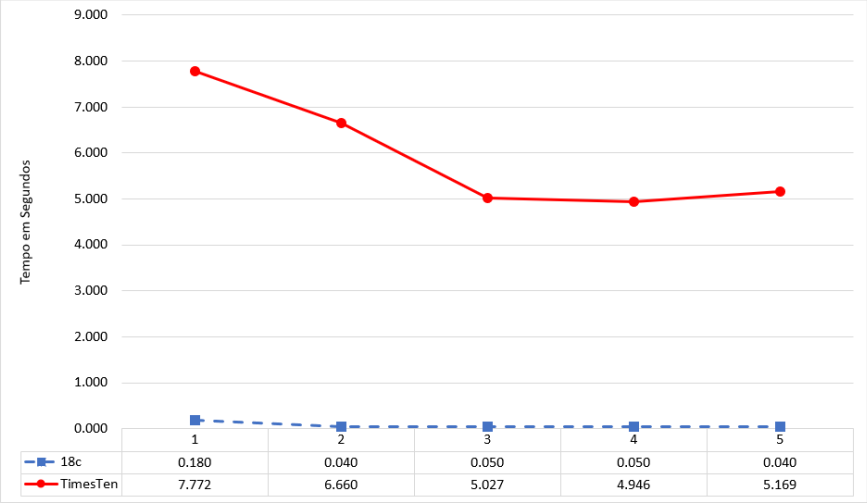
Figura 79 – Plano de execução TimesTen – *Join* mais seletivo

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
NestedLoop				
TmpRangeScan	COMP_TI.PACIENTE		P.NUM_FILHOS = 4	
TblLkRangeScan	COMP_TI.AIH	FK_AIH_PACIENTE	A.CO_PACIENTE = P.CO_PACIENTE	

Fonte: do autor, 2019

Figura 80 – Seletividade das condições de *Join* – Mais seletivo

	18c	TimesTen
1	0.180	7.772
2	0.040	6.660
3	0.050	5.027
4	0.050	4.946
5	0.040	5.169
Média	0.072	5.915
Mediana	0.050	5.169
Desvio Padrão	0.054	1.121



Fonte: do autor, 2019

Com o *join* mais seletivo, o 18c foi mais performático e voltou a se comportar como o esperado, tendo um tempo mais elevado para a primeira execução e tempos mais baixos para as demais.

O conjunto de resultados do Oracle 18c ratifica o que estabelece a Oracle (2015), tendo em vista que os tempos para a *query* com o *join* mais seletivo foram de fato muito melhores que os da outra consulta. Por outro lado, o desfecho destes testes no TimesTen foi justamente o oposto, isto é, a *query* com *join* menos seletivo executou mais rapidamente que a mais seletiva.

7.2.5. Quantidade de tabelas no *Join*

Quanto maior o número de tabelas em um *join*, maior a percentagem de tempo gasto no processamento do *join*. (ORACLE, 2015, tradução nossa)

Figura 81 – Mais tabelas no *Join*

```
1  SELECT A.N_AIH, P.DT_NASC, C.DS_CNES, I.DS_CID, S.DS_MOTIVO
2  FROM
3    COMP_TI.AIH A, COMP_TI.PACIENTE P,
4    COMP_TI.CNES C, COMP_TI.CID10 I, COMP_TI.SAIDA_PERM S
5  WHERE  A.CO_PACIENTE = P.CO_PACIENTE AND A.CO_CNES = C.CO_CNES AND
6         A.DIAG_PRINC = I.CO_CID AND A.CO_MOTIVO = S.CO_MOTIVO;
```

Fonte: do autor, 2019

Figura 82 – Plano de execução Oracle 18c – Mais tabelas no *Join*

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			54741842	128817
HASH JOIN			54741842	128817
Access Predicates				
A.DIAG_PRINC=I.CO_CID				
TABLE ACCESS	COMP_TI.CID10	INMEMORY FULL	14253	2
HASH JOIN			54741842	128675
Access Predicates				
A.CO_CNES=C.CO_CNES				
TABLE ACCESS	COMP_TI.CNES	INMEMORY FULL	2454	1
HASH JOIN			54741842	128534
Access Predicates				
A.CO_MOTIVO=S.CO_MOTIVO				
TABLE ACCESS	COMP_TI.SAIDA_PERM	INMEMORY FULL	28	3
HASH JOIN			54741842	128391
Access Predicates				
A.CO_PACIENTE=P.CO_PACIENTE				
TABLE ACCESS	COMP_TI.PACIENTE	INMEMORY FULL	12191575	779
TABLE ACCESS	COMP_TI.AIH	INMEMORY FULL	54741842	13980

Fonte: do autor, 2019

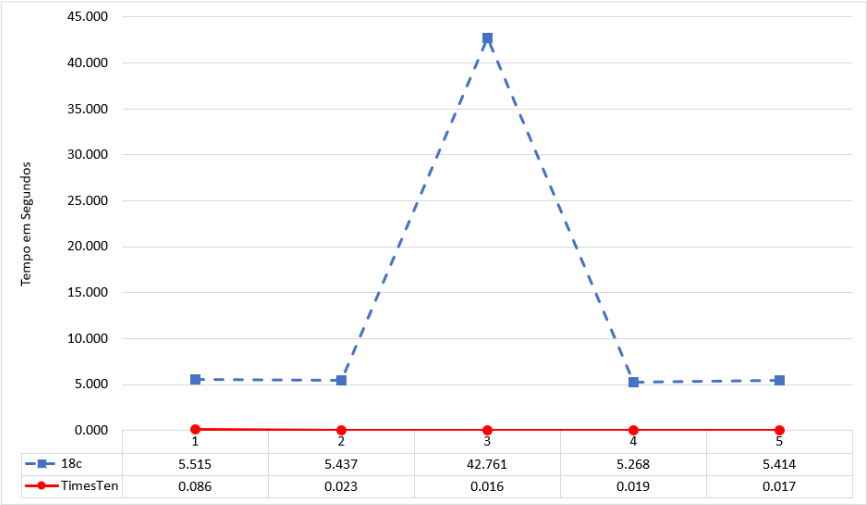
Figura 83 – Plano de execução TimesTen – Mais tabelas no *Join*

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
NestedLoop				
NestedLoop				
NestedLoop				
MergeJoin			I.CO_CID = A.DIAG_PRINC	
TblLkRangeScan	COMP_TI.CID10	PK_CID10		
TblLkRangeScan	COMP_TI.AIH	FK_AIH_DIAG_PRINC	A.DIAG_PRINC >= I.CO_CID	
TblLkRangeScan	COMP_TI.CNES	PK_CNES	C.CO_CNES = A.CO_CNES	
TblLkRangeScan	COMP_TI.PACIENTE	PK_PACIENTE	P.CO_PACIENTE = A.CO_PACIENTE	
TblLkRangeScan	COMP_TI.SAIDA_PERM	PK_SAIDA_PERM	S.CO_MOTIVO = A.CO_MOTIVO	

Fonte: do autor, 2019

Figura 84 – Quantidade de tabelas no *Join* – Mais tabelas

	18c	TimesTen
1	5.515	0.086
2	5.437	0.023
3	42.761	0.016
4	5.268	0.019
5	5.414	0.017
Média	12.879	0.032
Mediana	5.437	0.019
Desvio Padrão	14.941	0.027



Fonte: do autor, 2019

Aqui o TimesTen foi em média cerca de quatrocentas vezes mais rápido que o Oracle 18c. Além disso, em sua terceira execução o 18c apresentou um tempo aproximadamente oito vezes maior do que às suas demais rodadas. Isso ocorreu mesmo sem que houvesse uma alteração significativas no ambiente.

Figura 85 – Menos tabelas no *Join*

```
1  SELECT A.N_AIH, P.DT_NASC
2  FROM
3    COMP_TI.AIH A,
4    COMP_TI.PACIENTE P
5  WHERE
6    A.CO_PACIENTE = P.CO_PACIENTE;
```

Fonte: do autor, 2019

Figura 86 – Plano de execução Oracle 18c – Menos tabelas no *Join*

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
SELECT STATEMENT			54741842	91747
HASH JOIN			54741842	91747
Access Predicates				
A.CO_PACIENTE=P.CO_PACIENTE				
TABLE ACCESS	COMP_TI.PACIENTE	INMEMORY FULL	12191575	779
TABLE ACCESS	COMP_TI.AIH	INMEMORY FULL	54741842	13560

Fonte: do autor, 2019

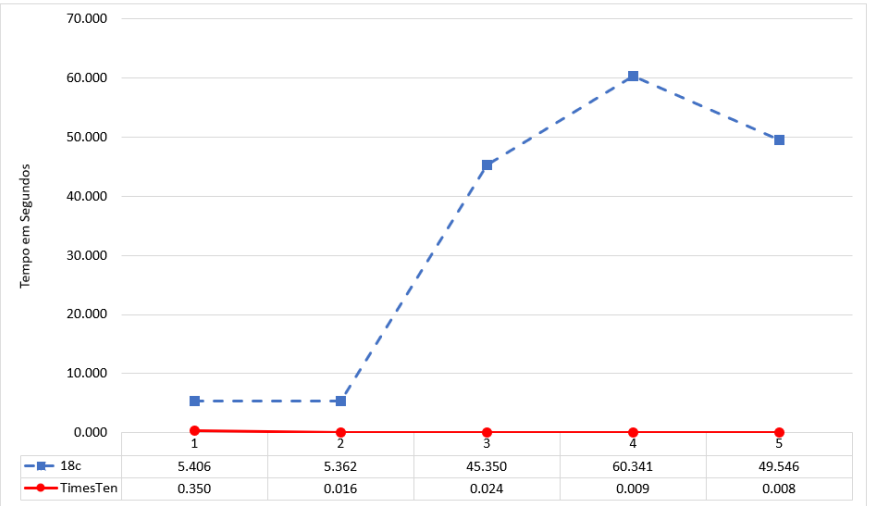
Figura 87 – Plano de execução TimesTen – Menos tabelas no *Join*

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
MergeJoin			P.CO_PACIENTE = A.CO_PACIENTE	
TblLkRangeScan	COMP_TI.PACIENTE	PK_PACIENTE		
TblLkRangeScan	COMP_TI.AIH	FK_AIH_PACIENTE	A.CO_PACIENTE >= P.CO_PACIENTE	

Fonte: do autor, 2019

Figura 88 – Quantidade de tabelas no *Join* – Menos tabelas

	18c	TimesTen
1	5.406	0.350
2	5.362	0.016
3	45.350	0.024
4	60.341	0.009
5	49.546	0.008
Média	33.201	0.081
Mediana	45.350	0.016
Desvio Padrão	23.233	0.134



Fonte: do autor, 2019

Mais uma vez o 18c teve tempos de execução que apresentaram um comportamento muito incomum. Suas duas primeiras execuções foram na casa dos 5

segundos, enquanto as demais passaram dos 45 segundos. O TimesTen, em contrapartida, manteve tempos muito bons.

Nestes experimentos os resultados não corroboraram com a tese de que o número de tabelas no *join* vá impactar na performance. O primeiro SQL possui junções com cinco tabelas, o segundo com apenas duas e, apesar disso, a query com piores tempos foi a segunda.

7.3. INSERÇÕES, ATUALIZAÇÕES E EXCLUSÕES

Neste grupo de experimentos foram executadas uma série de instruções DML, tendo como principal objetivo verificar qual dos dois bancos de dados apresenta melhor desempenho em operações que ocorrem com muita frequência em bases transacionais. Vale lembrar que as duas soluções aqui estudadas se propõem a unir o melhor de dois mundos, OLAP e OLTP, em um só banco de dados. Para cada um dos testes, foi gerado um lote de trinta mil instruções. À exemplo dos demais experimentos, cada lote foi executado por cinco vezes, os tempos foram registrados e a média e a mediana calculadas.

É importante ressaltar que o TimesTen utilizada por *default* a opção de *autocommit* e, embora a própria Oracle recomende que seja desabilitado, ele foi mantido porque o objetivo aqui é realizar os experimentos intervindo o mínimo possível nas configurações padrão dos dois bancos de dados. Conforme a Oracle (2018f), os *commits* podem custar caro para o desempenho e serem intrusivos se forem implicitamente executados após cada instrução. Logo, é possível, e até provável, que o *autocommit* tenha influenciado nos tempos alcançados pelo TimesTen neste grupo de experimentos.

7.3.1. Inserções

As inserções foram geradas a partir de registros que já existiam na base e, na prática, só tiveram alterados os campos N_AIH, ANO e MÊS. A Figura 89 traz a *query* que foi utilizada para a geração do lote de trinta mil *inserts*, que, como pode ser visto na Figura 90, são muito simples e não carecem de maiores explicações.

Figura 89– Query utilizada para criar a carga de inserções

```
SELECT
  'INSERT INTO COMP_TI.AIH (N_AIH, ANO,MES, CO_CNES, CO_PACIENTE, VAL_SH, VAL_SP, VAL_TOT)
  VALUES (COMP_TI.AIH_SEQ_II.NEXTVAL,2019,04,'
  ||CO_CNES||','||CO_PACIENTE||','||VAL_SH||','||VAL_SP||','||VAL_TOT||');'
FROM COMP_TI.AIH
WHERE N_AIH <= 30000;
```

Fonte: do autor, 2019

Figura 90 – Exemplo de *Insert*

```
INSERT INTO COMP_TI.AIH
  (N_AIH, ANO,MES, CO_CNES, CO_PACIENTE, VAL_SH, VAL_SP, VAL_TOT)
VALUES (COMP_TI.AIH_SEQ_II.NEXTVAL,2019,04,2223589,622737,1155.69,157.79,1313.48);
```

Fonte: do autor, 2019

Figura 91 – Plano de execução Oracle 18c – Inserções (*Inserts*)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
INSERT STATEMENT			1	1
LOAD TABLE CONVENTIONAL	COMP_TI.AIH			
SEQUENCE	COMP_TI.AIH_SEQ_II			

Fonte: do autor, 2019

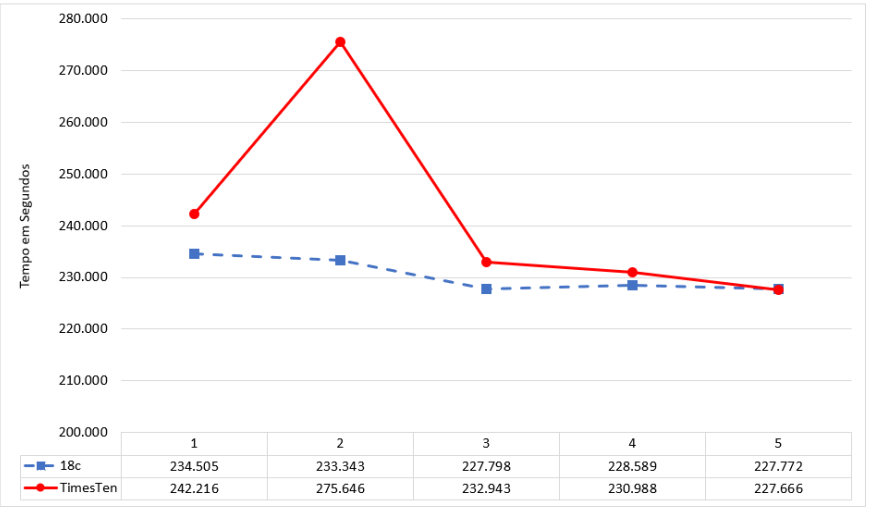
Figura 92 – Plano de execução TimesTen – Inserções (*Inserts*)

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
RowLkInsert	COMP_TI.AIH			

Fonte: do autor, 2019

Figura 93 – Resultados dos *Inserts*

	18c	TimesTen
1	234.505	242.216
2	233.343	275.646
3	227.798	232.943
4	228.589	230.988
5	227.772	227.666
Média	230.401	241.892
Mediana	228.589	232.943
Desvio Padrão	2.914	17.554



Fonte: do autor, 2019

Os tempos das inserções formam ligeiramente melhores com o Oracle 18c, mas de modo geral, apenas a segunda execução com o TimesTen distou um pouco das demais.

7.3.2. Atualizações

A Figura 94 exibe o *select* criado para a geração da massa de *updates*. Uma a uma, foram realizadas trinta mil atualizações da coluna US_TOT, tabela AIH, de seu valor atual para 1000.00, conforme pode ser visto na Figura 95.

Figura 94– Query utilizada para criar a carga de atualizações

```
1  SELECT 'UPDATE COMP_TI.AIH SET US_TOT = 1000.00 WHERE N_AIH = '||N_AIH||';'
2  FROM COMP_TI.AIH WHERE N_AIH <= 30000;
```

Fonte: do autor, 2019

Figura 95 – Exemplo de *Update*

```
1  UPDATE COMP_TI.AIH
2  SET US_TOT = 1000.00
3  WHERE N_AIH = 150001;
```

Fonte: do autor, 2019

Figura 96 – Plano de execução Oracle 18c – Atualizações (*Updates*)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
UPDATE STATEMENT			1	3
UPDATE	COMP_TI.AIH			
INDEX	COMP_TI.PK_AIH	UNIQUE SCAN	1	2
Access Predicates				
N_AIH=150001				

Fonte: do autor, 2019

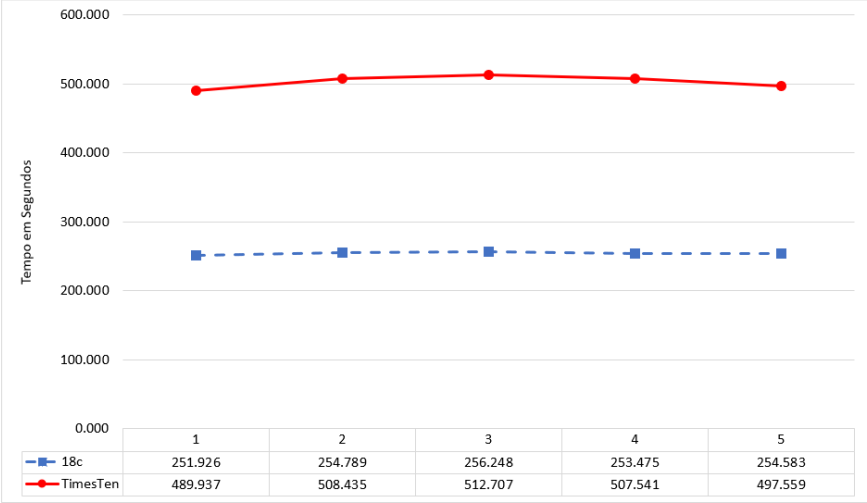
Figura 97 – Plano de execução TimesTen – Atualizações (*Updates*)

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
SQL Developer Plan's root				
RowLkRangeScan	COMP_TI.AIH	PK_AIH	AIH.N_AIH = 150001	
RowLkUpdate	COMP_TI.AIH			

Fonte: do autor, 2019

Figura 98 – Resultados dos *Updates*

	18c	TimesTen
1	251.926	489.937
2	254.789	508.435
3	256.248	512.707
4	253.475	507.541
5	254.583	497.559
Média	254.204	503.236
Mediana	254.583	507.541
Desvio Padrão	1.441	8.299



Fonte: do autor, 2019

Os *updates* do Oracle 18c levaram praticamente a metade do tempo das execuções no TimesTen e, como visto nas Figuras 96 e 97, ambos utilizaram o índice da PK para acessar o registro a ser atualizado, portanto, o resultado não deixa de ser surpreendente. Além disso, com base no plano de execução do 18c (Figura 96), ele não acessou a tabela em memória, mas em disco. Quando ele utiliza o objeto em memória a coluna *Options* do plano de execução exibe o valor INMEMORY FULL, como na Figura 86, por exemplo. Este não foi o caso aqui.

7.3.3. Exclusões

A Figura 99 exibe o *select* usado para que fossem criadas as trinta mil exclusões de registros na tabela AIH e um exemplo de um destes *deletes*.

Figura 99 – Query utilizada para criar a carga de exclusões

```
1  SELECT 'DELETE FROM COMP_TI.AIH WHERE N_AIH = '||N_AIH||';'
2  FROM COMP_TI.AIH
3  WHERE N_AIH > 150000;
```

Fonte: do autor, 2019

Figura 100 – Exemplo de *Delete*

```
1  DELETE FROM COMP_TI.AIH
2  WHERE N_AIH = 150001;
```

Fonte: do autor, 2019

Figura 101 – Plano de execução Oracle 18c – Exclusões (*Deletes*)

OPERATION	OBJECT_NAME	OPTIONS	CARDINALITY	COST
DELETE STATEMENT			1	2
DELETE	COMP_TI.AIH			
INDEX	COMP_TI.PK_AIH	UNIQUE SCAN	1	2
Access Predicates				
N_AIH=150001				

Fonte: do autor, 2019

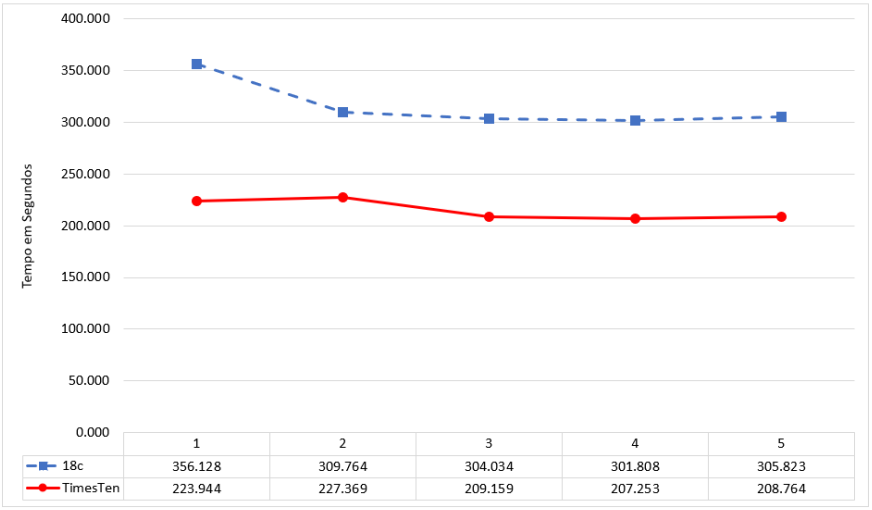
Figura 102 – Plano de execução TimesTen – Exclusões (*Deletes*)

OPERATION	TABLE_NAME	INDEX_NAME	PREDICATE	OTHER_PREDICATE
SQL Developer Plan's root				
RowLkRangeScan	COMP_TI.AIH	PK_AIH	AIH.N_AIH = 150001	
RowLkDelete	COMP_TI.AIH			

Fonte: do autor, 2019

Figura 103 – Resultados dos *Deletes*

	18c	TimesTen
1	356.128	223.944
2	309.764	227.369
3	304.034	209.159
4	301.808	207.253
5	305.823	208.764
Média	315.511	215.298
Mediana	305.823	209.159
Desvio Padrão	20.475	8.551



Fonte: do autor, 2019

Nas exclusões foi a vez do TimesTen ter um desempenho melhor. Ele realizou os deletes, em média, em 68% do tempo necessário para o Oracle 18c realizar a mesma operação. Como já havia ocorrido com os *updates*, os dois bancos utilizaram o índice da PK para acessar o registro a ser removido e o 18c não acessou a tabela na *IM Column Store*.

7.4. ANÁLISE DOS RESULTADOS

Esta seção é destinada a uma rápida análise dos resultados, vislumbrando um panorama geral dos experimentos, já que os resultados específicos foram comentados nas seções dedicadas aos próprios testes. Cada uma das linhas do Quadro 7 representa um dos testes executados. As células marcadas com um X simbolizam o banco de dados que teve o melhor tempo médio ao executar a instrução. Em contrapartida, os números correspondem a quantas vezes o outro banco foi mais lento. Por exemplo, no experimento 7.1.1 o Oracle 18c com *Database In-Memory* foi mais rápido que o TimesTen, o qual foi em média 3,51 vezes mais lento.

Quadro 7 – Quem foi mais rápido?

EXPERIMENTO	ORACLE 18c	TIMESTEN
7.1.1	X	3,51
7.1.2	X	210,50
7.1.3	9,46	X
7.1.4	X	9,46
7.1.5	X	16,54
7.1.6	X	1,36
7.1.7	X	5,42
7.2.1 a	X	2,90
7.2.1 b	1,80	X
7.2.2 a	2,36	X
7.2.2 b	X	10,76
7.2.3 a	14,86	X
7.2.3 b	34,20	X
7.2.4 a	1704,59	X
7.2.4 b	X	82,15

(continua)

(continuação)

EXPERIMENTO	ORACLE 18c	TIMESTEN
7.2.5 a	399,97	X
7.2.5 b	407,87	X
7.3.1	X	1,05
7.3.2	X	1,98
7.3.3	1,47	X

Fonte: do autor, 2019

O Oracle 18c obteve tempos melhores em praticamente todos os testes do primeiro grupo, exceto pelo experimento 3 (seção 7.1.3. Consulta com *full scan* e ordenação). Já a partir dos testes 7.2.3 (Seletividade da coluna de predicado), no segundo grupo de experimentos, percebe-se que o TimesTen passa a superar o 18c em vários testes e, mais do que isso, a ter tempos significativamente melhores.

Com base nestes resultados a impressão é de que o 18c obteve tempos melhores naquelas instruções mais simples e mais frequentes para a maioria dos bancos de dados. Por outro lado, o TimesTen passou a ter tempos melhores nas instruções mais complexas ou naquelas com características mais analíticas.

Estes testes não apresentam resultados definitivos e, como não houve a supremacia de nenhuma das duas arquiteturas, não é correto afirmar que haja o que se poderia chamar de um “vencedor”. Na verdade, o propósito desde o princípio era o de identificar, além da performance, características e peculiaridades na execução de instruções SQL por estas duas arquiteturas.

Avaliando os resultados coletados nos experimentos de performance, somados aos testes e análises teóricas que foram realizados no capítulo 6, o que se pode dizer é que o Oracle 18c, utilizando o recurso *Database In-Memory*, é o mais indicado para bancos de dados OLTP que tenham algumas características OLAP. Já o TimesTen, parece ser mais robusto e capaz de atender melhor demandas de ambientes predominantemente analíticos.

8. CONCLUSÃO

Os sistemas gerenciadores de bancos de dados baseados em disco têm encontrado enormes dificuldades para suplantar os obstáculos provenientes da crescente necessidade de processamento de dados em tempo real e do aumento exponencial no volume de dados produzidos diariamente. Garantir a performance e melhorar o acesso e a manipulação desta massa de dados, são apenas alguns dos problemas a serem enfrentados.

É neste cenário que os IMDBs vêm ganhando cada vez mais espaço como uma solução que pode proporcionar o tão almejado processamento em tempo real, ao suplantar um dos principais gargalos dos SGDBs convencionais: a escrita e a leitura em disco. A redução de custos de alguns componentes de hardware e o avanço das inovações tecnológicas, têm possibilitado o armazenamento de uma grande quantidade de dados na memória principal, bem como, têm permitido o surgimento de novas arquiteturas de banco de dados que se propõem a entregar desempenho para sistemas OLTP e OLAP em uma solução única.

O objetivo geral desta monografia foi atingido, já que se realizou o estudo comparativo entre duas soluções de bancos de dados em memória: uma arquitetura híbrida, com o Oracle 18c utilizando a funcionalidade *In-Memory,* e outra tipicamente em memória, com o Oracle TimesTen. Isto foi alcançado não apenas através de comparações teóricas, mas, também, de inúmeros experimentos práticos. Experimentos, aliás, que foram realizados de maneira mais intensa para medir a performance, mas que não ficaram restritos à este aspecto. Ainda que em menor escala, também ocorreram testes para avaliar questões relacionadas à indexação, *tuning*, durabilidade e tamanho de objetos em memória. Temas estes, que ainda foram objeto de uma extensa pesquisa teórica onde sempre se buscou colocar lado a lado as duas tecnologias estudadas.

No tocante à desempenho, o que se percebe é o *Database In-Memory* sendo em geral mais rápido que o TimesTen. Isso é notório especialmente na primeira parte dos experimentos. Em contrapartida, nos testes em que o 18c obteve um tempo ruim, ele foi drasticamente pior que o TimesTen. Em síntese, os testes apontam que o *Database In-Memory* é indicado para aquelas instruções mais comuns em um banco

de dados. De outra parte, o TimesTen se sai melhor quando as consultas são predominantemente OLAP.

O estudo revelou ainda que o 18c possui um índice específico para a *IM Column Store*, o *In-Memory Storage Index,* e que o TimesTen cria automaticamente os índices de chaves estrangeiras. Quanto ao *tuning* de SQL, percebeu-se que a utilização de "*Joins* com filtros *Bloom"* pode de fato beneficiar muito as consultas no *Database In-Memory,* pois em um dos testes ele teve tempos melhores que o TimesTen, mesmo este tendo utilizado um índice. Sobre a durabilidade, comportaram-se como esperado. Finalmente, no que corresponde ao tamanho de objetos em memória, o trabalho demonstrou que o TimesTen pode ocupar um espaço muito superior a um SGDB Oracle convencional e isto deve ser fortemente considerado. Nestes experimentos, o TimesTen chegou a dobrar o tamanho do banco de dados, se comparado ao Oracle 18c. A compactação máxima indica que ele possa reduzir para algo em torno de 55% de seu espaço total, mas, para tanto, o ideal seria a realização de mais testes, inclusive, avaliando a performance neste novo cenário. O Oracle 18c, por sua vez, usando sua compressão padrão, reduziu o tamanho dos objetos quando carregados em memória, otimizando a ocupação de espaço.

A ideia de que TimesTen e Oracle 18c utilizando *Database In-Memory* aparecem como duas ótimas opções para enfrentar os atuais desafios do processamento e armazenamento de dados foi confirmada nesta monografia.

O TimesTen apresentou tempos muito bons quando os experimentos tinham características OLAP e tempos razoáveis em transações mais simples. Além disso, seus resultados foram mais "lineares", isto é, não foram percebidas grandes discrepâncias como ocorreram em alguns testes com o *Database In-Memory*. Algo que não chega a ser negativo, mas que precisa ser considerado, é que a migração para o TimesTen não é tão trivial como seria para o *Database In-Memory*. Além disso, existem algumas pequenas diferenças no suporte a PL/SQL, por exemplo, e, é claro, que por tratar-se de uma arquitetura totalmente nova, mesmo que o administrador de banco de dados domine outras versões do Oracle, haverá um *gap* de conhecimento.

O *Database In-Memory*, por sua vez, surge como uma excelente alternativa para bancos de dados que se encaixam no perfil mais transacional, mas que tenham algumas características e necessidades de um sistema OLAP. Para quem pretende migrar de um banco de dados Oracle em disco, a transição é ainda mais tranquila.

Outro aspecto interessante é que as tabelas que serão carregadas em memória são eleitas. Ou seja, não é necessário carregar o banco de dados inteiro na memória, como no TimesTen. Desta forma, a transição pode ocorrer de forma gradual e caso os resultados não sejam percebidos, é simples voltar a utilizar o SGDB da maneira tradicional, baseado em disco.

Dentre as principais dificuldades encontradas ao longo da pesquisa, estão a abrangência do tema, as limitações de recurso do ambiente de testes e a ausência ou escassez de material teórico à respeito de alguns pontos do trabalho. O tema foi definido de forma a contemplar diferentes tópicos relacionados aos IMDBs. Naturalmente, isto fez com que complexidade da pesquisa aumentasse e o prazo para a realização do trabalho não fosse suficiente para a realização de mais testes. Algo mais, as configurações de hardware do servidor utilizado nos testes também foram um delimitador, pois em se tratando de experimentos envolvendo IMDBs, o ideal seria que houvesse mais memória RAM. Por fim, em alguns pontos da pesquisa, como na questão do tamanho dos objetos em memória, foi realmente difícil encontrar material que abordasse o assunto com mais aprofundamento. Além disso, a grande maioria do material consultado é disponibilizado pelo próprio fabricante e, em alguns casos, sente-se a falta de um outro ponto de vista.

Trabalhos futuros poderiam aprofundar os estudos sobre alguns pontos abordados nesta monografia. É perfeitamente possível, por exemplo, dedicar uma pesquisa às razões pelas quais o TimesTen ocupa mais espaço que o Oracle 18c e buscar soluções que possam reduzir o tamanho deste banco de dados. Ainda neste sentido, a compressão e a conversão de tipos poderiam ser analisadas do ponto de vista da redução de espaço e em relação a performance. Outro tema interessante seria a realização de testes de indexação, mas agora com o objetivo de identificar se de fato os IMDBs podem reduzir o tamanho e a quantidade de índices em relação a um SGDB tradicional. O *tuning* de instruções SQL no *Database In-Memory*, com simulações fazendo uso de *IM Expressions*, *Join Groups*, *IM Aggregation* e a otimização e repovoamento da *IM Column Store,* seria mais uma possibilidade. Enfim, quando o assunto é “banco de dados em memória”, há ainda um vasto universo de pesquisa a ser explorado.

REFERÊNCIAS BIBLIOGRÁFICAS

EMC Digital Universe with Research & Analysis by IDC, 2014. **The digital universe of opportunities:** Rich Data and the Increasing Value of the Internet of Things. Acessado em: 17/03/2018. Disponível em: <https://www.emc.com/leadership/digital-universe/2014iview/executive-summary.htm> Acesso em: 17 mar. 2018.

GARCIA-MOLINA, Hector; SALEM, Kenneth. **Main memory database systems:** An Overview, IEEE Transactions on Knowledge and Data Engineering, Vol. 4, No. 6, pp. 509-516, Dec. 1992.

GERHARDT, Tatiana E.; SILVEIRA, Denise T.. **Métodos de pesquisa.** Porto Alegre: Editora UFRGS, 2009. Disponível em: <http://www.ufrgs.br/cursopgdr/downloadsSerie/derad005.pdf> Acesso em: 22 mar. 2018.

GUPTA, Mohit K.; VERMA, Vishal; VERMA, Megha S.. **In-memory database systems** – A Paradigm Shift, Int. J. Eng. Trends Technol., vol. 6, no. 6, 2013.

HORN, Cristiano. **Análise técnica e aplicação de banco de dados em memória.** 2016. Trabalho de conclusão de curso (Monografia) – Curso de Bacharel em Sistemas da Informação, Universidade Feevale, Novo Hamburgo, RS, 2016. Disponível em: <https://tconline.feevale.br/NOVO/tc/files/0002_4045.doc> Acesso em: 17 mar. 2018.

ITL Education Solutions, 2010. **Introduction to database systems.** Delhi: Pearson, 2010.

KYLE, Tom. 2015. **On Oracle database in-memory**. Disponível em: <https://blogs.oracle.com/oraclemagazine/on-oracle-database-in-memory> Acesso em: 23 mai. 2018.

LEGATTI, Eduardo; FURUSHIMA, Carlos H. Y. 2015. **Performance, Diagnostic e Tuning**: Asynchronous Commit. Disponível em: <https://www.oracle.com/technetwork/pt/articles/database-performance/asynchronous-commit-2488891-ptb.html?printOnly=1> Acesso em: 31 mai. 2019.

Microsoft. **Control transaction durability**. 2016.Disponível em: <https://docs.microsoft.com/pt-br/sql/relational-databases/logs/control-transaction-durability?view=sql-server-2017> Acesso em: 19 mai. 2018.

NASCIMENTO, Rodrigo. **Afinal, o que é big data?** 2017. Disponível em: <http://marketingpordados.com/analise-de-dados/o-que-e-big-data-%F0%9F%A4%96/> Acesso em: 24 mar. 2018.

O'REILLY MEDIA, 2012. **Big data now – 2012 edition.** Sebastopol: Editora O'Reilly, Inc. 2012. Disponível em: <http://www.oreilly.com/data/free/files/big-data-now-2012.pdf> Acesso em: 24 mar. 2018.

Oracle, 2009. **Oracle TimesTen in-memory database:** SQL reference – release 11.2.1, Part Number E13070-05. Disponível em: <https://oracle.su/docs/11g/timesten.112/e13070/ttsql373.htm> Acesso em: 06 abr. 2019.

Oracle, 2011. **Oracle In-Memory Database Cache - Introduction**. Disponível em: <https://docs.oracle.com/cd/E13085_01/timesten.1121/e14261/overview.htm#TTCIN119> Acesso em: 31 mai. 2019.

Oracle, 2013. **Database SQL Tuning Guide**. Disponível em: <https://docs.oracle.com/database/121/TGSQL/tgsql_sqlproc.htm#TGSQL175> Acesso em: 22 mai. 2018.

Oracle, 2015. **When to use Oracle database in-memory identifying** – Use Cases for Application Acceleration. Acessado em: 22/05/2018. Disponível em: <http://www.oracle.com/technetwork/database/in-memory/overview/twp-dbim-usage-2441076.html> Acesso em: 22 mai. 2018.

Oracle, 2017a. **Oracle database in-memory implementation and usage.** Disponível em: <http://www.oracle.com/technetwork/database/in-memory/learnmore/twp-oracle-dbim-implementation-3863029.pdf> Acesso em: 26 mai. 2018.

Oracle, 2017b. **Oracle Database Online Documentation Library, 12c Release 1 (12.1.0.2).** Disponível em: <https://docs.oracle.com/database/121/REFRN/GUID-1CD070FD-2864-452D-B2AA-50E977C4885A.htm#REFRN30739> Acesso em: 06 mai. 2019.

Oracle, 2018a. **Oracle database 12c in-memory option.** Disponível em: <http://www.oracle.com/us/solutions/sap/nl23-db12c-imo-en-2209396.pdf> Acesso em: 17 mar. 2018.

Oracle, 2018b. **Oracle TimesTen in-memory database:** Introduction, Release 18.1. Disponível em: <https://docs.oracle.com/database/timesten-18.1/TTCIN/TTCIN.pdf> Acesso em: 19 mai. 2018.

Oracle, 2018c. **Oracle TimesTen in-memory database:** SQL Reference, Release 18.1. Disponível em: <https://docs.oracle.com/database/timesten-18.1/TTSQL/TTSQL.pdf> Acesso em: 11 mai. 2019.

Oracle, 2018d. **Oracle database – database in-memory guide.** Acessado em: 17 mar. 2018. Disponível em: <https://docs.oracle.com/en/database/oracle/oracle-database/12.2/inmem/database-memory-guide.pdf>

Oracle, 2018e. **Oracle database in-memory with Oracle database 18c – technical overview.** Disponível em: <www.oracle.com/technetwork/database/in-memory/overview/twp-oracle-database-in-memory-2245633.pdf> Acesso em: 17 mar. 2018.

Oracle, 2018f. **Oracle TimesTen in-memory database:** Operations Guide 11g Release 2 (11.2.2). Disponível em: <https://docs.oracle.com/cd/E11882_01/timesten.112/e21633.pdf> Acesso em: 18 mai. 2019.

Oracle, 2018g. **The ttImportFromOracle utility.** Disponível em: < https://download.oracle.com/otn_hosted_doc/timesten/1122/support/ttImportFromOracle.pdf> Acesso em: 18 mai. 2019.

Oracle Developer Community, 2012. **Questions after first TimesTen test:** Query performance and memory footprint. Disponível em: <https://community.oracle.com/message/10565531#10563531> Acesso em: 11 mai. 2019.

PINTO, Guilherme L.. **Um estudo comparativo entre banco de dados relacional em disco e em memória.** 2016. Trabalho de conclusão de curso (Monografia) – Curso de Bacharel em Tecnologias de Informação e Comunicação, Universidade Federal de Santa Catariba, Araranguá, SC, 2016. Disponível em: < http://docplayer.com.br/48638615-Guilherme-lucon-pinto-um-estudo-comparativo-entre-banco-de-dados-relacional-em-disco-e-em-memoria.html> Acesso em: 24 mar. 2018.

PRADO, Fábio, 2014. **O que é tuning?** Disponível em: <http://www.fabioprado.net/2014/04/o-que-e-tuning-como-tunar.html> Acesso em: 09 jun. 2018.

PRODANOV, Cleber Cristiano; FREITAS; Ernani Cesar, **Metodologia do trabalho científico**: Métodos e Técnicas da Pesquisa e do Trabalho Acadêmico, 2ª edição. Novo Hamburgo, RS, 2013.

SIGALAEV, Gennady. **Oracle TimesTen and others Oracle Technologies:** TimesTen and Memory allocation. 2012. Disponível em: < http://ggsig.blogspot.com/2012/01> Acesso em: 19 mai. 2019.

SILVA, Marcell G.. **Banco de dados em memória (In-Memory)**. 2013. Trabalho de conclusão de curso (Monografia) – Curso de Bacharel em Ciência da Computação, Fundação Educacional do Município de Assis – FEMA, Assis, PR, 2013. Disponível em: <https://cepein.femanet.com.br/BDigital/arqTccs/0711271057.pdf> Acesso em: 17 mar. 2018.

STÖRL, Uta. 2017. **Big data technologies:** In-Memory DBMS. Disponível em: <http://www.fbi.h-da.de/fileadmin/personal/u.stoerl/BigData-SoSe17/BigData-SoSe17-4-InMemory.pdf> Acesso em: 13 mai. 2018.

UAB/UFRGS, 2009. **Métodos de pesquisa.** Disponível em: <http://www.ufrgs.br/cursopgdr/downloadsSerie/derad005.pdf> Acesso em: 22 mar. 2018.

VARGAS, Rafael C.. **Análise comparativa do SGDB SQL Server baseado em disco e em memória.** 2017. Trabalho de conclusão de curso (Monografia) – Curso de Bacharel em Sistemas da Informação, Universidade Feevale, Novo Hamburgo, RS, 2017. Disponível em: <https://tconline.feevale.br/NOVO/tc/files/0002_4045.doc> Acesso em: 17 mar. 2018.

YU, William E.. **Computação in-memory** – Evolução, oportunidades e riscos. ISACA JOURNAL. Vol. 5, 2013. Disponível em: <https://www.isaca.org/Journal/archives/2013/Volume-5/Documents/In-memory-Computing-Evolution-Opportunity-and-Risk_jrn_Portuguese_0913.pdf> Acesso em: 26 mar. 2018.