

André Cid Ferrizzi

*Uma abordagem lógica para o
gerenciamento de identificadores de objetos
em sistemas gerenciadores de banco de
dados não convencionais*

São José do Rio Preto - SP, Brasil

21 de maio de 2010

André Cid Ferrizzi

*Uma abordagem lógica para o
gerenciamento de identificadores de objetos
em sistemas gerenciadores de banco de
dados não convencionais*

Dissertação apresentada para a obtenção do título de Mestre em Ciência da Computação, área de Banco de Dados, junto ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto.

Orientador:

Prof. Dr. Carlos Roberto Valêncio

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO
DEPARTAMENTO DE CIÊNCIAS DE COMPUTAÇÃO E ESTATÍSTICA
UNIVERSIDADE ESTADUAL PAULISTA "JÚLIO DE MESQUITA FILHO"

São José do Rio Preto - SP, Brasil

21 de maio de 2010

André Cid Ferrizzi

*Uma abordagem lógica para o
gerenciamento de identificadores de objetos
em sistemas gerenciadores de banco de
dados não convencionais*

Dissertação apresentada para a obtenção do título de Mestre em Ciência da Computação, área de Banco de Dados, junto ao Programa de Pós-Graduação em Ciência da Computação do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", Campus de São José do Rio Preto

BANCA EXAMINADORA

Prof. Dr. Carlos Roberto Valêncio
UNESP - São José do Rio Preto
Orientador

Profa. Dra. Rogéria Cristiane Gratão de Souza
UNESP - São José do Rio Preto

Profa. Dra. Marilde Terezinha Prado Santos
Universidade Federal de São Carlos

São José do Rio Preto, 21 de maio de 2010

Enquanto o homem massacrar os animais, irão matar uns aos outros. Aquele que semeia a semente da matança e da dor não pode colher alegria e amor.

Pitágoras.

Dedico este trabalho à Josi, aos meus pais, João e Iara, e ao meus quatro irmãos em ordem crescente de chegada: Vítor, João, Lucas e Hugo.

Agradecimentos

Ao meu amor, Josi.

Aos meus pais, João e Iara, e aos meus irmãos Vítor, João, Lucas e Hugo.

Ao professor Valêncio, pela amizade e orientação. Sou muito grato pelo crescimento pessoal e profissional ao longo dos anos de trabalho juntos.

A todos os professores do departamento, bem como alguns que no momento não estão mais nele. Cada um teve grande importância na minha formação e sempre serei grato por isso.

Aos meus colegas e amigos Leandro, Geraldo, Álvaro, Willian Lima, Jorge, Antonio, Bruno, Juliano, Thiago, Toni, Marcelo, José Nelson, Evandro e a todo o pessoal do Grupo de Banco de Dados.

A todos aqueles que contribuíram para minha formação e que, por infeliz esquecimento de minha parte, não foram mencionados aqui.

Resumo

Os Sistemas Gerenciadores de Banco de Dados Não Convencionais são utilizados por aplicações que necessitam de eficiência no gerenciamento de objetos complexos. Um dos conceitos fundamentais nestes sistemas é o de identidade de objetos, pois em uma base de dados cada objeto possui um identificador único que é utilizado para acessá-lo e referenciá-lo em relacionamentos com outros objetos. A implementação de identidade pode ser feita com OIDs físicos ou OIDs lógicos. A abordagem física apresenta o problema de fragmentação da base de dados, pois os OIDs são formados diretamente pelos endereços dos objetos. Já a abordagem lógica não tem este problema, e as técnicas são árvore-B, *hashing* e mapeamento direto. Cada uma destas abordagens apresenta um determinado problema: árvore-B pode ocupar muita memória e o tempo de mapeamento possui complexidade logarítmica; em *hashing* ocorrem colisões pois o conhecimento prévio do tamanho da tabela *hash* se torna inviável em base de dados, que crescem de maneira imprevisível; e por último, mapeamento direto, que apesar de possuir o menor tempo de mapeamento dentre as três abordagens, não permite a relocação de todas as páginas da base de dados. Uma outra abordagem lógica é utilizada no Núcleo Gerenciador de Dados Multimídia (NUGEM), o qual vem sendo desenvolvido junto ao Grupo de Banco de Dados do IBILCE de São José do Rio Preto com o intuito de gerenciar dados não convencionais. Neste trabalho é proposta uma nova estrutura e funcionalidades para a técnica de gerenciamento de OIDs, cuja experimentação foi efetivada junto ao NUGEM, caracterizando uma nova abordagem com um menor tempo de manipulação dos OIDs, a qual pode ser nomeada de mapeamento indireto. É também avaliado o esgotamento de OIDs, comprovando-se que este não representa um problema para a abordagem de mapeamento indireto. Em comparação com as abordagens da literatura, a abordagem de mapeamento indireto possui tempo de mapeamento constante, utiliza menos memória do que a abordagem de árvore-B, não possui problemas de colisões como em *hashing* e, ao contrário de mapeamento direto, permite a total flexibilidade na relocação das páginas em disco.

Abstract

Non-Conventional Database Management Systems are used for applications that require efficient management of complex objects. One fundamental concept of these systems is object identity, because in a database each object has a unique identifier that is used to access it and reference it in relationships with another objects. The implementation of identity can be made with physical OIDs or logical OIDs. The physical approach presents the problem of fragmenting the database, because the OIDs are formed directly by addresses of objects. The logical approach does not have this problem, and the techniques are B-tree, hashing and direct mapping. Each of these approaches presents a particular problem: B-tree can occupy a lot of memory and its mapping time is logarithmic, in hashing collisions occur because the prior knowledge of the table size becomes a problem in databases, which grow in unpredictable ways, and finally, direct mapping, which despite having the smallest mapping time of the three approaches, does not allow the relocation of all pages in the database. Another logical approach is used in the Multimedia Data Manager Kernel (NUGEM), which is being developed by the Database Group at IBILCE, São José do Rio Preto, São Paulo, Brazil, in order to manage non-conventional data. This work proposes a new structure and features for the OID management, which was tested in NUGEM, featuring a new approach with a shorter handling time of OIDs, that can be named indirect mapping. It is also evaluated the exhaustion of OIDs, proving that this is not a problem for the indirect mapping approach. Compared with the approaches of the literature, the approach of indirect mapping has constant mapping time, uses less memory than the B-tree approach, has no problems like collisions and, unlike direct mapping, allows total flexibility for relocating pages in the database.

Sumário

Lista de Figuras

Lista de Abreviaturas

1	Introdução	p. 1
1.1	Considerações iniciais	p. 1
1.2	Motivação e escopo	p. 2
1.3	Objetivos	p. 3
1.4	Organização dos capítulos	p. 4
2	Fundamentação teórica	p. 5
2.1	Considerações iniciais	p. 5
2.2	O conceito de identidade	p. 5
2.3	O modelo de dados ODMG 3.0	p. 7
2.4	Servidor de dados e servidor de nomes	p. 8
2.5	OIDs físicos	p. 10
2.6	OIDs lógicos	p. 11
2.6.1	<i>Hashing</i>	p. 12
2.6.2	Árvore-B	p. 14

2.6.3	Mapeamento direto	p. 15
2.6.4	Esgotamento de OIDs	p. 18
2.7	Núcleo Gerenciador de Dados Multimídia	p. 18
2.7.1	Subsistema de Gerenciamento de Registros Reais	p. 20
2.7.2	Subsistema de Gerenciamento de Registros Lógicos	p. 26
2.8	Considerações finais	p. 31
3	Desenvolvimento do trabalho	p. 32
3.1	Considerações iniciais	p. 32
3.2	Subsistema de Gerenciamento de Registros Reais	p. 33
3.2.1	Tabela de Conversão Temporária e Tabela de Conversão de Registros Reais, Físicos, <i>Overflow</i> e OIDs	p. 33
3.2.2	Funcionalidades para o gerenciamento da Tabela de Conversão Temporária	p. 35
3.2.3	Otimização das listas do Subsistema de Gerenciamento de Registros Reais	p. 39
3.2.4	Linearidade no crescimento da base de dados	p. 40
3.3	Subsistema de Gerenciamento de Registros Lógicos	p. 41
3.3.1	Atribuição do máximo contOID e espalhamento dos contOIDs	p. 41
3.3.2	Avaliação do esgotamento de OIDs na abordagem de mapeamento indireto	p. 41
3.4	Comparação entre mapeamento indireto, mapeamento direto, <i>hashing</i> e árvore-B	p. 43
3.5	Considerações finais	p. 46

4	Testes e resultados	p. 47
4.1	Considerações iniciais	p. 47
4.2	Aplicativos de testes	p. 47
4.3	Subsistema de Gerenciamento de Registros Reais	p. 50
4.3.1	Listas de recuperação de registros reais e físicos	p. 51
4.3.2	Listas de registros reais requisitados, lidos, escritos e liberados de uma transação	p. 52
4.3.3	Tabela de Conversão Temporária	p. 53
4.4	Subsistema de Gerenciamento de Registros Lógicos	p. 56
4.4.1	Lista de recuperação de registros lógicos	p. 57
4.4.2	Tempos utilizados para a busca do máximo contOID e espalha- mento de contOIDs	p. 58
4.5	Avaliação do tempo de mapeamento e utilização de memória	p. 59
4.5.1	Testes com páginas de tamanho 1KB	p. 60
4.5.2	Testes com páginas de tamanho 4 KB	p. 62
4.6	Considerações finais	p. 65
5	Conclusões e trabalhos futuros	p. 66
	Referências	p. 70

Lista de Figuras

1	Exemplo de relacionamento no modelo ODMG (FERRIZZI, 2006).	p. 8
2	Cliente se comunica com o servidor de nomes e com o servidor de dados (EICKLER; GERLHOF; KOSSMANN, 1995).	p. 9
3	Cliente se comunica apenas com o servidor de nomes (EICKLER; GER- LHOF; KOSSMANN, 1995).	p. 9
4	Cliente se comunica com o servidor de dados (EICKLER; GERLHOF; KOS- SMANN, 1995).	p. 10
5	Mapeamento ilustrando as páginas de ponteiros e de dados.	p. 12
6	Exemplo com mapeamento direto (EICKLER; GERLHOF; KOSSMANN, 1995).	p. 16
7	Arquitetura do NUGEM (FERRIZZI, 2006).	p. 19
8	Tabela de Conversão de Registros Reais, Físicos, <i>Overflow</i> e OIDs.	p. 21
9	TCREFOID após a criação de um registro real.	p. 23
10	TCREFOID após a leitura de um registro real.	p. 24
11	TCREFOID após a escrita de um registro real.	p. 24
12	TCREFOID após a remoção de um registro real.	p. 25
13	Representação dos registros reais e lógicos no NUGEM.	p. 27
14	Representação da estrutura de um bloco lógico.	p. 27
15	Fórmula utilizada para o cálculo de um endereço lógico.	p. 28

16	Mapeamento de um OID para a localização física de um registro lógico do tipo objeto.	p. 29
17	Espalhamento do contadorSalvaOID.	p. 30
18	Representação de um bloco lógico durante a remoção de um registro lógico do tipo objeto.	p. 30
19	Atribuição do máximo contOID.	p. 31
20	Tabela de conversão temporária.	p. 34
21	Gerenciamento de OIDs na TCREFOID por meio da coluna CSOID. . .	p. 35
22	Redução do número de operações de escrita e leitura em disco da operação de criar uma nova página de OIDs e de dados.	p. 37
23	Número de registros lógicos do tipo objeto por tamanho de página. . . .	p. 41
24	Tempo de esgotamento de um OID na abordagem de mapeamento indireto.	p. 42
25	Características das abordagens de <i>hashing</i> , árvore-B, mapeamento direto e mapeamento indireto.	p. 45
26	Aplicativo utilizado para a criação das bases de dados (OLIVEIRA, 2006).	p. 48
27	Aplicativo desenvolvido para a realização de testes na SGReal.	p. 49
28	Aplicativo desenvolvido para a realização de testes na SGLog.	p. 50
29	Resultados dos testes com as listas de recuperação da SGReal.	p. 51
30	Gráfico do percentual de tempo ganho com a nova implementação para as listas de recuperação.	p. 52
31	Resultados dos testes com as listas de registros reais requisitados, lidos, escritos, liberados de uma transação.	p. 52

32	Gráfico do percentual de tempo ganho com a nova implementação para as listas de registros reais requisitados, lidos, escritos e liberados de uma transação.	p. 53
33	Redução da operação de escrita na funcionalidade de criar novos registros reais.	p. 54
34	Redução da operação de leitura na funcionalidade de remover registro real.	p. 55
35	Resultados dos testes utilizando a TCTEMP.	p. 55
36	Gráfico do percentual de tempo ganho utilizando a TCTEMP.	p. 56
37	Resultados dos testes com lista linear e árvore vermelho-preto para a lista de recuperação de registros lógicos.	p. 57
38	Gráfico do percentual de tempo ganho utilizando árvore vermelho-preto na SGLog.	p. 58
39	Medição dos tempos utilizados para a busca do máximo contOID e espalhamento de contOIDs.	p. 58
40	Porcentagem do tempo utilizado para as operações de espalhamento e busca pelo máximo contOID.	p. 59
41	Tempo para as operações e mapeamento com páginas de 1 KB.	p. 60
42	Tempo para o mapeamento com páginas de 1 KB.	p. 61
43	Tempo de gravação e inicialização da coluna CSOID com páginas de 1 KB.	p. 61
44	Tempo total de gerenciamento dos 7,7 milhões de OIDs.	p. 62
45	Utilização de memória principal para páginas de 1 KB.	p. 62
46	Tempo para as operações e mapeamento com páginas de 4 KB.	p. 63
47	Tempo para o mapeamento com páginas de 4 KB.	p. 63

48	Tempo de gravação e inicialização das colunas CSOID e FIS com páginas de 4 KB.	p. 64
49	Tempo total de gerenciamento dos 31,5 milhões de OIDs.	p. 64
50	Utilização de memória principal para páginas de 4 KB.	p. 64

Lista de Abreviaturas

BLOB	<i>Binary Large Object</i>
CSOID	Contador Salva OID
FTIRR	Fragmento da Tabela de Indexação de Registros Reais
LRRF	Lista de Recuperação de Registros Físicos
LRRR	Lista de Recuperação de Registros Reais
NUGEM	Núcleo Gerenciador de Dados Multimídia
NUGO	Núcleo Gerenciador de Objetos
ODL	<i>Object Definition Language</i>
ODMG	<i>Object Data Management Group</i>
OID	<i>Object Identifier</i>
OQL	<i>Object Query Language</i>
SGAtrib	Subsistema Gerenciador de Atributos
SGBD	Sistema Gerenciador de Banco de Dados
SGBDOO	Sistema Gerenciador de Banco de Dados Orientado a Objeto
SGLog	Subsistema Gerenciador de Registros Lógicos
SGObj	Subsistema Gerenciador de i-Objetos
SGReal	Subsistema Gerenciador de Registros Reais
STL	<i>Standard Template Library</i>
TCREFOID	Tabela de Conversão de Registros Reais, Físicos, <i>Overflow</i> e OIDs
TCTEMP	Tabela de Conversão Temporária

1 Introdução

1.1 Considerações iniciais

Os Sistemas Gerenciadores de Banco de Dados Relacionais têm oferecido suporte satisfatório às diversas aplicações computacionais, como por exemplo o gerenciamento de bases de dados de banco de tumores (FERRIZZI et al., 2008a) (FERRIZZI et al., 2008b). Porém, uma outra gama de aplicações apresenta diferentes desafios para o gerenciamento de dados e como exemplos têm-se os sistemas de desenvolvimento de software auxiliado por computador, sistemas computacionais para projeto e manufatura de engenharia (KHOSHFIAN; BAKER, 1995) e sistemas para o gerenciamento de informações do projeto Genoma (STEIN; THIERRY-MIEG, 1999). Estas aplicações geram uma elevada quantidade de informações inter-relacionadas, que apresentam modelos de dados com objetos complexos exigindo o compartilhamento e armazenamento persistente de maneira hábil e confiável de seus dados.

Ao longo do tempo, sistemas relacionais como o Oracle (ORACLE,) estenderam seu modelo para suportar a orientação a objeto (ABITEBOUL; KANELAKIS, 1998). Porém, outras soluções puramente orientadas a objeto são os Sistemas Gerenciadores de Bancos de Dados Orientados a Objeto (SGBDOO) e como exemplos têm-se o Versant (WIETRZYK; ORGUN, 1998) e o EyeDB (VIARA; BARILLOT; VAYSSEIX, 1999), os quais proporcionam suporte a aplicações com objetos complexos.

1.2 Motivação e escopo

Entre os principais motivos que acarretaram no surgimento de SGBDOOs, tem-se a representação de objetos complexos, que podem ser divididos em duas categorias: estruturados e não estruturados. Objetos complexos estruturados são construídos pelas aplicações de maneira recursiva, utilizando-se os construtores de tipos. Objetos complexos não estruturados normalmente são representados por um tipo de dado que necessita de um grande volume de armazenamento, como *Binary Large Objects* (BLOBs) e *Character Large Objects* (ELMASRI; NAVATHE, 2005).

Identidade de objetos é um dos principais conceitos de um sistema orientado a objetos (SOUSA P., 1995). Para cada objeto na base de dados existe um identificador único associado denominado de *Object Identifier* (OID), em português, identificador de objeto, usado para acessá-lo e referenciá-lo em relacionamentos com outros objetos. Uma das funcionalidades importantes do gerenciamento de OIDs é o mapeamento para os seus endereços físicos na base de dados, o que representa um fator importante no desempenho de um SGBDOO e deve ser implementado de maneira a ser eficiente. Algumas técnicas exigem estruturas de dados auxiliares, como por exemplo, árvore-B, enquanto outras já incluem a localização física do objeto diretamente no OID. Além do mapeamento, outros fatores devem ser levados em consideração para o gerenciamento de OIDs, como a escalabilidade, utilização do espaço físico, controle de concorrência, replicação e relocação.

O Núcleo Gerenciador de Dados Multimídia (NUGEM) teve seu início com o Núcleo Gerenciador de Objetos (NUGO) (VALÊNCIO, 2000) e com o passar dos anos evoluiu para o NUGEM, e vem sendo desenvolvido junto ao Grupo de Banco de Dados do Instituto de Biociências, Letras e Ciências Exatas de São José do Rio Preto, com o propósito do gerenciamento de dados multimídia, além do suporte à implementação de um modelo de dados qualquer. A implementação e testes da abordagem proposta foram realizados no NUGEM, porém pode ser utilizada por qualquer arquitetura de um SGBD que contemple as técnicas e estruturas apresentadas neste trabalho.

1.3 Objetivos

A implementação de OIDs pode ser feita com OIDs físicos e OIDs lógicos. A abordagem física apresenta o problema de fragmentação da base de dados, pois os OIDs são formados diretamente pelos endereços dos objetos. Já a abordagem lógica não apresenta este problema, e pode ser feita com árvore-B, *hashing* e mapeamento direto. Cada uma destas abordagens apresenta um determinado problema: árvore-B pode ocupar muita memória e o tempo de mapeamento possui complexidade logarítmica; *hashing* pode apresentar colisões pois o conhecimento prévio do tamanho da tabela *hash* se torna inviável em base de dados, que crescem de maneira imprevisível; por último, a abordagem de mapeamento direto, que apesar de possuir um tempo de mapeamento melhor entre as três abordagens, apresenta o problema de não permitir a relocação de todas as páginas da base de dados. Outra abordagem lógica é utilizada no NUGEM, na qual uma tabela de indireção das páginas em disco é gerenciada para a manipulação dos OIDs.

Os objetivos deste trabalho foram:

- *Proposta de uma nova abordagem lógica para o gerenciamento de OIDs* - uma nova proposta para o gerenciamento de OIDs foi feita, nomeada de mapeamento indireto, cuja experimentação foi realizada no NUGEM. Foi desenvolvida uma estrutura de dados, representada por uma tabela com duas colunas, permitindo tempo de acesso constante aos seus itens. Foram desenvolvidas as funcionalidades de criar, ler, escrever, remover, *commit* e *abort* dos elementos desta tabela;
- *Avaliação do esgotamento de OIDs* - dependendo da estratégia utilizada os OIDs podem se esgotar ao longo do tempo (SHI et al., 2006). Assim, foi avaliado o esgotamento de OIDs no mapeamento indireto;
- *Comparação com as técnicas da literatura* - foi realizada também a comparação com as propostas da literatura;

1.4 Organização dos capítulos

No capítulo 2 é apresentado o conceito de identidade, o modelo de dados ODMG 3.0, as configurações de servidor de nomes e de dados, as abordagens de gerenciamento de OIDs físicos e lógicos bem como a abordagem de gerenciamento de OIDs do NUGEM. No capítulo 3 é apresentado o desenvolvimento do trabalho, no capítulo 4 os testes realizados e, por fim, no capítulo 5 é feita a conclusão e sugestão de trabalhos futuros.

2 *Fundamentação teórica*

2.1 Considerações iniciais

Os Sistemas Gerenciadores de Bancos de Dados Não Convencionais apresentam diversas funcionalidades, como gerenciamento de *cache*, gerenciamento de transações, gerenciamento de dados persistentes e transientes, controle de versões, entre outras. Nesta seção é dado enfoque aos conceitos que envolvem de maneira direta o uso de identificadores de objetos.

2.2 O conceito de identidade

O modelo relacional representa os dados por meio de tabelas, chamadas também de relações. Cada linha ou tupla em uma tabela representa um conjunto de valores relacionados e os dados a respeito das relações são armazenados em esquemas (SILBERSCHATZ; KORTH; SUDARSHAN, 1999). As tuplas são identificadas unicamente dentro de uma relação, não existindo duas tuplas com os mesmos valores para todos seus atributos. Ao definir um esquema de uma relação, o administrador da base de dados deve especificar um conjunto de atributos do esquema que deve possuir valor único para cada tupla. A esse conjunto de atributos, que identifica unicamente cada tupla em uma relação, é dada a denominação de chave primária.

No modelo orientado a objetos cada objeto é identificado unicamente dentro de toda a base de dados por meio de seu OID, não existindo dois objetos com o mesmo OID em uma base de dados. Entretanto podem existir objetos com o mesmo estado, ou seja,

mesmo valor de seus atributos. Ao contrário do modelo relacional, a identificação única de cada objeto em uma base de dados orientada a objetos não é especificada pelo administrador da base de dados, mas sim automaticamente pelo próprio SGBDOO. Ao contrário dos gerenciadores relacionais, que utilizam operações de produto cartesiano entre tabelas para conectar as tuplas, SGBDOOs utilizam OIDs para conectar objetos diretamente relacionados. Assim, entre objetos de classes com relacionamento de agregação o acesso aos objetos relacionados é feito de forma direta pelo OID, o que é mais eficiente do que operações de produto cartesiano. O produto cartesiano é efetuado apenas para objetos de classes sem relacionamento de agregação (LEE; LEE, 1995). Isto explica o motivo de SGBDOOs se apresentarem com melhor desempenho do que SGBDRs para aplicações que envolvem um grande número de objetos complexos (SHI et al., 2006).

A implementação de uma estratégia de gerenciamento de OIDs deve levar em consideração alguns requisitos a fim de que seja garantida identidade única a um objeto e seja mantida a integridade referencial. O gerenciamento de OIDs deve assegurar (KHOSHAFIAN; COPELAND, 1986):

- *Identificação única aos objetos* - um OID deve identificar apenas um objeto na base de dados e de maneira inversa, para um determinado objeto, deve existir apenas um OID o identificando;
- *Imutabilidade* - um OID deve ser invariável, ou seja, possuir uma estrutura estática por todo o ciclo de vida de um objeto, possuindo os seguintes tipos de independência:
 - *Independência de estado*: um OID não deve possuir nenhuma relação com os valores dos atributos de seu objeto;
 - *Independência de localização*: a mudança de localização na base de dados de um objeto não deve afetar seu OID;
 - *Independência estrutural*: um OID deve possuir autonomia quanto à mudança estrutural de seu objeto. Um exemplo de modificação estrutural é a alteração do esquema de um objeto.

2.3 O modelo de dados ODMG 3.0

Com a necessidade de um padrão para SGBDOOs, um consórcio formado por fornecedores de SGBDOOs, denominado *Object Data Management Group* (ODMG) , propôs um padrão que inclui um modelo de objetos, linguagem para a definição de objetos - *Object Definition Language* (ODL) , linguagem de consultas a objetos - *Object Query Language* (OQL) e integração com as linguagens de programação orientadas a objetos (ELMASRI; NAVATHE, 2005). O modelo ODMG 3.0 especifica que a hierarquia de tipos de dados para o modelo orientado a objetos pode ser dividida em tipos literais e tipos objeto. As construções usando tipos literais não possuem OID, ao passo que as do tipo objeto possuem. A completa hierarquia do modelo ODMG é (BERLER et al., 2000):

- *Literal atômico* - são os tipos de dados que representam caracteres, inteiros, valores de ponto flutuante, valores booleanos e tipos enumerados, definidos pelas palavras chave;
- *Literal coleção* - armazenam coleções de valores do tipo literal ou de objetos;
- *Literal estruturado* - são os tipos de dados que possuem uma estrutura determinada, como datas por exemplo. Além de tipos pré-definidos, o usuário pode definir novos tipos com o uso do tipo *struct*;
- *Objeto atômico* - representa um objeto possuindo atributos e métodos especificados pelo usuário;
- *Objeto coleção* - instâncias desse tipo são utilizadas para representar coleções de um tipo literal ou objeto;
- *Objeto estruturado* - é composto pelos tipos Date, Time, Timestamp, Interval, cada qual possuindo operações pré-definidas para sua manipulação.

Os relacionamentos são realizados entre dois tipos e podem possuir as seguintes cardinalidades: um para um, um para muitos ou muitos para muitos. São especificados

com o uso da palavra chave *relationship* e *inverse* e devem ser definidos em cada classe participante do relacionamento, conforme o exemplo da figura 1 (BERLER et al., 2000).

```
class Carro {  
    ...  
    relationship set<Peça> tem  
    inverse Peça :: eh_de_um;  
    ...  
}  
  
class Peça {  
    ...  
    relationship Carro eh_de_um  
    inverse Carro :: tem;  
    ...  
}
```

Figura 1: Exemplo de relacionamento no modelo ODMG (FERRIZZI, 2006).

2.4 Servidor de dados e servidor de nomes

Os dados gerenciados por um Sistema Gerenciador de Banco de Dados são transferidos para os clientes de acordo com o modelo cliente-servidor. O servidor pode ser dividido em servidor de dados e servidor de nomes, e o servidor de dados gerencia os objetos ao passo que o servidor de nomes é responsável pelo gerenciamento de OIDs.

Três configurações para a arquitetura cliente-servidor utilizando servidor de nomes e servidor de dados podem ser utilizadas (EICKLER; GERLHOF; KOSSMANN, 1995). Na primeira delas, ilustrada na figura 2, o cliente se comunica diretamente com o servidor de nomes e o de dados. Para acessar um objeto, o cliente passa o OID para o servidor de nomes, que retorna o endereço do objeto no servidor de dados. O cliente então passa este endereço ao servidor de dados e obtém o objeto.

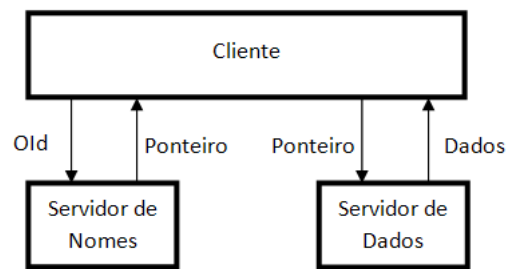


Figura 2: Cliente se comunica com o servidor de nomes e com o servidor de dados (EICKLER; GERLHOF; KOSSMANN, 1995).

Na segunda configuração, ilustrada na figura 3, o cliente se comunica apenas com o servidor de nomes. O servidor de nomes mapeia o OID para o endereço do objeto, requisita ao servidor de dados o objeto e o retorna ao cliente.

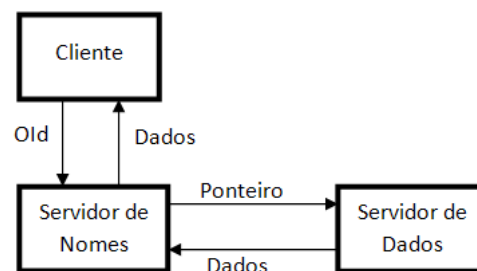


Figura 3: Cliente se comunica apenas com o servidor de nomes (EICKLER; GERLHOF; KOSSMANN, 1995).

Na última configuração, ilustrada na figura 4, o cliente se comunica com o servidor de dados, que retorna para o cliente o objeto requisitado por meio do servidor de nomes. Para evitar-se a comunicação em rede, o servidor de nomes e o de dados podem ser utilizados na mesma máquina. Esta arquitetura é a utilizada pelo NUGEM e na prática, é uma das mais utilizada por SGBDs.

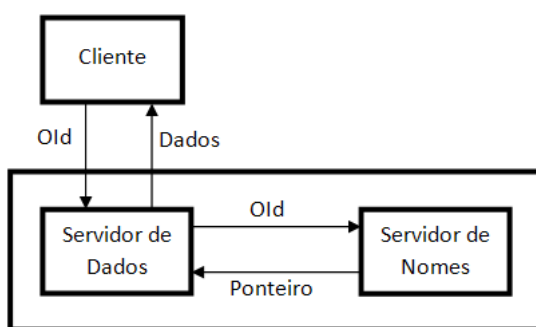


Figura 4: Cliente se comunica com o servidor de dados (EICKLER; GERLHOF; KOSSMANN, 1995).

2.5 OIDs físicos

OIDs físicos são assim denominados pois contêm o endereço físico do objeto que referenciam e uma forma de se implementar esta abordagem é compor um OID da seguinte maneira (EICKLER; GERLHOF; KOSSMANN, 1995):

- *Endereço de seu container* - em caso de bases de dados federadas este campo pode ser o identificador da base;
- *Endereço de sua página* - como o próprio nome diz, este campo representa a página no disco onde o objeto está localizado;
- *Slot* - é a localização do objeto dentro da página.

A vantagem de se utilizar esta abordagem é que objetos podem ser mapeados de maneira direta. Entretanto, para mover um objeto de um local X para Y em disco é necessário que no local X seja armazenado um ponteiro de redirecionamento indicando o novo endereço do objeto. Isso faz com que seja necessário mais de um acesso ao disco para a recuperação de um objeto, aumentando o seu tempo de leitura e eliminando a principal vantagem de OIDs físicos. Além disso, o uso de ponteiros de redirecionamento fragmenta as páginas da base de dados desperdiçando espaço físico (EICKLER; GERLHOF; KOSSMANN, 1995).

Um exemplo de sistema que faz uso de OIDs físicos é o O2. Quando novos objetos são criados, OIDs temporários são criados, sendo os OIDs permanentes criados somente após a operação de *commit* na base de dados. Para as operações de remoção de objetos, ponteiros de redirecionamento são utilizados conforme explicado anteriormente (DEUX, 1990).

2.6 OIDs lógicos

Para solucionar os principais problemas de OIDs físicos existem os OIDs lógicos, que geralmente são implementados como números inteiros, gerados de maneira incremental. OIDs lógicos, também chamados *surrogates*, são mais flexíveis que OIDs físicos pois não contêm o endereço físico do objeto que referenciam. A mudança de localização dos objetos não necessita do uso de ponteiros de redirecionamento e dessa forma o processo de reorganização da base de dados é menos complexo.

OIDs lógicos não têm a localização direta dos objetos, sendo necessário o mapeamento do OID para o respectivo endereço físico, o que gera um custo adicional ao gerenciamento de OIDs. Entretanto, este custo é baixo quando comparado ao uso de OIDs físicos (BRAU-MANDL et al., 2000).

A seguir são discutidas três abordagens para a implementação de OIDs lógicos: *hashing*, árvore-B e mapeamento direto. Para cada uma destas técnicas os seguintes fatores são discutidos:

- *Tempo de mapeamento* - o tempo de mapeamento dos OIDs para as suas localizações físicas;
- *Utilização de memória* - a utilização de memória principal para as estruturas de dados responsáveis pelo mapeamento;
- *Escalabilidade* - como a estratégia de gerenciamento de OIDs se comporta com o crescimento da base de dados;

- *Utilização do espaço físico* - como a utilização de espaço físico é afetada pelo gerenciamento de OIDs;
- *Controle de concorrência e recuperação* - como se comporta o gerenciamento de OIDs diante do controle de concorrência e recuperação;
- *Replicação* - como os OIDs podem ser replicados em diferentes servidores;
- *Relocação* - na prática SGBDs precisam efetuar a reorganização da base de dados. Várias operações como compactação da base de dados, *clustering*, alteração no esquema e *garbage collection* (SOCKUT; IYER, 2009) utilizam como base a reorganização de objetos em uma base de dados (LAKHAMRAJU et al., 2000) (GERLHOF; KEMPER; MOERKOTTE, 1996).

Considera-se a existência básica de dois tipos de páginas, as páginas de ponteiros e as páginas de dados. Na figura 5 é ilustrado este conceito, no qual um OID primeiramente é mapeado em memória para a localização física de sua página de ponteiro, que por sua vez aponta para a localização dos dados do objeto.

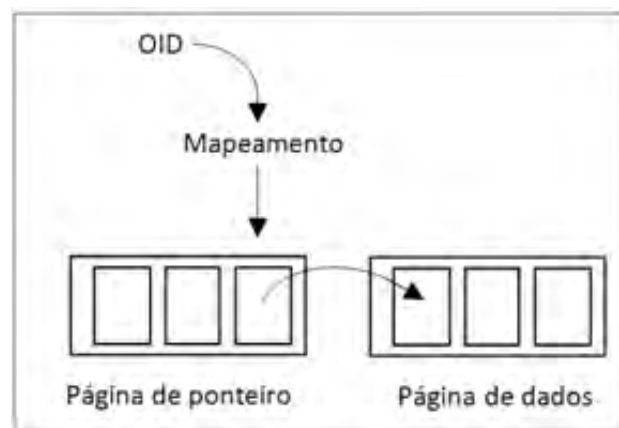


Figura 5: Mapeamento ilustrando as páginas de ponteiros e de dados.

2.6.1 Hashing

Para mapear um OID para o endereço físico de um objeto uma função *hash* pode ser utilizada, com o intuito de proporcionar tempo de acesso constante ao endereço de um

objeto dado seu OID.

A escolha da função *hash* é muito importante, pois podem ocorrer colisões e uma função *hash* que faça uma distribuição uniforme pela tabela *hash* é necessária para não prejudicar o desempenho do SGBDOO. Os quesitos tempo de mapeamento, utilização de memória, escalabilidade, utilização do espaço físico, controle de concorrência e recuperação, replicação e relocação ficam da seguinte forma (EICKLER; GERLHOF; KOSSMANN, 1995):

- *Tempo de mapeamento* - conhecendo-se previamente o tamanho da tabela *hash*, é possível que se obtenha tempo constante de mapeamento com a função *hash* mais um acesso ao disco para se obter os dados do objeto. Caso contrário, colisões podem ocorrer o que aumenta o tempo de mapeamento dos OIDs;
- *Utilização de memória* - não são necessárias estruturas adicionais, desde que a função mapeie os OIDs diretamente para suas páginas em disco;
- *Escalabilidade* - um dos maiores problemas com o uso de *hashing* em bancos de dados é a escalabilidade. Se o tamanho da tabela *hash* é conhecido previamente, a estratégia de *hash* pode ser adaptada de forma a tratar melhor as colisões, distribuindo uniformemente os objetos entre as páginas de uma base de dados. Porém, o tamanho da base de dados cresce constantemente, fazendo com que a estimativa do tamanho da tabela *hash* seja complicada;
- *Utilização do espaço físico* - de acordo com análise de várias abordagens de *hashing*, em média 69% das páginas em disco são utilizadas e essa média pode ser bem maior caso seja conhecido previamente o tamanho da base de dados;
- *Controle de concorrência e recuperação* - ainda faltam estudos avaliando o desempenho do controle de concorrência e recuperação explicando se uma abordagem simples como recuperação baseada em *log* é suficiente ou se é necessário o uso de técnicas mais elaboradas;

- *Replicação* - os OIDs podem ser replicados sem maiores problemas entre vários servidores de nomes. Porém, ao se remover um objeto todos os servidores de nomes devem ser informados;
- *Relocação* - a função *hash* distribui os objetos nas páginas da base de dados e para modificar o endereço dos objetos seria necessário encontrar uma nova função *hash*, o que seria inviável.

Dois exemplos de SGBDOOs que utilizam *hashing* são o Versant e Itasca. No Itasca, cada classe possui uma tabela *hash* associada e a leitura de um objeto é feita por meio de duas tabelas *hash*: a primeira tabela é acessada utilizando o valor da classe do objeto, a qual retorna um valor que é utilizado por uma segunda tabela *hash* para se obter a localização do objeto na base de dados (EICKLER; GERLHOF; KOSSMANN, 1995).

2.6.2 Árvore-B

Árvore-B armazena em seus nós os OIDs e os endereços físicos e deve ser percorrida toda vez que um mapeamento é necessário. Uma variante da árvore-B é a árvore-B+, que armazena em seus nós intermediários apenas os OIDs, ficando nos nós folhas os endereços físicos. Isso permite que seja mantido em memória principal um maior número de nós quando comparada com a árvore-B. Uma árvore-B+ com altura quatro pode mapear quase um bilhão de objetos. Se os dois primeiros níveis da árvore forem mantidos em memória principal, são requeridos no máximo dois acessos ao disco para recuperar o endereço de um objeto. Exemplos de sistemas que fazem uso desta abordagem são o GemStone e o SHORE (EICKLER; GERLHOF; KOSSMANN, 1995).

Os quesitos tempo de mapeamento, utilização de memória, escalabilidade, utilização do espaço físico, controle de concorrência e recuperação, replicação e relocação ficam da seguinte forma (EICKLER; GERLHOF; KOSSMANN, 1995):

- *Tempo de mapeamento* - o tempo de busca em árvore-B têm a complexidade $O(t * \log_t n)$, sendo t o grau da árvore e n o número de chaves (CORMEN et al., 2002). Além

do tempo de mapeamento, árvore-B também precisa ser gravada em disco durante a execução das transações e carregadas em disco durante a inicialização da base de dados;

- *Utilização de memória* - para cada nó é necessário manter suas chaves bem como ponteiros para os nós do próximo nível;
- *Escalabilidade* - ao contrário de *hashing*, árvore-B se comporta melhor para bases de dados grandes que crescem continuamente. Conforme os OIDs são gerados, basta a inserção de novos nós na árvore-B quando necessário. Porém, um aumento de níveis da árvore implica em um número maior de acessos ao disco para o mapeamento de OIDs;
- *Utilização do espaço físico* - a ocupação dos nós da árvore-B pode se aproximar de 100% caso objetos não sejam removidos da base de dados;
- *Controle de concorrência e recuperação* - estruturas de índice representam um gargalo no sistema se elas forem atualizadas. Alguns algoritmos foram propostos, porém reduzem significativamente o desempenho do sistema;
- *Replicação* - assim como na abordagem de *hashing*, os OIDs podem ser replicados sem maiores problemas entre os vários servidores de nomes;
- *Relocação* - quando um objeto mudar de localização na base de dados a árvore-B deve ser atualizada com o novo endereço do mesmo.

2.6.3 Mapeamento direto

Conforme ilustrado na figura 6, um OID possui três campos: *container* (2 bytes), endereço de sua página (4 bytes), endereço do registro de ponteiros (2 bytes) e o contador (4 bytes). Os três primeiros campos servem para proporcionar o acesso ao objeto e o último campo, contador, serve para manter a unicidade dos OIDs. Toda vez que um objeto é removido da base de dados, este contador é incrementado em uma unidade, invalidando

assim todas as referências ao objeto removido e permitindo a reutilização do seu registro de ponteiro (EICKLER; GERLHOF; KOSSMANN, 1995).

Na figura 6 é mostrado um segmento de ponteiros, contendo páginas de ponteiros com capacidade para cinco registros de ponteiros. Um campo chamado bitmap é uma estrutura que utiliza um bit para cada registro de ponteiros e é usado para marcar quais registros de ponteiros estão livres. O bit 0 indica que a posição está livre e o bit 1 que está ocupada. Na figura 6, pode ser observado que os segmentos de objetos possuem páginas de objetos com capacidade para armazenar dois objetos.

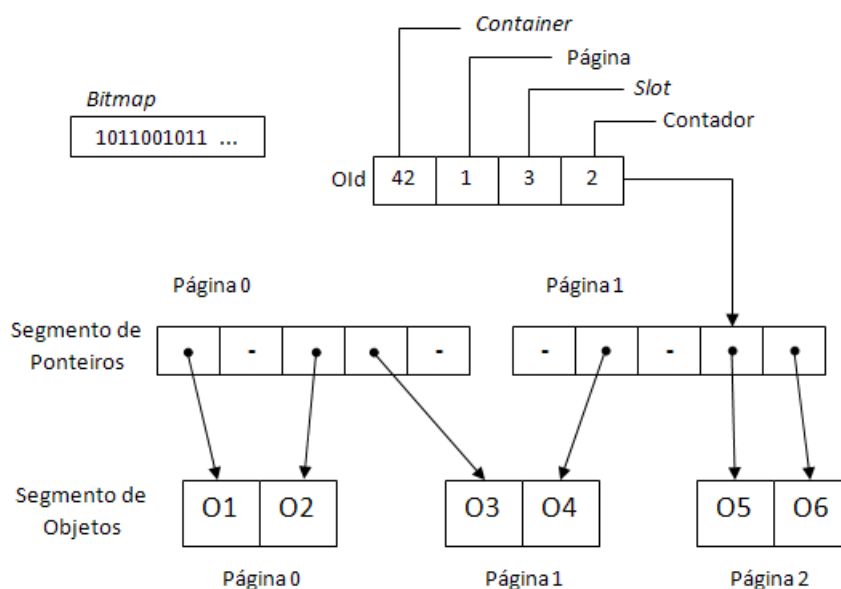


Figura 6: Exemplo com mapeamento direto (EICKLER; GERLHOF; KOSSMANN, 1995).

Por meio da abordagem descrita acima as páginas de objetos podem ser deslocadas livremente pela base de dados, mantendo a independência física dos OIDs. Ao deslocar um objeto de sua posição, basta atualizar seu registro de ponteiro, o que não modifica seu OID. Embora as páginas de ponteiros possam ser deslocadas livremente, um registro de OID deve permanecer na mesma página em que foi inicialmente armazenado até que o objeto que referencia seja removido da base de dados.

Os quesitos tempo de mapeamento, utilização de memória, escalabilidade, utilização do espaço físico, controle de concorrência e recuperação, replicação e relocação ficam da seguinte forma (EICKLER; GERLHOF; KOSSMANN, 1995):

- *Tempo de mapeamento* - o tempo de mapeamento é constante igual a um acesso ao disco;
- *Utilização de memória* - para esta abordagem não são necessárias estruturas em memória principal;
- *Escalabilidade* - o desempenho da técnica de mapeamento direto independe do tamanho da base de dados, como no caso de *hashing* e de árvore-B. Com mapeamento direto, é feito no máximo um acesso ao disco para mapear um OID, independentemente do tamanho da base de dados;
- *Utilização do espaço físico* - as páginas de ponteiros podem ser ocupadas quase que 100%. Para isto, basta reutilizar as páginas de ponteiros com espaços livres, localizando os registros de ponteiros livres através do *bitmap*;
- *Controle de concorrência e recuperação* - nenhuma estrutura auxiliar é utilizada para o mapeamento de OIDs, como em árvore-B, por exemplo. Quando um novo objeto é criado é necessário escrever apenas na página de ponteiros. Para a recuperação, técnicas usando recuperação baseada em *log* poderiam ser utilizadas;
- *Replicação* - as páginas de ponteiros podem ser replicadas entre os diversos servidores de nomes;
- *Relocação* - apenas as páginas de objetos podem ser reorganizadas na base de dados livremente, não ocorrendo o mesmo para as páginas de ponteiros de objetos. OIDs utilizam os endereços das páginas de ponteiro e reorganizá-las implicaria em modificar o OID dos objetos. Como consequência, a reorganização das páginas de objetos pode acabar não mantendo o *clustering* linear;

A técnica de mapeamento direto é a mais robusta comparada com *hashing* e árvore-B, pois precisa de no máximo um acesso ao disco para mapear um OID e não possui seu desempenho afetado pelo crescimento da base de dados. Entre *hashing* e árvore-B não há uma distinção clara em qual é melhor: se a árvore-B pode ser armazenada completamente

em memória, o seu desempenho é aproximadamente o mesmo que em *hashing*. Por outro lado, se a árvore-B não pode ser armazenada completamente na memória, o desempenho de *hashing* pode ser melhor (EICKLER; GERLHOF; KOSSMANN, 1995).

2.6.4 Esgotamento de OIDs

Algumas aplicações de larga escala podem inserir e remover milhões de objetos diariamente, e, dependendo da abordagem utilizada, os OIDs podem se esgotar em poucos anos. Para a solução do problema do esgotamento de OIDs lógicos utilizando-se *surrogates*, foi proposto recentemente um mecanismo de reuso de OIDs (SHI et al., 2006), no qual os OIDs são reutilizados e a integridade referencial é mantida atualizando-se todas as referências a um objeto que é removido.

São utilizados ponteiros bilaterais para a conexão entre os objetos, o que faz um objeto O1 armazenar uma referência ao objeto O2 e O2 armazenar uma referência ao objeto O1. Assim, quando um objeto é removido da base de dados é possível encontrar todos os objetos que o referenciam e informá-los da operação de remoção, cancelando todas as referências ao objeto removido. Os OIDs removidos são armazenados em uma lista de recuperação que é consultada sempre que um novo OID precisa ser gerado. As operações efetuadas para a reutilização de OIDs são registradas em *log* para permitir a recuperação da base de dados em caso de falhas (SHI et al., 2006).

2.7 Núcleo Gerenciador de Dados Multimídia

O NUGEM é estruturado em camadas como está ilustrado na figura 7. A camada mais inferior, o Subsistema Gerenciador de Registros Reais (SGReal) é responsável por gerenciar as páginas do disco (WATANABE; PEREIRA, 2003); o Subsistema Gerenciador de Registros Lógicos (SGLog) lida com os registros armazenados dentro das páginas em disco (LIMA, 2006); o Subsistema Gerenciador de Atributos (SGAtrib) gerencia o relacionamento entre os registros (GOUTIER, 2004) e o Subsistema Gerenciador de i-Objetos

(SGObj) é responsável por dar suporte necessário aos módulos semânticos para que se possa implementar um determinado modelo de dados (FERRIZZI, 2006). Existe também um *benchmark* para a avaliação do desempenho do NUGEM (OLIVEIRA, 2006). O foco deste trabalho foram as camadas SGReal e SGLog, as quais serão descritas em mais detalhes.

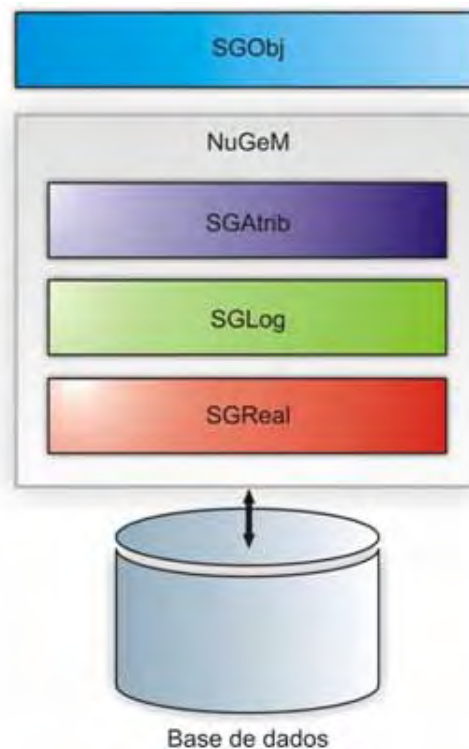


Figura 7: Arquitetura do NUGEM (FERRIZZI, 2006).

Um OID no NUGEM é representado utilizando-se um endereço lógico e um contador de quantas vezes este endereço foi reutilizado. Embora as camadas SGObj e SGAtrib utilizem os OIDs gerados pelas camadas mais inferiores, a SGLog e SGReal, estas últimas são as responsáveis pela implementação da abordagem de gerenciamento de OIDs.

A camada SGReal utiliza uma tabela de indireção denominada de Tabela de Conversão de Registros Reais, Físicos, *Overflow* e OIDs, para referenciar as páginas em disco. Durante suas operações de criação, remoção, escrita, leitura, *commit* e *abort* das páginas em disco, outras estruturas de dados auxiliares são acessadas: as listas de recuperação que armazenam índices da tabela de indireção, os quais são denominados de endereços reais, bem como endereços das páginas em disco; e as listas das transações, que armazenam

os endereços reais das páginas que foram criadas, lidas, escritas e removidas por uma determinada transação.

A camada SGLog, por sua vez, é responsável por requisitar páginas em disco à camada SGReal. Dentro destas páginas estão armazenados os registros que contêm os OIDs dos objetos, os quais são interpretados e manipulados durante as operações de criação, remoção, escrita e leitura destes registros. A camada SGLog também mantém uma lista com os endereços disponíveis dos registros que foram removidos.

Neste trabalho toda esta estrutura de gerenciamento de OIDs foi otimizada. As estruturas de dados e funcionalidades das camadas SGReal e SGLog são descritas em detalhes a seguir.

2.7.1 Subsistema de Gerenciamento de Registros Reais

A camada SGReal é a camada mais inferior do NUGEM que por meio de interação com o sistema operacional é responsável pelas seguintes funcionalidades: gerenciamento de OIDs; alocação de espaço físico em disco; gerenciamento de *cache*, gerenciamento de transações; recuperação de falhas e controle de concorrência (WATANABE; PEREIRA, 2003).

Dentre as diversas estruturas presentes na camada SGReal, as principais relacionadas a este trabalho são:

- *Tabela de Conversão de Registros Reais, Físicos, Overflow e OIDs* - a base de dados é formada por páginas no disco de tamanho fixo, podendo ser estas páginas com tamanho de 1 KB, por exemplo. Uma página é identificada dentro da base de dados por seu endereço físico, o qual é então indexado em uma tabela denominada de Tabela de Conversão de Registros Reais, Físicos, *Overflow* e OIDs (TCREFOID). Cada índice nesta tabela é denominado de endereço real e a página associada a este índice passa a ser chamada de registro real. A TCREFOID, ilustrada na figura 8, além do mapeamento de endereços reais para físicos é utilizada para a paginação *shadow* e gerenciamento de OIDs. A coluna FIS é responsável por armazenar o

endereço da página original, em estado consistente, e a coluna OVER é responsável por armazenar o endereço da página que está sendo usada por uma transação. A coluna Contador Salva OID (CSOID) é utilizada para o gerenciamento de OIDs e o seu valor é atribuído de acordo com o gerenciamento de OIDs feito na camada SGLLog.

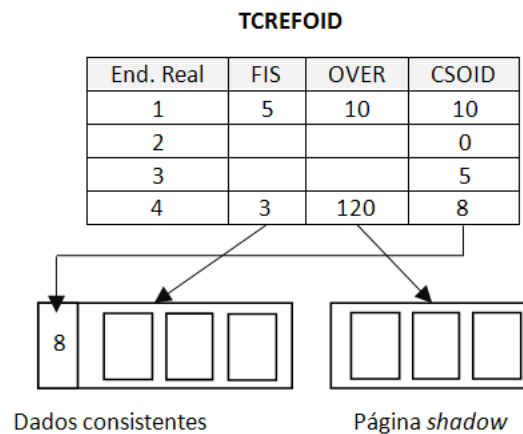


Figura 8: Tabela de Conversão de Registros Reais, Físicos, *Overflow* e OIDs.

Cada registro real contém um cabeçalho responsável por armazenar informações de controle, e uma área de dados, utilizada para armazenar os registros de dados. No cabeçalho de um registro real encontra-se o campo contadorSalvaOID, utilizado para o gerenciamento de OIDs. Este atributo e o campo CSOID presente na TCREFOID estão intimamente ligados, porém os seus valores são gerenciados na camada mais acima, a SGLLog. Quando um registro real é solicitado, a SGReal recupera um registro real e um físico, a paginação *shadow* é acionada e em seguida o valor do campo CSOID é atribuído ao campo contadorSalvaOID no cabeçalho da página. Quando um registro real é removido, o valor do atributo ContadorSalvaOID do registro real é atribuído à coluna CSOID da TCREFOID.;

- *Lista dos registros lidos, escritos, liberados e requisitados por uma transação* - cada transação possui quatro listas de endereços reais utilizadas para o controle de suas operações: lista dos registros reais lidos, lista dos registros reais escritos, lista dos registros reais liberados e lista dos registros reais requisitados.

- *Listas de recuperação de registros reais e de registros físicos* - os registros reais e os registros físicos da base de dados são removidos frequentemente durante a execução das transações e é necessária a utilização de estruturas para armazenar estes endereços para que sejam reutilizados. A SGReal possui a Lista de Recuperação de Registros Reais (LRRR) e Lista de Recuperação de Registros Físicos (LRRF), responsáveis por armazenar endereços de registros reais e físico, respectivamente, que foram removidos pelas transações. A LRRR e a LRRF são listas lineares encadeadas mantidas ordenadas de forma crescente, pois toda vez que um registro é solicitado sempre é retornado o registro com o menor endereço para que se tenha um melhor controle dos espaços vazios tanto na TCREFOID quanto em disco. As listas de recuperação possuem subdivisões para que seja feito o tratamento adequado das transações.

A LRRR se subdivide em duas listas:

- *LRRR Final* - é a lista de recuperação contendo todos os registros reais que foram liberados e efetivados na base de dados por meio da operação de *commit*;
- *LRRR Temp* - mantida apenas em memória principal, contém os registros reais liberados já efetivados na base de dados e os que ainda não foram efetivados pelas transações.

A LRRF se subdivide em três listas:

- *LRRF Final* - é a lista de recuperação contendo todos os registros físicos que foram liberados e efetivados na base de dados por meio da operação de *commit*;
- *LRRF Temp* - esta lista é mantida apenas em memória principal, e além de conter os registros físicos liberados já efetivados na base de dados também armazena os que ainda não foram efetivados pelas transações;
- *LRRF Bloqueada* - durante o *commit* são inseridos nesta lista os registros físicos com o estado consistente da base de dados e, ao término do *commit*, estes registros são então transferidos para a LRRF Final e LRRF Temp.

As funcionalidades relacionadas ao gerenciamento de registros reais utilizam como base a TCREFOID, as listas de recuperação de registros reais e físicos e as listas das transações. Durante a criação, leitura, escrita, remoção, e operações de *commit* e *abort* novos registros físicos e reais são obtidos das suas respectivas listas de recuperação, bem como atualizações nas listas de registros reais de cada transação são realizadas. Abaixo são descritas cada funcionalidade em seus detalhes:

- *Novo registro real* - Funcionalidade responsável por obter um novo registro real, recebe como parâmetro o identificador da transação e retorna um endereço de um novo registro real. Inicialmente são obtidos um endereço real da LRRR e um endereço físico da LRRF Temp. Em seguida, a coluna OVER da TCREFOID recebe o endereço físico obtido e o cabeçalho do registro real recebe o valor contido na coluna CSOID. Na figura 9 é ilustrada a TCREFOID após criação do registro real 1, que possui o registro físico localizado no endereço físico 5.

End. Real	FIS	OVER	CSOID
1	-	5	3

Figura 9: TCREFOID após a criação de um registro real.

O registro físico referenciado na coluna OVER é lido da base de dados e o seu cabeçalho contendo o contadorSalvaOID é atualizado com o valor contido na coluna CSOID da TCREFOID, e em seguida é escrito na base de dados. Por fim, o endereço real requisitado é inserido na lista dos registros reais requisitados da transação que o solicitou;

- *Leitura de um registro real* - Esta funcionalidade é responsável por ler um determinado registro real, recebendo como parâmetros um endereço real, o identificador da transação e um ponteiro para um *buffer*. Se o registro real não estiver bloqueado com modo de escrita para outra transação, o endereço real é inserido com o bloqueio de leitura na tabela de bloqueio para a transação corrente e a operação continua, caso contrário a leitura não é permitida. Em seguida a coluna OVER da TCRE-

FOID recebe o mesmo valor da coluna FIS e a página é lida do disco. Na figura 10 é ilustrada a TCREFOID após a leitura do registro real 1, que possui o registro físico localizado no endereço físico 5. Por fim, o endereço real é inserido na lista dos registros reais lidos da transação que o solicitou.

End. Real	FIS	OVER	CSOID
1	5	5	3

Figura 10: TCREFOID após a leitura de um registro real.

- *Escrita de um registro real* - Este método recebe como parâmetros um endereço real, o identificador da transação e um ponteiro para um *buffer*. Inicialmente é verificado se o registro não está bloqueado com escrita ou leitura para outra transação e se o registro real não estiver bloqueado, então é inserido com o bloqueio de escrita na tabela de bloqueio da transação corrente e a operação continua.

Um novo endereço físico, que armazena as novas modificações feitas no registro real, é obtido da LRRF Temp e atribuído à coluna OVER da TCREFOID. O endereço real é inserido na lista dos registros reais escritos da transação e a página é escrita na base de dados.

Na figura 11 é ilustrada a escrita do registro real 1, que possui o registro físico original localizado no endereço físico 5. Devido ao esquema utilizado de paginação *shadow*, as alterações feitas são gravadas no registro físico 12 até que a transação seja finalizada.

End. Real	FIS	OVER	CSOID
1	5	12	3

Figura 11: TCREFOID após a escrita de um registro real.

- *Remoção de um registro real* - Este método recebe como parâmetros um endereço real e o identificador de transação. Inicialmente é verificado se o registro não está bloqueado com escrita ou leitura para outra transação e se não houver bloqueio o endereço real é inserido na LRRR Temp. Em seguida a coluna CSOID recebe o

valor contido no cabeçalho do registro real e o endereço físico é inserido na LRRF Temp. O endereço real é inserido na lista dos registros reais liberados da transação e à coluna OVER da TCREFOID é atribuído o valor zero. Na figura 12 é mostrado como fica a TCREFOID após a remoção do registro real 1.

End. Real	FIS	OVER	CSOID
1	-	-	20

Figura 12: TCREFOID após a remoção de um registro real.

- *Efetivar registro real* - Esta operação efetiva na base de dados as alterações feitas nos registros reais. Para cada transação existe uma lista de registros reais escritos, e durante o *commit* esta lista é percorrida e todos os registros reais nela contidos são efetivados. O primeiro passo desta operação é verificar a existência de um endereço físico na coluna FIS da TCREFOID, e caso exista, este endereço físico é inserido na LRRF Bloqueada. Em seguida o endereço real é removido da LRRR Temp e a coluna FIS da TCREFOID recebe o valor da coluna OVER. Por fim, o registro físico associado ao registro real é escrito na base de dados e, ao final da operação de *commit*, o Fragmento da Tabela de Indexação de Registros Reais (FTIRR) deste endereço real é gravada na base de dados;
- *Liberar registro real* - Esta operação reflete na base de dados os registros reais que foram removidos por uma transação. Durante o *commit* a lista dos registros liberados de uma transação é percorrida e todos os registros reais nela contidos são liberados de forma efetiva da base de dados. Se existir um endereço físico na coluna FIS da TCREFOID este endereço é inserido na LRRF Bloqueada. O endereço real é então inserido na LRRR Final e na LRRR Temp, e o FTIRR deste registro real é gravada na base de dados ao final do *commit*;
- *Cancelar registro real* - Esta operação é utilizada para cancelar as alterações feitas em um registro real. Durante o *commit* este método é chamado para cada item da lista dos registros requisitados e não utilizados e durante o *abort* para cada item das

listas de lidos, escritos, liberados e requisitados de uma transação. O cancelamento de um registro real é feito da seguinte forma: se a coluna FIS da TCREFOID for igual a zero, isso indica que o registro real não possui um estado anterior consistente e o seu endereço é inserido na LRRR Temp. Em seguida é verificado se a coluna OVER possui um endereço físico, para então inseri-lo na LRRF Temp.

2.7.2 Subsistema de Gerenciamento de Registros Lógicos

Uma página em disco indexada pela TCREFOID representa um registro real, o qual armazena por sua vez vários registros, denominados de registros lógicos. São disponibilizados os tipos lógicos colônia, objeto, tupla, lista, BLOB e índice, sendo o mais importante para o gerenciamento de OIDs o tipo objeto. Um registro lógico objeto referencia um ou mais registros lógicos dos outros tipos, e esta configuração pode representar um objeto com seus dados.

De acordo com o tipo dos registros lógicos armazenados por um registro real, ele pode ser denominado de registro real objeto, indicando que armazena registros lógicos do tipo objeto, ou registro real genérico, armazenando registros lógicos de outros tipos exceto o tipo objeto. Um registro real objeto representa uma página de ponteiro, ao passo que um registro real genérico representa uma página de dados. Estes conceitos são ilustrados na figura 13.

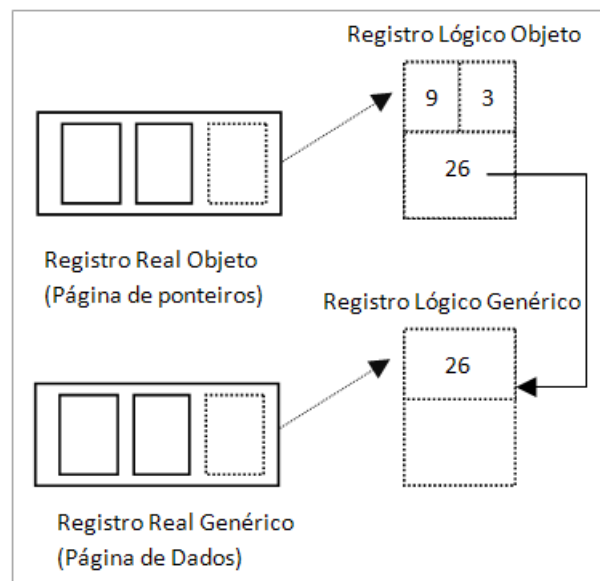


Figura 13: Representação dos registros reais e lógicos no NUGEM.

Nos trabalhos realizados na SGLog, utiliza-se a denominação de bloco lógico para os registros reais gerenciados na SGLog(LIMA, 2006)(WATANABE; PEREIRA, 2003). Assim, um bloco lógico contém vários registros lógicos, e na figura 14 é ilustrada esta configuração. Dentre as informações contidas no cabeçalho de um bloco lógico tem-se o campo contadorSalvaOID, que é utilizado para que um bloco lógico liberado possa ser reutilizado para armazenar qualquer tipo de registro lógico. Dentre os campos de um registro lógico, tem-se o seu endereço e um contador, denominado de contOID, que representa quantas vezes o registro lógico foi removido. Na figura 14 cada registro lógico é representado por um retângulo azul, e esta cor indica que o registro está em uso. Os campos presentes no cabeçalho de cada registro lógico são o endereço lógico e o contOID.

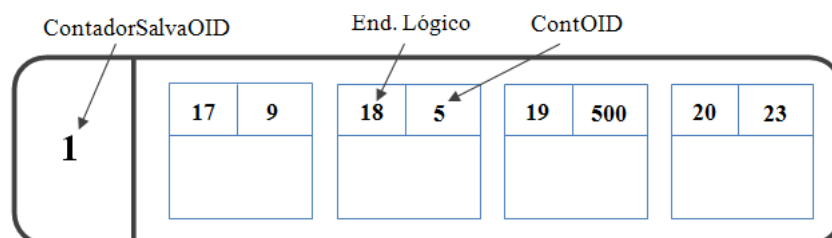


Figura 14: Representação da estrutura de um bloco lógico.

A estratégia utilizada para identificar unicamente objetos no Núcleo Gerenciador de Dados Multimídia - NUGEM utiliza dois atributos dos registros lógicos do tipo objeto:

- *endLogico* - endereço lógico do registro objeto, não emprega atributos que indiquem diretamente a sua localidade física, permitindo a independência física de um OID. É calculado de acordo com a seguinte fórmula, na qual *iBloco* representa o endereço real, *MAX_REG_POR_BLOCO* é uma constante indicando o número máximo de registros lógicos em um bloco lógico e *iReg* indica o índice do registro lógico dentro do bloco:

$$\text{endLogico} = \text{iBloco} * \text{MAX_REG_POR_BLOCO} + \text{iReg}$$

Figura 15: Fórmula utilizada para o cálculo de um endereço lógico.

- *contOID* - representa o número de vezes que *endLogico* foi reutilizado. Cada vez que um registro lógico do tipo objeto é removido esse valor é incrementado em uma unidade, garantindo que ao longo do tempo não existam OIDs com o mesmo valor. É implementado como um inteiro sem sinal de 4 *bytes*, o que permite a criação de aproximadamente 4,3 bilhões de objetos por endereço lógico.

Dado um OID na forma (*EndLogico*, *ContOID*), o mapeamento para sua localização física é constante. Primeiramente é necessário utilizar a fórmula da figura 15 para localizar a entrada na *TCREFOID*, para em seguida localizar a sua página em disco. Como a *TCREFOID* é implementada utilizando vetores, este acesso também é constante. O mapeamento de um OID até a localização de um registro lógico é ilustrado na figura 16.

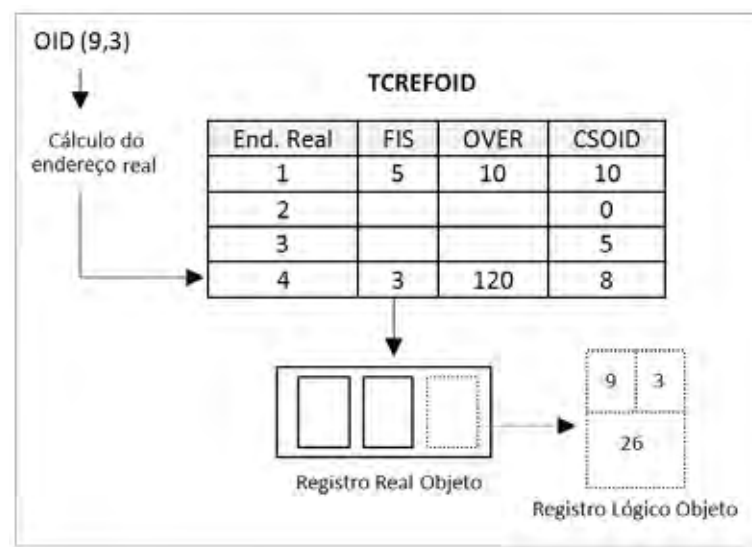


Figura 16: Mapeamento de um OID para a localização física de um registro lógico do tipo objeto.

As principais funcionalidades da SGLog referentes ao gerenciamento de OIDs são a alocação de um novo endereço lógico, responsável pela criação de um novo OID, e a operação de remover um registro lógico, responsável por remover um OID da base de dados. A operação de alocar um novo endereço lógico obtém um bloco lógico com uma posição livre, e retorna o endereço desta posição e o seu contOID. Já a operação de remoção, localiza o registro lógico, o marca como livre e incrementa o valor do seu contOID. Estas funcionalidades são descritas em detalhes:

- *Alocação de um endereço lógico* - esta operação recebe como parâmetro o tipo lógico e o identificador da transação e retorna um novo endereço lógico. A SGLog mantém uma lista linear encadeada de recuperação com os blocos lógicos que possuem espaços livres. No caso mais simples, um endereço é obtido diretamente de uma posição livre de um bloco na lista de recuperação. O outro caso é quando nenhum endereço lógico está disponível e é necessário solicitar um novo bloco lógico à SGReal. Ao obter-se um novo bloco lógico do tipo objeto, é efetuado o espalhamento do contadorSalvaOID, que consiste em copiar o valor do contadorSalvaOID presente no cabeçalho do bloco lógico para o campo contOID de todos os registros lógicos contidos no bloco. Na figura 17 é ilustrado o espalhamento do contadorSalvaOID,

garantindo-se que todos os novos registros lógicos possuam seu contOID igual ao valor 501.

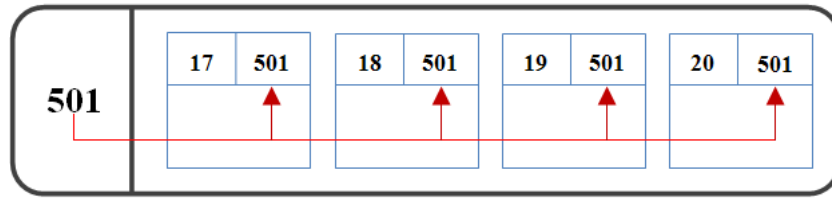


Figura 17: Espalhamento do contadorSalvaOID.

- *Remoção de um endereço lógico* - recebe como parâmetros o endereço lógico, o tipo lógico e o identificador da transação. É dado o exemplo de remoção do registro lógico com endereço 20, ilustrado anteriormente na figura 14. Quando um registro lógico do tipo objeto é removido, o seu contOID é incrementado em uma unidade como ilustrado em vermelho na figura 18. O incremento do contOID garante que não exista a duplicidade de OIDs, pois cada par (Endereço Lógico, ContOID) é único ao longo do tempo.

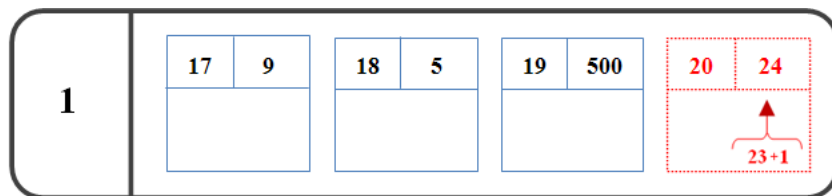


Figura 18: Representação de um bloco lógico durante a remoção de um registro lógico do tipo objeto.

Se o bloco ainda tiver registros em uso não é necessário fazer mais nada em relação ao gerenciamento de OIDs. Se o registro lógico removido for o último do bloco, então é necessário efetuar a atribuição do máximo ContOID, que consiste em obter o máximo contOID presente nos registros lógicos do bloco e atribuir este valor ao campo contadorSalvaOID no cabeçalho do bloco lógico. Como exemplo, suponha que sejam removidos os registros com endereços 17, 18 e 19, ilustrados anteriormente na figura 18. Eles têm então o seu contOID incrementado em uma unidade, conforme ilustrado na figura 19. Em seguida, é localizado o máximo ContOID, no caso o valor

501, o qual é atribuído ao ContadorSalvaOID no cabeçalho do bloco lógico, como ilustrado na figura 19. A atribuição do máximo contOID é uma forma de manter a unicidade dos OIDs e ao mesmo tempo manter o uso genérico dos blocos lógicos.

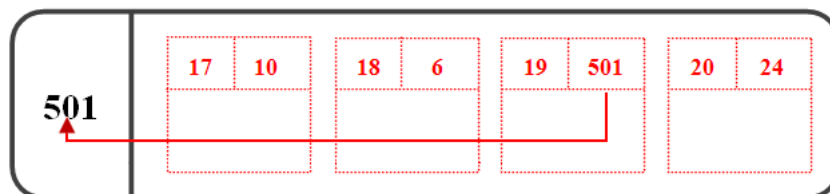


Figura 19: Atribuição do máximo contOID.

2.8 Considerações finais

Neste capítulo foram apresentadas as principais contribuições na área de gerenciamento de identificadores de objetos. Pode-se notar a carência de trabalhos correlatos, pois na literatura encontram-se poucos trabalhos relacionados diretamente ao desenvolvimento de técnicas de gerenciamento de OIDs, sendo o artigo mais recente abordando a implementação de OIDs do ano de 1995 (EICKLER; GERLHOF; KOSSMANN, 1995). Em um trabalho do ano de 2000, é proposta uma nova abordagem de gerenciamento de OIDs, porém para o tipo específico de bases de dados temporais (NØRVÅG, 2000). Nesta abordagem, os OIDs possuem um *timestamp* que são atualizados toda vez que um objeto é alterado, e é discutido que o desempenho de sua técnica é ligeiramente menor do que os das técnicas convencionais. No ano de 2006, foi publicado um artigo no qual é discutido o problema do esgotamento de OIDs, onde uma arquitetura de reuso de OIDs é proposta (SHI et al., 2006), entretanto não são discutidas implementações de gerenciamento de OIDs.

3 *Desenvolvimento do trabalho*

3.1 Considerações iniciais

Na arquitetura do NUGEM, as camadas que estão relacionadas ao desempenho na geração de OIDs são a SGReal e a SGLog. A camada SGReal, utilizada para a manipulação das páginas em disco, é responsável por indexar as páginas da base de dados em uma tabela de indireção (TCREFOID) e então repassá-las para a camada mais acima, a SGLog, que então interpreta e gerencia os OIDs dentro destas páginas.

O objetivo deste trabalho foi proporcionar uma nova estratégia para o gerenciamento dos OIDs com um menor número de acessos ao disco e menor tempo de processamento, reduzindo o tempo de gerenciamento das páginas de OIDs. Além disso, as páginas contendo dados de objetos também tiveram seu desempenho melhorado.

As atividades realizadas para a abordagem de mapeamento indireto foram:

- *Nova estrutura para o gerenciamento de OIDs* - nomeada de Tabela de Conversão Temporária (TCTEMP), é utilizada na SGReal para melhorar o desempenho no gerenciamento das páginas contendo OIDs e dados;
- *Nova abordagem para o gerenciamento de OIDs na tabela de indireção TCREFOID* - a coluna CSOID da TCREFOID, antes utilizada para armazenar o máximo contOID de um registro real, referencia agora uma página em disco contendo contOIDs consistentes;
- *Avaliação do problema de esgotamento de OIDs* - o esgotamento de OIDs pode

prejudicar o desempenho de um SGBD, portanto este problema foi avaliado para a abordagem de mapeamento indireto;

- *Otimização das listas de recuperação e listas utilizadas pelas transações* - as listas utilizadas pelo NUGEM nas camadas SGReal e SGLog foram otimizadas com a utilização de árvores vermelho-preto, que constituem estruturas de dados formadas por árvores de pesquisa binária balanceadas, garantindo que operações básicas como busca, inserção e remoção tenham tempo $O(\log n)$ no pior caso (CORMEN et al., 2002).

3.2 Subsistema de Gerenciamento de Registros Reais

As principais contribuições nesta camada para o gerenciamento de OIDs foram o desenvolvimento e gerenciamento da Tabela de Conversão Temporária (TCTEMP) e a modificação do gerenciamento do campo CSOID na TCREFOID. Outra contribuição foi a utilização da classe árvore vermelho-preto nas listas LRRR, LRRF, listas de registros lidos, escritos, requisitados e liberados das transações.

3.2.1 Tabela de Conversão Temporária e Tabela de Conversão de Registros Reais, Físicos, *Overflow* e OIDs

Durante a execução de uma transação registros físicos são removidos e solicitados à LRRF, e o tempo de acesso a esta lista é $O(\log(n))$ para o pior caso. Entretanto, registros físicos removidos por uma transação poderão ser reutilizados com tempo de acesso constante utilizando a TCTEMP, o que permite um ganho de desempenho na criação de novos registros reais objetos e genéricos. A estrutura TCTEMP, ilustrada na figura 20, é uma tabela de conversão temporária e não é armazenada em disco, sendo implementada utilizando um vetor, proporcionando acesso rápido aos seus elementos. Possui duas colunas, a saber:

- *FIS_OID_TEMP* - coluna utilizada para o gerenciamento dos OIDs, armazena temporariamente uma referência aos contOIDs de um registro real do tipo objeto que foi

removido, porém ainda não efetivado na base de dados. Isto permite a reutilização de um endereço físico de maneira eficiente, além de armazenar a última versão dos contOIDs para que eles possam ser efetivados na base de dados caso a transação efetue a operação de *commit*;

- *FIS_GEN_TEMP* - utilizada para melhorar o desempenho da criação de novos registros reais genéricos, armazena temporariamente o endereço de um registro físico que estava associado a um registro real genérico que foi removido.

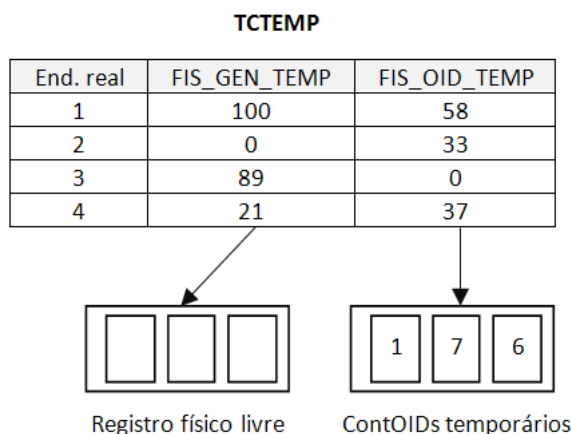


Figura 20: Tabela de conversão temporária.

Toda vez que um registro real é criado ou removido, é preciso buscar ou remover um endereço físico da lista de recuperação LRRF, e o tempo desta busca possui complexidade $O(\log(n))$. Já na TCTEMP, este acesso tem tempo constante, permitindo um ganho no desempenho das operações de criação e remoção de registros reais objeto e genérico. Outro fator é a redução de acessos ao disco durante as operações da SGReal, pois na abordagem antiga era necessário efetuar operações de escrita e leitura no disco para se atualizar o valor do campo contadorSalvaOID contido no cabeçalho das páginas. Como a TCTEMP referencia diretamente em disco os contOIDs temporários, e a coluna CSOID os contOIDs consistentes, as operações de leitura e escrita em disco foram reduzidas.

A coluna CSOID da TCREFOID referencia o máximo contOID do registro real, porém com a nova abordagem esta coluna armazena o endereço de um registro físico que contém

os contOIDs consistentes do registro real, conforme ilustrado na figura 21. Isso evita as operações de espalhamento e busca pelo máximo contOID, realizadas na SGLog.

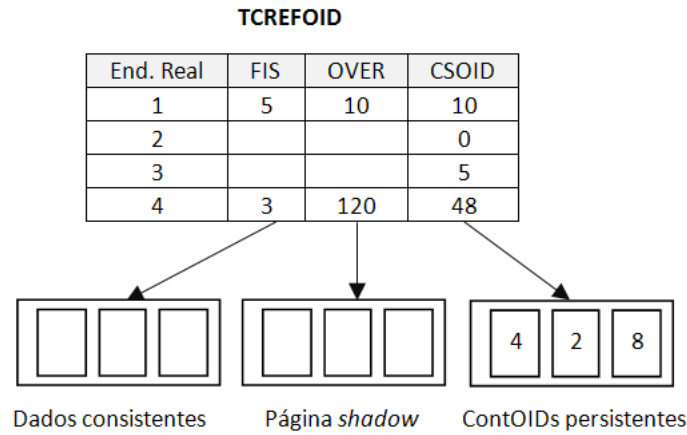


Figura 21: Gerenciamento de OIDs na TCREFOID por meio da coluna CSOID.

Considera-se neste trabalho que as estruturas TCREFOID e TCTEMP possam ser armazenadas completamente em memória. Assumindo-se a utilização de páginas de 4 KB, 300 milhões de páginas na base de dados permitiriam armazenar aproximadamente por volta de 1,12 TB de dados, sendo necessário aproximadamente 4,5 GB de memória RAM. Entretanto, otimizações no uso de memória pelas estruturas poderiam ser realizadas como trabalhos futuros, visando reduzir a utilização de memória utilizada diminuindo o custo para se manter um servidor.

3.2.2 Funcionalidades para o gerenciamento da Tabela de Conversão Temporária

Para a manipulação dos elementos da TCTEMP, bem como da coluna CSOID da TCREFOID, foram propostas diversas funcionalidades, que tiveram como base as operações já implementadas pelo NUGEM. No entanto, todas tiveram de ser desenvolvidas novamente, para se adequar à nova abordagem de mapeamento indireto:

- *Novo registro real* - toda vez que esta funcionalidade é executada, um novo endereço real é obtido da LRRR e então é necessário obter um endereço físico para ser alocado na coluna OVER da TCREFOID. Na implementação antiga, o endereço físico era

sempre obtido da LRRF Temp, ao passo que com a nova abordagem poderão ocorrer os cenários descritos a seguir.

Se o registro real requisitado for do tipo objeto, são possíveis três casos:

1. *Registro real novo* - neste caso um endereço físico é obtido da LRRF Temp e o registro real é inicializado com zero para todos os contOIDs;
2. *Registro real persistente* - o registro real já foi usado e removido por uma transação já efetivada na base de dados. O endereço físico é obtido verificando-se primeiramente se há um endereço físico na coluna FIS_GEN_TEMP da TC-TEMP, e, caso esta coluna possua um valor igual a zero, um novo endereço é obtido da LRRF Temp. Em seguida são lidos os contOIDs efetivados na base de dados e referenciados pela coluna CSOID da TCREFOID para que sejam utilizados no novo registro real;
3. *Registro real transiente* - o registro real já foi usado e removido pela transação atual. A coluna OVER da TCREFOID recebe o valor da coluna FIS_OID_TEMP da TCTEMP, que referencia o registro físico com os contOIDs temporários.

Se o registro real requisitado for do tipo genérico, são possíveis dois casos:

1. *Registro real novo* - é atribuído à coluna OVER da TCREFOID um novo endereço físico obtido da LRRF Temp.
2. *Registro real já foi usado* - neste caso a coluna OVER da TCREFOID recebe o valor da coluna FIS_GEN_TEMP da TCTEMP;

Assim que um endereço físico é obtido, o mesmo é inserido na coluna OVER da TCREFOID e na lista de registros requisitados da transação corrente.

Na implementação antiga era necessário escrever e ler na base de dados toda vez que um endereço real era requisitado, pois após o registro físico ser obtido tinha seu campo contadorSalvaOID atualizado com o valor contido na coluna CSOID da

TCREFOID. Com a nova abordagem é necessário apenas efetuar operações de escrita para registros reais do tipo objeto e que sejam ou registros reais nunca utilizados ou que tenham sido efetivados na base de dados. Se o registro real foi removido pela transação atual, então a coluna FIS_OID_TEMP possui os contOIDs temporários e permite que esta página seja utilizada diretamente, pois já está atualizada na base de dados. Operações de leitura são realizadas apenas para o caso no qual os registros reais foram removidos por uma transação já efetivada na base de dados.

Na figura 22 é ilustrada a divisão da operação de criar em subcasos, o que permitiu a redução dos acessos ao disco. A seta azul ilustra uma operação de leitura e a vermelha uma operação de escrita na base de dados. Antes estas operações eram realizadas durante toda operação de criar, e com a nova abordagem são realizadas apenas para registros reais objeto, que sejam novos ou usados persistentes.

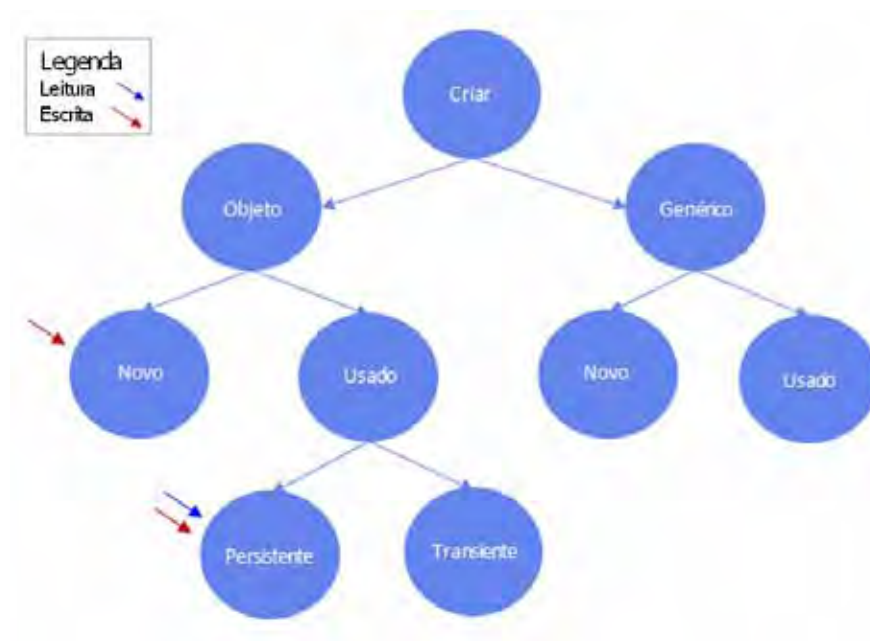


Figura 22: Redução do número de operações de escrita e leitura em disco da operação de criar uma nova página de OIDs e de dados.

Outra vantagem no uso da TCTEMP é que em algumas situações evita-se o acesso à LRRF Temp, o que permite um ganho de desempenho na criação de novos registros reais e consequentemente na criação de novos OIDs, pois o acesso às colunas FIS_GEN_TEMP e FIS_OID_TEMP é de ordem constante, ao passo que o acesso à

LRRF é de ordem logarítmica.

- *Escrever registro real* - para realizar a operação de escrita em um registro real é necessário obter uma página *shadow* para garantir a consistência da base de dados. Primeiramente é verificada a coluna FIS_GEN_TEMP da TCTEMP e, em seguida, caso esta coluna seja igual a zero, o endereço físico é obtido da LRRF Temp. Novamente, o uso da TCTEMP permite um ganho de desempenho para a escrita dos registros reais, pois primeiramente procura-se um endereço físico na TCTEMP, e caso não seja encontrado, a LRRF Temp é acessada. Como a TCTEMP é similar à TCREFOID em sua estrutura, ou seja, implementada utilizando um vetor, o acesso às suas colunas não causa sobrecarga no sistema;
- *Remover registro real* - se o registro real removido for do tipo objeto, o endereço físico da página *shadow* é inserido na coluna FIS_OID_TEMP da TCTEMP, caso contrário é inserido na coluna FIS_GEN_TEMP. Além de evitar a operação de inserção em árvore vermelho-preto, isso diminui o tamanho em memória da LRRF, reduzindo o seu espaço de busca. Outra contribuição nesta operação é a de eliminar a necessidade de leitura ao disco na base de dados. Anteriormente era preciso ler da base de dados o endereço físico da coluna OVER para se obter o valor do contadorSalvaOID e como a TCTEMP lida com os endereços dos registros físicos onde os contOIDs estão armazenados, isso não é mais necessário;
- *Efetivar registro real* - nesta funcionalidade primeiramente é verificado se a coluna OVER da TCREFOID contém um endereço físico, para poder atribuí-lo à coluna FIS da TCREFOID. Em seguida, é verificado se a coluna FIS da TCREFOID possui um endereço físico e, caso exista, este endereço é inserido então na LRRF Bloqueada, pois o registro físico deste endereço contém dados consistentes da versão anterior da base de dados. Após estas verificações, o endereço da coluna OVER é então atribuído à coluna FIS. O próximo passo é verificar se a coluna FIS_OID_TEMP da TCTEMP é diferente de zero, o que indica que o registro real foi utilizado e

removido anteriormente contendo registros lógicos do tipo objeto. Neste caso é necessário então atualizar a coluna CSOID da TCREFOID com este novo endereço físico. Se a coluna CSOID possuir um valor diferente de zero, significa que contém um endereço físico de contOIDs consistentes e então é necessário inserir este endereço físico na LRRF Bloqueada;

- *Liberar registro real* - nesta operação a coluna FIS_OID_TEMP da TCTEMP indica se houve ou não a remoção do registro real contendo registros lógicos do tipo objeto. Se a FIS_OID_TEMP possuir um valor diferente de zero, então está referenciando um registro físico contendo contOIDs que precisam ser efetivados na base de dados. O valor desta coluna é então armazenado na coluna CSOID da TCREFOID, e o registro físico antes contido na coluna CSOID é inserido na LRRF Bloqueada;
- *Cancelar registro real* - o endereço físico contido na coluna OVER da TCREFOID é inserido na FIS_GEN_TEMP da TCTEMP caso esta coluna tenha seu valor igual a zero, ou inserido na LRRF Temp, caso contrário. Em seguida, é avaliado se o registro real foi utilizado e removido com registros lógicos do tipo objeto, verificando-se a coluna FIS_OID_TEMP da TCTEMP. Se o valor desta coluna for diferente de zero, este endereço físico é então atribuído à coluna FIS_GEN_TEMP, caso esta última coluna tenha o seu valor igual a zero. Caso contrário, o valor da FIS_OID_TEMP é inserido na LRRF Temp.

3.2.3 Otimização das listas do Subsistema de Gerenciamento de Registros Reais

As listas LRRR, LRRF, listas dos registros lidos, escritos, requisitados e liberados pelas transações utilizavam lista encadeada linear ordenada, o que gerava certa sobrecarga na manipulação dos seus itens. Como melhoria no desempenho da SGReal, as listas de recuperação de registros reais e de registros físicos foram desenvolvidas utilizando-se árvore vermelho-preto, o que permite um ganho no desempenho durante as operações de busca, inserção e remoção de endereços devido à menor complexidade de tempo de árvores

vermelho-preto em relação a listas lineares.

3.2.4 Linearidade no crescimento da base de dados

A linearidade no crescimento da base de dados permite que tanto as estruturas de dados TCTEMP e TCREFOID, bem como a base de dados, sejam utilizadas de forma controlada, sempre dando-se prioridade aos menores endereços, facilitando operações como *garbage collection* e compactação da base de dados.

A utilização da lista de recuperação de registros reais (LRRR) permite que a utilização dos índices da TCREFOID e TCTEMP seja feita de forma linear, ao passo que a lista de recuperação dos registros físicos (LRRF) permite que as páginas da base de dados sejam utilizadas de forma linear. Com a abordagem proposta, a linearidade para a LRRR e LRRF é mantida, no entanto um subconjunto de endereços físicos é mantido na TCTEMP, e para este subconjunto a linearidade não é garantida.

Durante a criação de registros reais, as páginas da base de dados são obtidas dando-se prioridade à TCTEMP, pois esta possui tempo de acesso constante. Em seguida, se essa tabela não possuir um endereço físico livre para um registro real, é então solicitado um endereço à LRRF. Assim, ganha-se desempenho, porém um subconjunto de endereços físicos livres é mantido na TCTEMP, enquanto outro subconjunto é mantido na LRRF. Embora o subconjunto de endereços físicos na TCTEMP seja utilizado, ele não é feito em ordem linear assim como é feito na LRRF. Dessa forma, é mantida a linearidade no crescimento da base apenas para o subconjunto de endereços contidos na LRRF.

O tratamento da linearidade no crescimento da base de dados está fora do escopo deste trabalho, e poderá ser tratada futuramente em operações como *Garbage Collection* e compactação da base de dados.

3.3 Subsistema de Gerenciamento de Registros Lógicos

Com o uso da TCTEMP e a nova abordagem do uso da coluna CSOID da TCREFOID as operações de espalhamento de OIDs e de atribuição do máximo contOID não são mais necessárias. Além disso, nesta camada foi otimizada a lista de recuperação de registros lógicos com a utilização de árvores vermelho-preto.

3.3.1 Atribuição do máximo contOID e espalhamento dos contOIDs

Cada vez que um bloco é removido, é preciso buscar junto aos registros lógicos do bloco lógico o maior contOID, e, cada vez que um bloco novo é alocado, é necessário atribuir a cada registro lógico o valor do campo contadorSalvaOID contido no cabeçalho do bloco lógico. O número de iterações realizadas nestas operações depende do tamanho da página utilizada pela base de dados, como mostrado na figura 23.

Tamanho da página	Número médio de registros
1KB	22
2KB	44
4KB	90
8KB	181

Figura 23: Número de registros lógicos do tipo objeto por tamanho de página.

Com as alterações propostas para a SGReal as operações de espalhamento e atribuição do máximo contOID não são mais necessárias, otimizando as operações de criação e remoção de registros lógicos.

3.3.2 Avaliação do esgotamento de OIDs na abordagem de mapeamento indireto

Uma grande quantidade de remoções de objetos feita diariamente pode gerar o esgotamento de OIDs (SHI et al., 2006). Assim, durante o desenvolvimento de uma estratégia

de gerenciamento de OIDs deve-se avaliar o tamanho do campo utilizado para o OID e em quanto tempo poderiam se esgotar suas possibilidades.

O campo contOID de um registro lógico é um inteiro sem sinal de 32 bits, e por isso possui seu valor máximo aproximadamente igual a 4,3 bilhões. Se um contOID atingir este valor máximo, o registro lógico não poderá ser reutilizado, devendo ser inutilizado, pois todas as referências a esta localização serão inválidas.

Para avaliar em quanto tempo um contOID de um determinado registro lógico se esgota, neste trabalho foi proposta a fórmula apresentada na figura 24.

$$esgotamento = \frac{2^{\text{NumBits}} - 1}{\text{RemocoesPorDia}}$$

Figura 24: Tempo de esgotamento de um OID na abordagem de mapeamento indireto.

O campo NumBits representa a quantidade de bits utilizada para a implementação do contOID, que no mapeamento indireto é igual a 32 bits, e RemocoesPorDia representa a quantidade estimada de remoções por dia de um determinado registro lógico. Assumindo-se que um determinado registro lógico seja removido mil vezes por dia, o seu endereço lógico seria inutilizado em aproximadamente 4 milhões de anos. Se a quantidade de remoções fosse de um milhão por dia, o esgotamento ocorreria em pouco mais de 4 mil anos.

Enquanto na solução inicial para o esgotamento de OIDs é considerada a abordagem de *surrogates* (SHI et al., 2006), que utiliza apenas um único contador para a criação dos OIDs, o mapeamento indireto utiliza um contador para cada registro lógico, o que praticamente evita o problema do esgotamento de OIDs.

3.4 Comparação entre mapeamento indireto, mapeamento direto, *hashing* e árvore-B

Comparando-se as estratégias de mapeamento indireto, mapeamento direto, *hashing* e árvore-B em relação aos quesitos tempo de mapeamento, utilização de memória, escalabilidade, utilização do espaço físico, controle de concorrência e recuperação, replicação e relocação tem-se:

- *Tempo de mapeamento* - árvore-B possui tempo de mapeamento logarítmico, ao passo que mapeamento direto, *hashing* e mapeamento indireto possuem tempo de mapeamento constante. Entretanto, com a abordagem de *hashing* podem ocorrer colisões, aumentando o seu tempo de mapeamento. A abordagem de mapeamento indireto possui um tempo ligeiramente maior do que a abordagem de mapeamento direto, pois utiliza as tabelas de indireção. No entanto, este tempo representa uma pequena porcentagem diante do tempo total das operações, e como benefício deste custo tem-se a flexibilidade na relocação das páginas, atributo não contemplado pela abordagem de mapeamento direto. No capítulo 4 são discutidos os testes realizados para se aferir o tempo de mapeamento da abordagem de mapeamento indireto;
- *Utilização de memória* - dentre as abordagens, apenas árvore-B e mapeamento indireto utilizam estruturas em memória para o mapeamento. Com o propósito de ilustrar a diferença na utilização de memória entre as duas abordagens, assumiu-se OIDs de 4 bytes e ponteiros para os nós folhas de 4 bytes para árvore-B. Com a utilização de páginas de 4 KB, tem-se 31,5 milhões de OIDs e a utilização de memória para árvore-B é de aproximadamente 240 MB, quase 45 vezes maior do que a abordagem de mapeamento indireto, que utiliza aproximadamente 5,34 MB. Isso ocorre pois, enquanto árvore-B é formada por OIDs, as estruturas TCREFOID e TCTEMP referenciam páginas em disco, e quanto maior o tamanho da página utilizada melhor é a utilização da memória, pois cada endereço real armazenado em memória referencia um maior número de OIDs na base de dados. Com mapeamento

indireto, assumiu-se que a TCREFOID e TCTEMP são armazenadas totalmente em memória, porém como trabalho futuro pode ser realizada a análise do crescimento destas estruturas e maneiras de otimizá-las em memória para reduzir a utilização de espaço;

- *Escalabilidade* - com o aumento do número de OIDs, árvores-B crescem em memória e se armazenadas em disco, podem aumentar o tempo de mapeamento. Se forem totalmente armazenadas em memória, o tempo de mapeamento é sempre logarítmico. *Hashing*, por sua vez, tem seu desempenho reduzido com o aumento da base de dados, pois um fator importante para a função *hash* é o conhecimento prévio do tamanho da base de dados. Mapeamento direto, por não utilizar estruturas auxiliares não tem seu desempenho reduzido com o aumento da base de dados, no entanto, a falta de estruturas auxiliares causa o problema de não permitir a relocação das páginas de ponteiros. Assumindo-se o armazenamento das tabelas de indireção em memória, mapeamento indireto possui tempo sempre constante de mapeamento, conforme o aumento da base de dados;
- *Utilização do espaço físico* - neste quesito, a abordagem que possui problemas é *hashing*, pois em média 69% das páginas são ocupadas, ao passo que nas outras abordagens a ocupação pode chegar a 100% (EMBODY; DU, 1988);
- *Controle de concorrência e recuperação* - para o mapeamento direto, uma das tecnologias indicadas é a recuperação baseada em *log* e um controle de concorrência de granularidade fina, porém não são apresentados estudos comparativos. No NUGEM, a TCREFOID é utilizada para a recuperação baseada em páginas *shadow* e o controle de concorrência é feito por bloqueio físico nas páginas utilizadas pelas transações. A cada operação de *commit*, é necessário efetuar a gravação de partes da TCREFOID. Como trabalho futuro, poderiam ser implementadas técnicas de controle de concorrência e recuperação para a abordagem de mapeamento direto e compará-las com a abordagem do NUGEM;

- *Replicação* - todas as abordagens podem possuir seus OIDs replicados em diferentes servidores;
- *Relocação* - a relocação das páginas na base de dados é uma importante operação para funcionalidades como *clustering*, compactação da base de dados e evolução de esquemas (LAKHAMRAJU et al., 2000) (GERLHOF; KEMPER; MOERKOTTE, 1996). Dentre as abordagens discutidas, apenas árvore-B e mapeamento indireto proporcionam relocação total das páginas em disco. Entretanto, entre estas duas abordagens a de mapeamento indireto apresenta uma menor utilização de memória e um menor tempo de mapeamento.

Na figura 25 são resumidas as características de cada abordagem:

	<i>Hashing</i>	Árvore-B	Mapeamento Direto	Mapeamento Indireto
Tempo de Mapeamento	$O(1)$	$O(t * \log_t n)$	$O(1)$	$O(1)$
Utilização de Memória	Não utiliza estruturas auxiliares	Armazena OIDs em memória	Não utiliza estruturas auxiliares	Armazena endereços das páginas em memória
Escalabilidade	Aumenta as colisões	Aumenta a árvore em memória	Não apresenta mudança	Aumenta o uso de memória pelas tabelas de indireção
Utilização do espaço físico	69% das páginas	100% das páginas	100% das páginas	100% das páginas
Concorrência	Não consta	Deve atualizar a árvore, reduz a concorrência	Não apresenta mudança	Deve atualizar tabela de indireção, reduz a concorrência
Recuperação	Não consta	Não consta	Não consta	Paginação <i>shadow</i>
Replicação	Sem problemas	Sem problemas	Sem problemas	Sem problemas
Relocação	Não permite	Permite total relocação	Relocação apenas das páginas de ponteiros	Permite total relocação
SGBDs que utilizam	Versant, Itasca	GemSone, Shore	Não consta	NUGEM

Figura 25: Características das abordagens de *hashing*, árvore-B, mapeamento direto e mapeamento indireto.

3.5 Considerações finais

Neste capítulo foi apresentada a Tabela de Conversão Temporária, bem como suas funcionalidades. O objetivo foi reduzir o número de acessos ao disco e diminuir o tempo de processamento das funcionalidades de gerenciamento das páginas contendo OIDs e dados. Foi discutida a otimização feita nas listas de recuperação e das transações, utilizando-se árvore vermelho-preto.

Já na camada SGLog, foi discutido que as operações de atribuição do máximo contOID e espalhamento dos contOIDs não eram mais necessárias, pois os contOIDs consistentes são referenciados pela TCREFOID. Foi abordado também o esgotamento de OIDs, e concluiu-se que este não é um problema que a abordagem de mapeamento indireto apresenta. Por fim, foi feita a comparação entre a estratégia de mapeamento indireto com mapeamento direto, *hashing* e árvore-B, sendo a estratégia mais completa a de mapeamento indireto. No próximo capítulo, são discutidos os resultados dos testes avaliando-se o tempo de mapeamento e utilização de memória da abordagem proposta neste trabalho.

4 Testes e resultados

4.1 Considerações iniciais

Neste capítulo são apresentados os resultados do desenvolvimento deste trabalho. Inicialmente é comparada a nova abordagem de gerenciamento de OIDs com a antiga abordagem utilizada pelo NUGEM, por meio dos testes realizados nas camadas SGReal e SGLog. Em seguida, é feita a análise do tempo de mapeamento e utilização de memória da abordagem utilizada pelo mapeamento indireto.

A configuração da máquina utilizada para realizar os testes foi um processador Intel Core 2 Duo E7400 2,8 GHz com 3,5 GB de memória RAM e 500 GB de memória secundária.

4.2 Aplicativos de testes

Na figura 26 é ilustrado o aplicativo que foi utilizado para a criação das bases de dados (OLIVEIRA, 2006). Com este aplicativo configura-se diversos parâmetros durante a criação da base de dados.

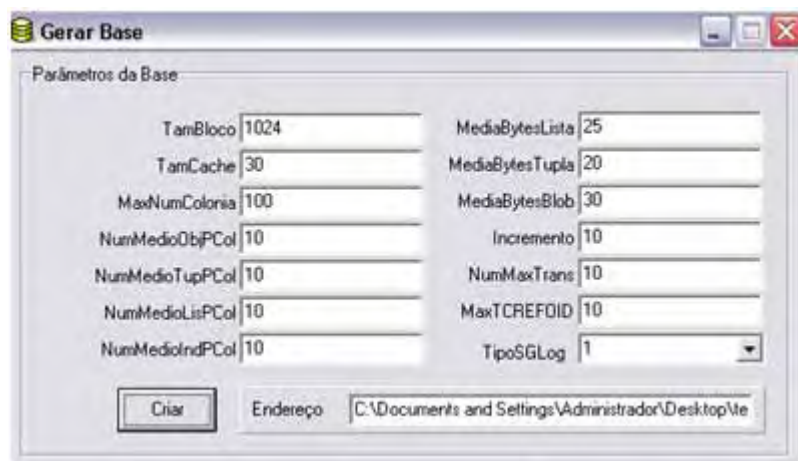


Figura 26: Aplicativo utilizado para a criação das bases de dados (OLIVEIRA, 2006).

Na figura 27 é ilustrado o aplicativo que foi desenvolvido para a realização dos testes na SGReal. Os campos Total e Incremento permitem escolher a quantidade de registros reais que serão criados, sendo que o primeiro campo indica a quantidade de medições, e o último o incremento feito entre cada medição. São permitidas as operações de criação, escrita, leitura, remoção, *commit* e *abort* dos registros reais, e após estas operações é exibido um gráfico de barras, um de linha, entre outras estatísticas como a quantidade de acessos ao disco.

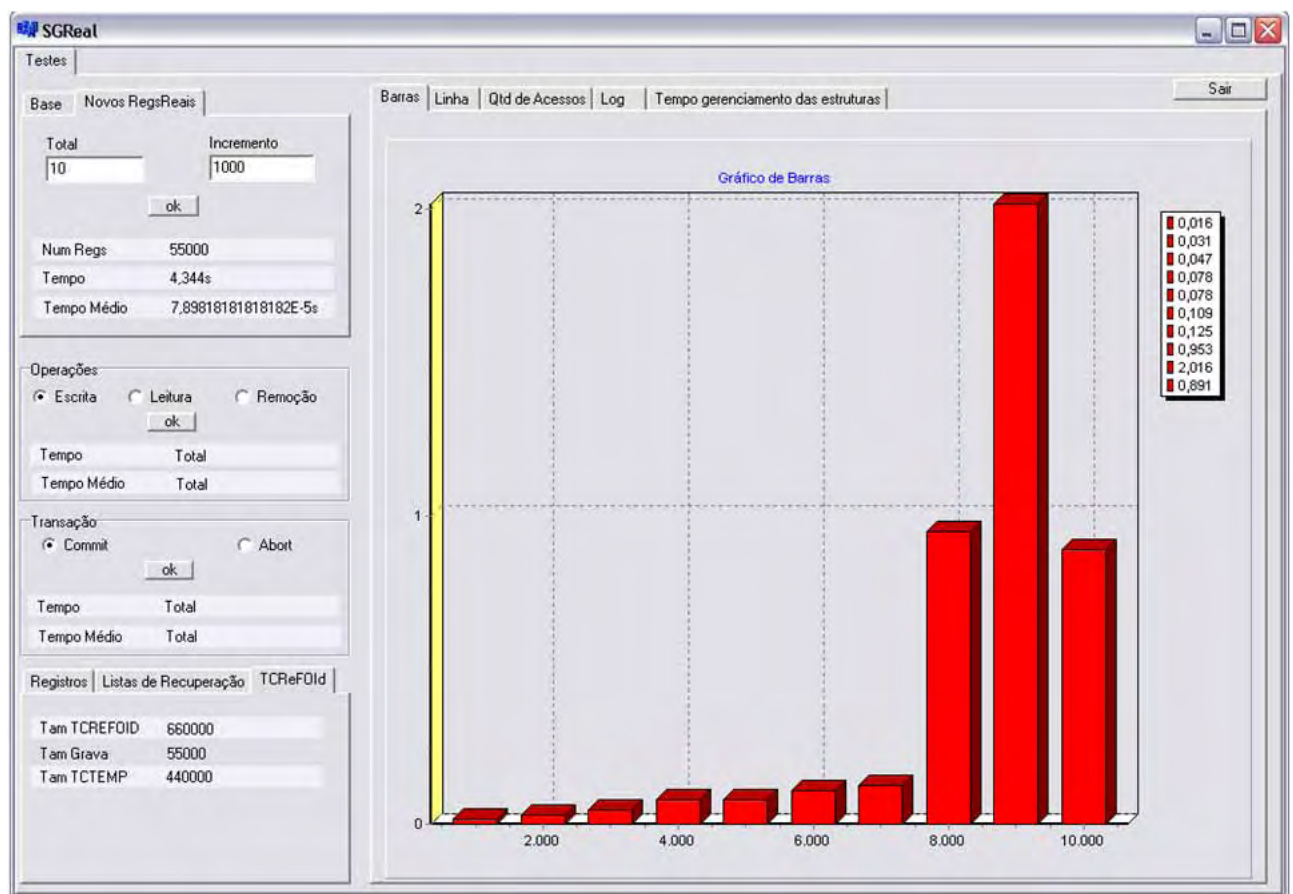


Figura 27: Aplicativo desenvolvido para a realização de testes na SGReal.

Na figura 28 é ilustrado o aplicativo que foi desenvolvido para a realização dos testes na SGLog. São permitidas as operações de criação, escrita, leitura, remoção, *commit* e *abort* dos registros lógicos. São mostrados gráficos de barras e de linha, entre outras estatísticas.

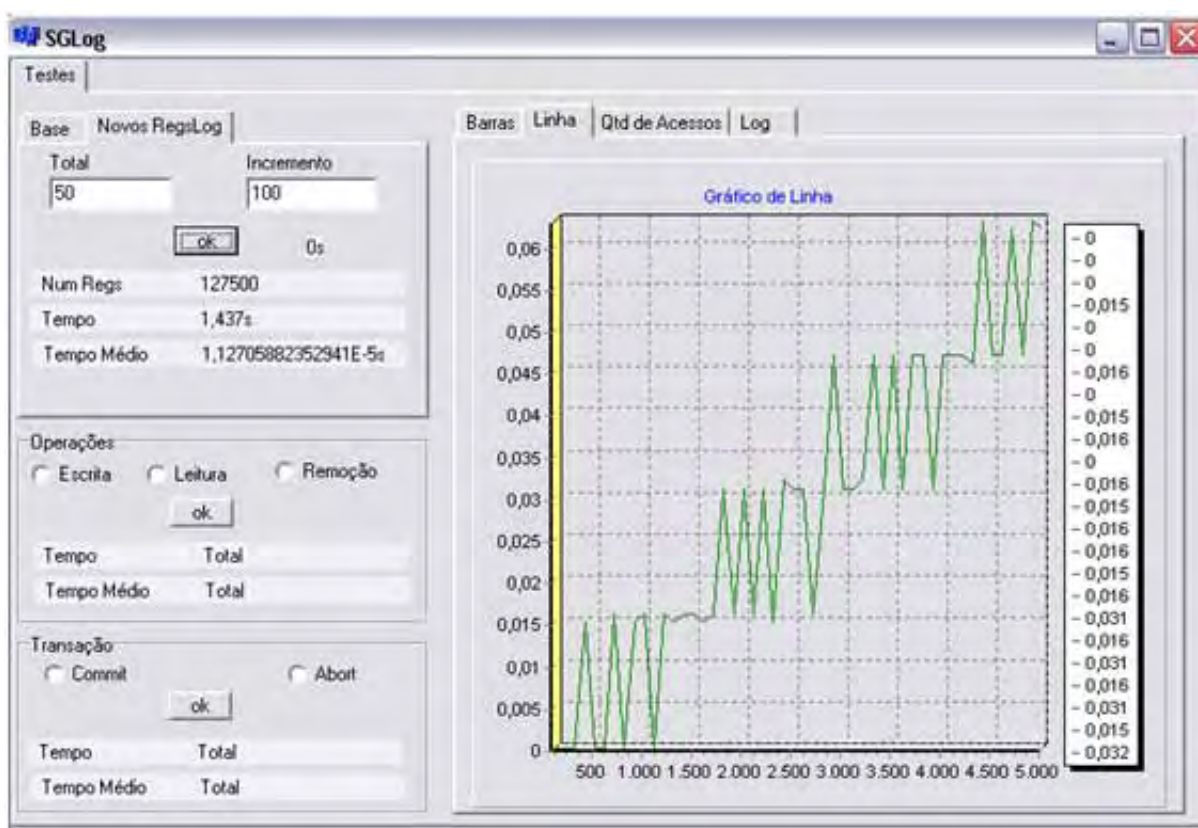


Figura 28: Aplicativo desenvolvido para a realização de testes na SGLog.

4.3 Subsistema de Gerenciamento de Registros Reais

A primeira otimização feita no NUGEM foi a utilização de árvore vermelho-preto para o gerenciamento de suas listas de endereços. Assim, primeiramente são discutidos os testes com as listas de recuperação de registros reais e físicos e com as listas de registros reais requisitados, lidos, escritos e liberados de uma transação. Em seguida, são mostrados os testes com a estrutura TCTEMP. Foi utilizada a seguinte configuração para a realização dos testes:

- *Tipo do registro real* - metade dos registros reais criados foram do tipo objeto e metade do tipo lista, para representar a gravação de um objeto com seus atributos;
- *Tamanho de página* - foi utilizado o tamanho de 1 KB para as páginas da base de dados;

- *Quantidade de registros criados* - as operações foram feitas com a utilização de 38250 registros reais, relativamente um número baixo de registros. Este número foi escolhido pois o desempenho geral do NUGEM estava baixo com a utilização de listas lineares;
- *Operações utilizadas* - foram realizadas as operações de criação, escrita, leitura e remoção de registros reais e as operações de *commit* e *abort* para as transações;
- *Unidade de tempo* - a unidade de tempo utilizada para a apresentação dos dados é de segundos.

4.3.1 Listas de recuperação de registros reais e físicos

Neste teste foi avaliado o desempenho das listas de recuperação com as abordagens de árvore vermelho-preto e lista linear encadeada. Na figura 29, são ilustrados os tempos no NUGEM Antigo, que usava lista linear para a LRRR/LRRF, no NUGEM Atual, com árvore vermelho-preto, a diferença entre os resultados e, por último, o percentual de ganho no desempenho que se obteve com a nova implementação.

LRRR/LRRF com Árvore Vermelho-Preto						
	Criação	Escrita	Leitura	Remoção	Commit	Abort
NUGEM Antigo	27,656 s	18,813 s	11,688 s	25,125 s	41,344 s	14,969 s
NUGEM Atual	4,969 s	13,375 s	8,344 s	4,000 s	10,891 s	0,234 s
Diferença	22,687 s	5,438 s	3,344 s	21,125 s	30,453 s	14,735 s
Percentual	82,033 %	28,906 %	28,611 %	84,080 %	73,658 %	98,437 %

Figura 29: Resultados dos testes com as listas de recuperação da SGReal.

Na figura 30 é ilustrado o gráfico do percentual de tempo ganho com a nova implementação para as listas de recuperação de registros reais e de registros físicos. Nota-se uma melhoria em todas as operações realizadas.

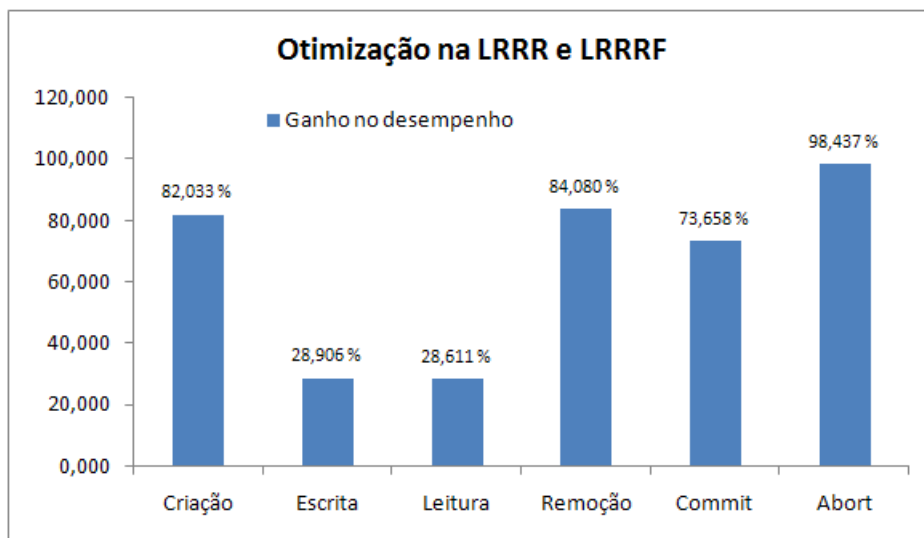


Figura 30: Gráfico do percentual de tempo ganho com a nova implementação para as listas de recuperação.

4.3.2 Listas de registros reais requisitados, lidos, escritos e liberados de uma transação

Neste teste foi feita a comparação das listas de registros reais requisitados, lidos, escritos e liberados de uma transação utilizando-se lista linear e árvore vermelho-preto, conforme ilustrado na figura 31.

Listas das Transações com Árvore Vermelho-Preto						
	Criação	Escrita	Leitura	Remoção	Commit	Abort
Lista	4,969 s	13,375 s	8,344 s	4,000 s	10,891 s	0,234 s
Árvore V.P.	1,188 s	1,203 s	0,329 s	0,187 s	7,219 s	0,203 s
Diferença	3,781 s	12,172 s	8,015 s	3,813 s	3,672 s	0,031 s
Percentual	76,092 %	91,006 %	96,057 %	95,325 %	33,716 %	13,248 %

Figura 31: Resultados dos testes com as listas de registros reais requisitados, lidos, escritos, liberados de uma transação.

Na figura 32 é ilustrado o gráfico do percentual de tempo ganho com a nova implementação, e analisando-se os resultados conclui-se que houve ganho de desempenho em todas as operações, sendo que para leitura e remoção os ganhos foram maiores.

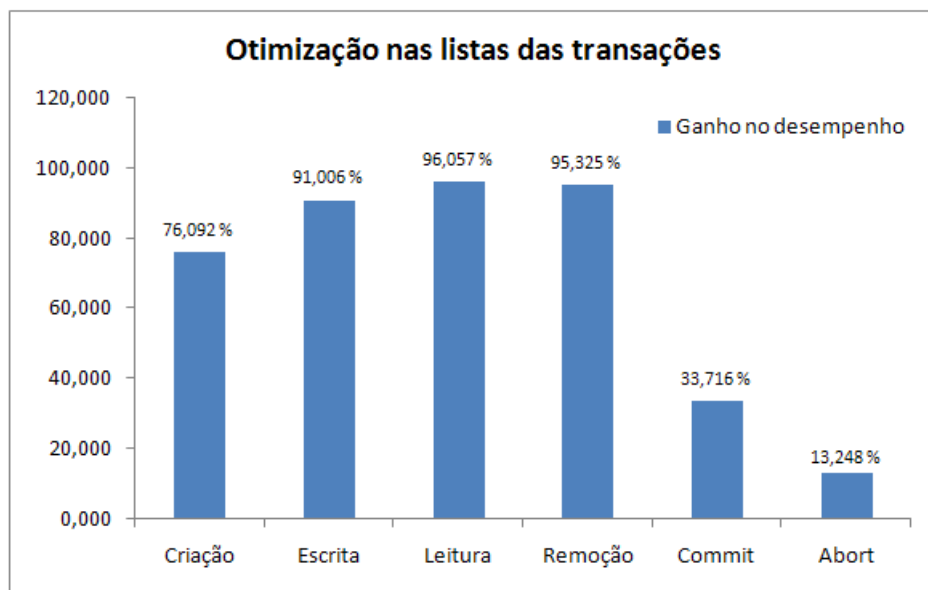


Figura 32: Gráfico do percentual de tempo ganho com a nova implementação para as listas de registros reais requisitados, lidos, escritos e liberados de uma transação.

4.3.3 Tabela de Conversão Temporária

A utilização da TCTEMP permitiu reduzir operações de escrita e leitura no disco para as operações de criação e remoção de registros reais. Os testes realizados a seguir foram feitos com a memória *cache* não populada, ou seja, o número total de acessos representa acessos fisicamente ao disco. Os 38250 registros reais criados foram divididos em metade para o tipo objeto e metade para o tipo genérico. Para os registros reais do tipo objeto foi feita a divisão para os três cenários possíveis da funcionalidade de criar um registro real:

- *O registro real nunca foi utilizado* - foram criados 6375 registros reais utilizando-se endereços reais que nunca haviam sido usados. Para este caso as operações de escrita no disco foram 6375 e de leitura foram 0;
- *O registro real foi removido por uma transação já efetivada na base de dados* - foram criados 6375 registros reais a partir de endereços reais que já haviam sido removidos e efetivados por outras transações, por meio da operação de *commit*. Neste caso foram realizadas 6375 operações de leitura no disco e 6375 de escrita;

- *O registro real foi removido pela transação atual* - foram criados 6375 registros reais utilizando-se endereços reais que haviam sido removidos pela própria transação, porém que não foram efetivados na base de dados. Neste cenário não foram realizadas operações de escrita e nem de leitura no disco.

Já para a criação dos registros reais genéricos, foram considerados os dois casos possíveis:

- *Registro real já foi removido* - foram criados 9562 registros reais com endereços reais que já haviam sido removidos. Neste cenário não foram realizadas operações de leitura e nem de escrita no disco;
- *Registro real novo* - foram criados 9563 registros reais utilizando-se novos endereços reais. Neste cenário não foram realizadas operações de leitura e nem de escrita no disco.

No gráfico da figura 33 é ilustrada a redução do número de operações de escrita e leitura no disco para a operação de se criar um novo registro real.

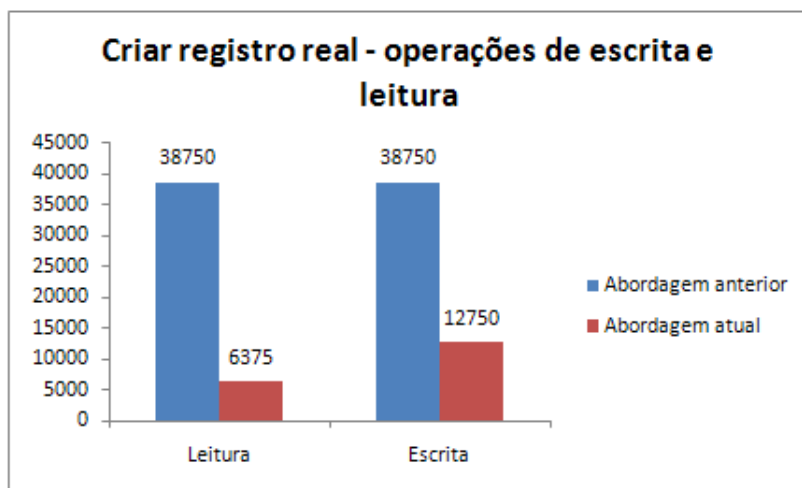


Figura 33: Redução da operação de escrita na funcionalidade de criar novos registros reais.

Para o teste de remoção os registros reais foram divididos entre os tipos genérico e objeto. No entanto, esta distinção não altera o desempenho em relação ao número de acessos ao disco. Conforme ilustrado na figura 34, nota-se uma redução significativa nas operações leitura, enquanto as operações de escrita se mantiveram nulas.

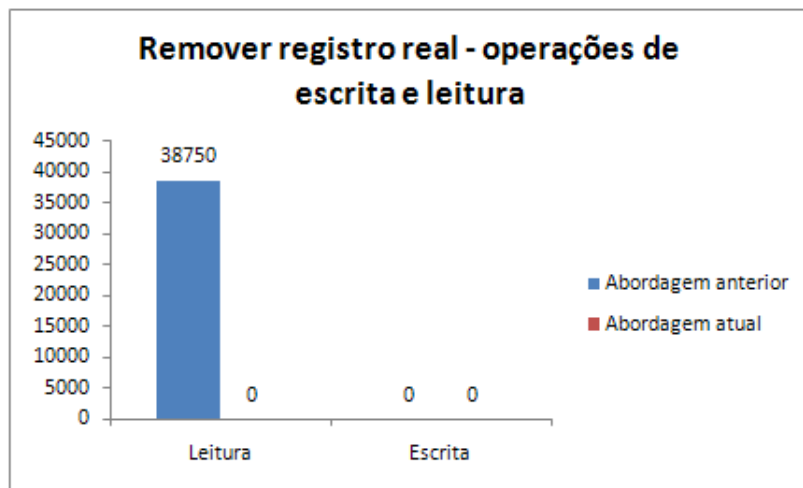


Figura 34: Redução da operação de leitura na funcionalidade de remover registro real.

A estrutura de dados TCTEMP proporcionou, para a criação de um novo registro real, a redução de 83,55% de operações de escrita em disco e 67,1% de leitura. Para a remoção de registros reais, as operações de leitura foram reduzidas em 100%, enquanto as operações de escrita se mantiveram nulas.

Os tempos de processamento para as operações de criação, escrita, remoção, *commit* e *abort* dos registros reais podem ser visualizados na figura 35.

TCTEMP						
	Criação	Escrita	Leitura	Remoção	Commit	Abort
Sem TCTEMP	0,156 s	1,094 s	0,281 s	0,157 s	7,250 s	0,204 s
Com TCTEMP	0,094 s	1,078 s	0,277 s	0,125 s	7,266 s	0,204 s
Diferença	0,062 s	0,016 s	0,004 s	0,032 s	-0,016 s	0,000 s
Percentual	39,744 %	1,463 %	1,423 %	20,382 %	-0,221 %	0,000 %

Figura 35: Resultados dos testes utilizando a TCTEMP.

De acordo com gráfico da figura 36, as operações que tiveram seu desempenho melhorados significativamente foram as de criação e remoção de registros reais. As operações de leitura e escrita tiveram um pequeno ganho de desempenho, ao passo que a operação de *commit* teve alteração desprezível e *abort* não teve alteração.

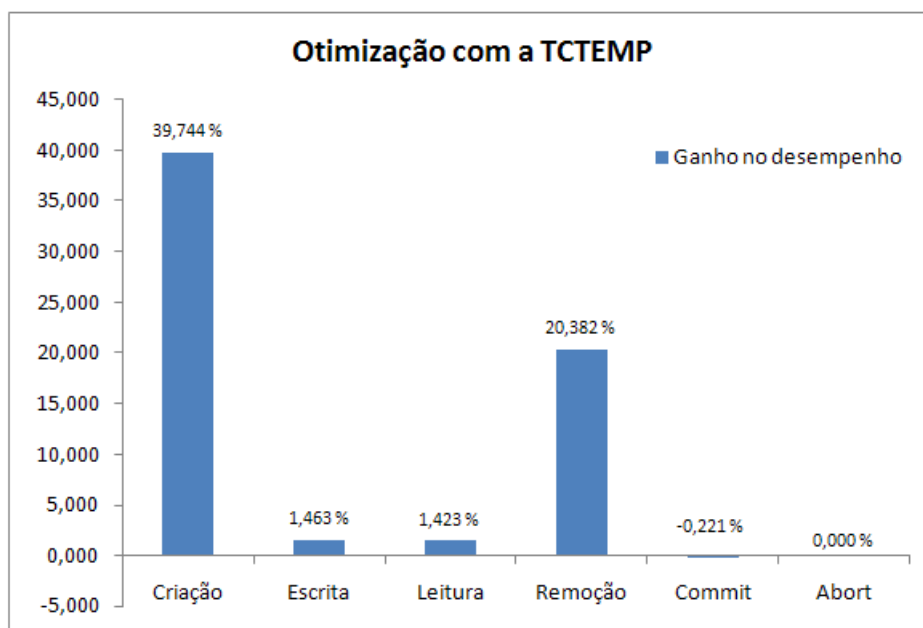


Figura 36: Gráfico do percentual de tempo ganho utilizando a TCTEMP.

Nota-se que os resultados utilizando-se TCTEMP, quando comparados com a abordagem antiga de gerenciamento de OIDs do NUGEM, se mostraram superiores em relação ao número de acessos ao disco e tempo de processamento.

4.4 Subsistema de Gerenciamento de Registros Lógicos

Na SGLog foram realizados testes para a lista de recuperação de registros lógicos e do tempo utilizado para a busca do máximo contOID e espalhamento de contOIDs. Foram utilizados os seguintes parâmetros:

- *Tipo lógico* - foi usado o tipo lógico objeto para os registros lógicos;
- *Tamanho de página* - o tamanho de página foi de 1 KB para as páginas em disco;
- *Quantidade de registros* - no total foram criados um milhão de registros lógicos, o que representa um milhão de OIDs;
- *Operações* - foram realizadas as operações de criação, escrita, leitura e remoção de registros lógicos e as operações de *commit* e *abort* para as transações;

- *Unidade de tempo* - a unidade de tempo utilizada para a apresentação dos dados é de segundos.

4.4.1 Lista de recuperação de registros lógicos

Neste teste foi comparado o desempenho da SGLog utilizando-se lista linear e árvore vermelho-preto para a lista de recuperação de registros lógicos, conforme ilustrado na figura 37.

Lista de recuperação Linear x Árvore Vermelho-Preto						
	Criação	Escrita	Leitura	Remoção	Commit	Abort
Lista linear	12,500 s	8,641 s	4,922 s	46,984 s	9,844 s	0,406 s
Árvore Vermelho-Preto	11,890 s	8,594 s	4,875 s	13,515 s	9,812 s	0,407 s
Diferença	0,610 s	0,047 s	0,047 s	33,469 s	0,032 s	-0,001 s
Percentual	4,880 %	0,544 %	0,955 %	71,235 %	0,325 %	-0,246 %

Figura 37: Resultados dos testes com lista linear e árvore vermelho-preto para a lista de recuperação de registros lógicos.

Por meio da figura 38 nota-se que a operação que teve maior ganho de desempenho foi a de remoção de registros lógicos, seguida pela operação de criação, enquanto as demais operações praticamente mantiveram o mesmo desempenho.

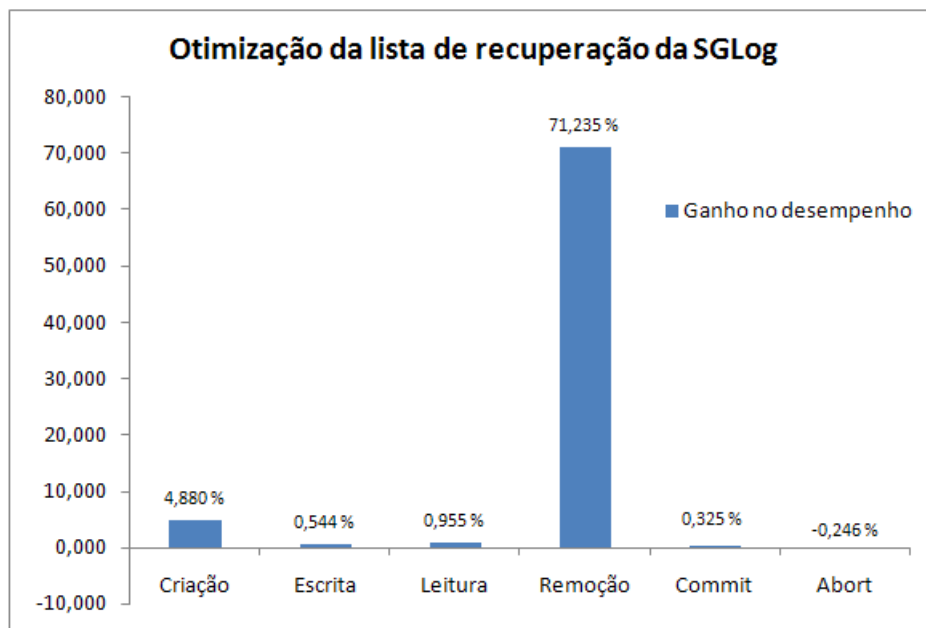


Figura 38: Gráfico do percentual de tempo ganho utilizando árvore vermelho-preto na SGLog.

4.4.2 Tempos utilizados para a busca do máximo contOID e espalhamento de contOIDs

A medição dos tempos utilizados para a busca do máximo contOID e espalhamento de contOIDs permitiu avaliar o custo necessário para executar estas operações nas funcionalidades de criação e remoção de registros lógicos. Na figura 39 são ilustrados os resultados da medição do tempo utilizado para o espalhamento e busca pelo máximo contOID.

Criação	Espalhamento	Porcentagem	Remoção	Busca Max	Porcentagem
11,656 s	0,015 s	0,129 %	13,375 s	0,015 s	0,112 %

Figura 39: Medição dos tempos utilizados para a busca do máximo contOID e espalhamento de contOIDs.

Os tempos utilizados para o espalhamento e busca pelo máximo contOID são baixos comparados aos tempos totais de criação e remoção de registros lógicos do tipo objeto, de acordo com a figura 40.

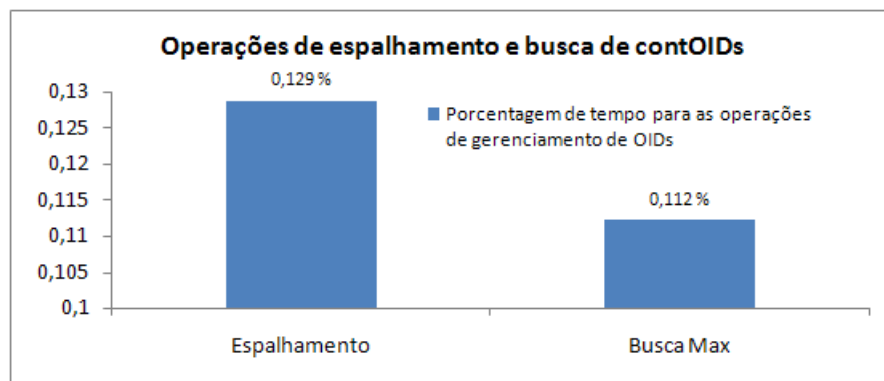


Figura 40: Porcentagem do tempo utilizado para as operações de espalhamento e busca pelo máximo contOID.

4.5 Avaliação do tempo de mapeamento e utilização de memória

A medição do tempo de mapeamento e utilização de memória pela abordagem de mapeamento indireto foi realizada na SGReal considerando-se os seguintes parâmetros:

- *Quantidade de registros reais* - foram criados 350000 registros reais do tipo objeto;
- *Tamanho de página* - foram realizados dois testes, o primeiro com o tamanho de 1 KB e o segundo de 4 KB para os registros físicos;
- *Operações utilizadas* - foram executadas as funcionalidades de criação, escrita, leitura e remoção de registros reais e as operações de *commit* e *abort* para as transações;
- *Medição do tempo de mapeamento* - foi medido o tempo utilizado para a manipulação da TCREFOID e da TCTEMP, que envolveu as operações de mapeamento do endereço real para a sua respectiva entrada na TCREFOID e TCTEMP, bem como as operações de comparação e atribuição envolvendo estas estruturas. Esta aferição foi representada nas colunas nomeadas de "Mapeamento";
- *Porcentagem dos tempos de mapeamento, gravação e inicialização* - foram apresentados também a porcentagem destes tempos em relação ao tempo total das operações realizadas;

- *Unidade de tempo* - a unidade de tempo utilizada para a apresentação dos dados é a de segundos.

4.5.1 Testes com páginas de tamanho 1KB

Neste teste foi utilizado o tamanho de 1 KB para as páginas em disco, o que permitiu armazenar 22 registros lógicos do tipo objeto por página, totalizando 7,7 milhões de OIDs gerados. Na figura 41 é ilustrado o tempo para as operações e mapeamento com páginas de 1 KB. A coluna tempo indica o tempo total da operação realizada, a coluna mapeamento indica o tempo de mapeamento e a coluna porcentagem indica o percentual de tempo utilizado para o mapeamento em relação ao tempo total da operação.

	Tempo	Mapeamento
Criação	10,402 s	0,024 s
Escrita	8,066 s	0,139 s
Leitura	2,496 s	0,142 s
Remoção	1,333 s	0,044 s
Commit	13,222 s	0,067 s
Abort	3,458 s	0,074 s
Total	38,977 s	0,490 s

Figura 41: Tempo para as operações e mapeamento com páginas de 1 KB.

As operações de leitura e escrita foram as que mais consumiram tempo de mapeamento, conforme ilustrado na figura 42.

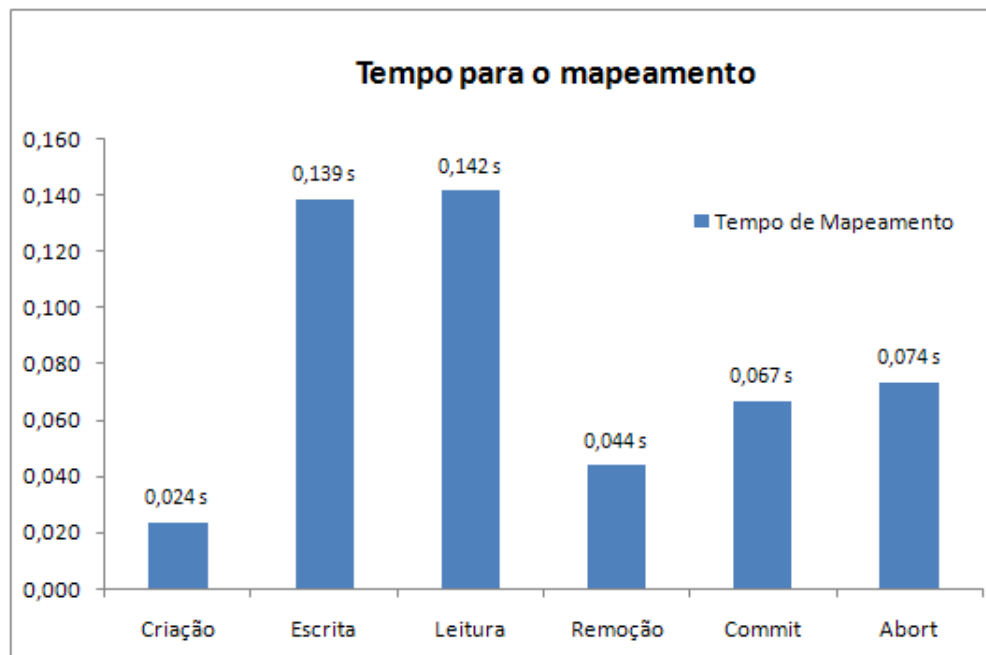


Figura 42: Tempo para o mapeamento com páginas de 1 KB.

Durante a inicialização da base de dados é necessário ler as colunas CSOID e FIS da TCREFOID do disco, e ao final da operação de *commit* é necessário gravá-las em disco. Estes tempos de inicialização e gravação são ilustrados na figura 43.

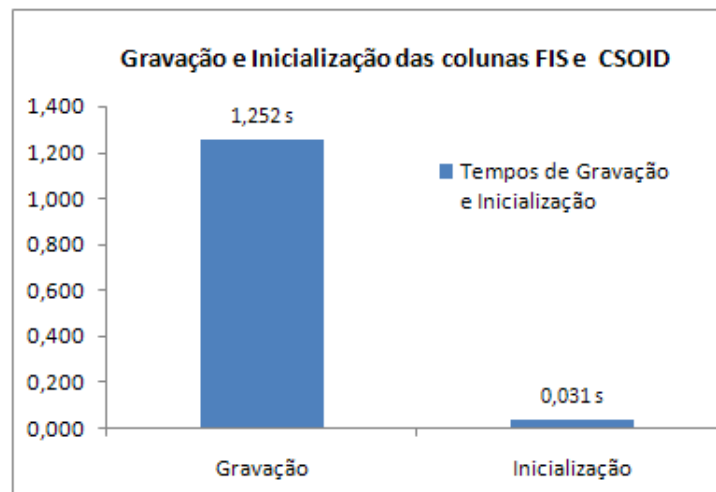


Figura 43: Tempo de gravação e inicialização da coluna CSOID com páginas de 1 KB.

Somando-se os tempos de mapeamento e os tempos de gravação e inicialização, tem-se o tempo total para o gerenciamento dos 7,7 milhões OIDs, como é ilustrado na figura 44.

Tempo das operações	Gerenciamento dos OIDs	Porcentagem
38,977 s	1,773 s	4,549 %

Figura 44: Tempo total de gerenciamento dos 7,7 milhões de OIDs.

Além do tempo de processamento utilizado para o mapeamento, tem-se também outro custo, que é a utilização de memória. Para o gerenciamento de OIDs, são utilizados 16 bytes em memória para cada registror real, sendo 8 bytes na TCREFOID e 8 na TCTEMP. Para este teste, a utilização total de memória foi de 5,34 MB, como é ilustrado na figura 45.

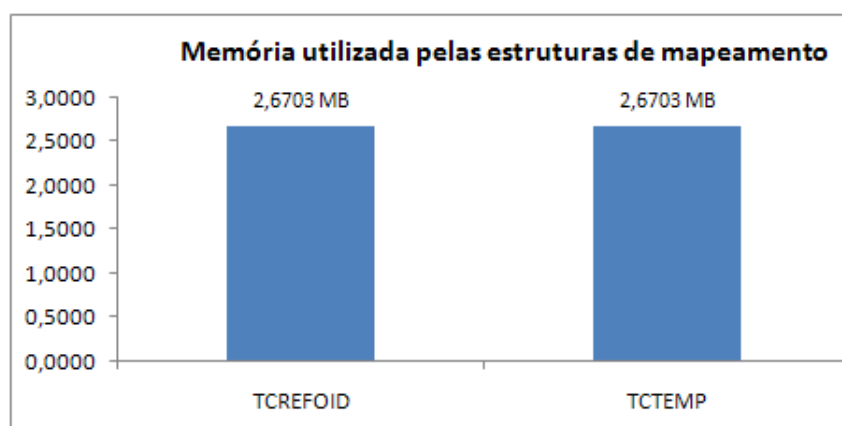


Figura 45: Utilização de memória principal para páginas de 1 KB.

O tempo de processamento para o gerenciamento dos 7,7 milhões de OIDs foi de 4,549% do tempo total de processamento durante as operações realizadas, e a utilização de memória foi de 5,34 MB.

4.5.2 Testes com páginas de tamanho 4 KB

Neste teste foi utilizado o tamanho de 4 KB para as páginas em disco, o que permitiu armazenar 90 registros lógicos do tipo objeto por página, totalizando 31,5 milhões de OIDs gerados. Na figura 46 é ilustrado o tempo para as operações e mapeamento com páginas de 4 KB.

	Tempo	Mapeamento
Criação	17,566 s	0,038 s
Escrita	15,442 s	0,169 s
Leitura	3,196 s	0,152 s
Remoção	1,330 s	0,031 s
Commit	6,036 s	0,065 s
Abort	3,413 s	0,091 s
Total	46,983 s	0,546 s

Figura 46: Tempo para as operações e mapeamento com páginas de 4 KB.

Assim como para páginas de 1 KB, as operações de leitura e escrita foram as que mais consumiram tempo de mapeamento, como é ilustrado na figura 47.

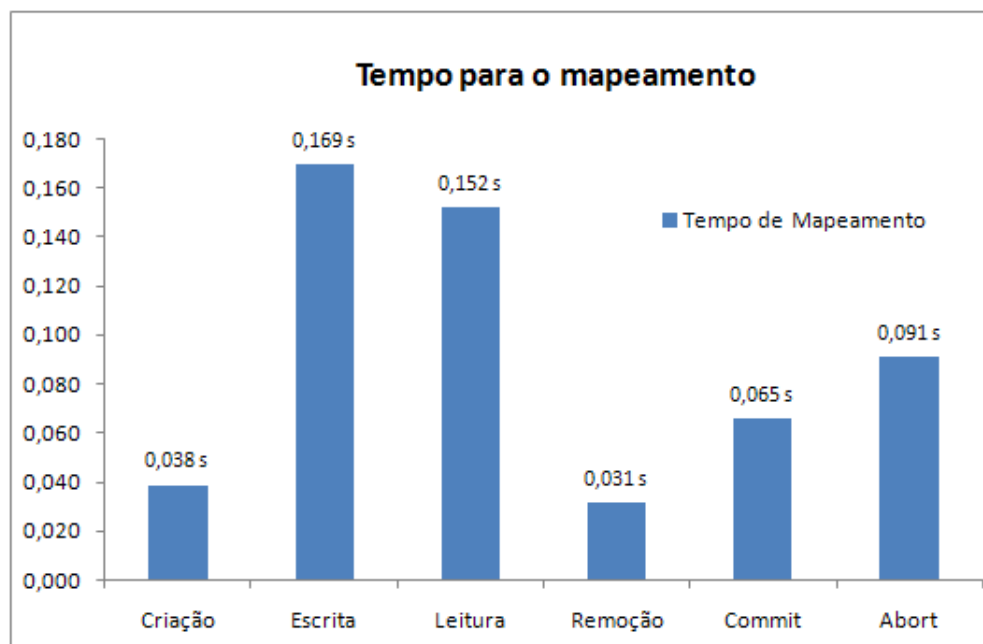


Figura 47: Tempo para o mapeamento com páginas de 4 KB.

Os tempos de inicialização e gravação das colunas FIS e CSOID são ilustrados na figura 48. Estes tempos são menores do que os tempos para páginas de 1 KB, devido ao fato de que usando-se páginas de 4 KB mais dados são lidos do disco a cada acesso realizado.

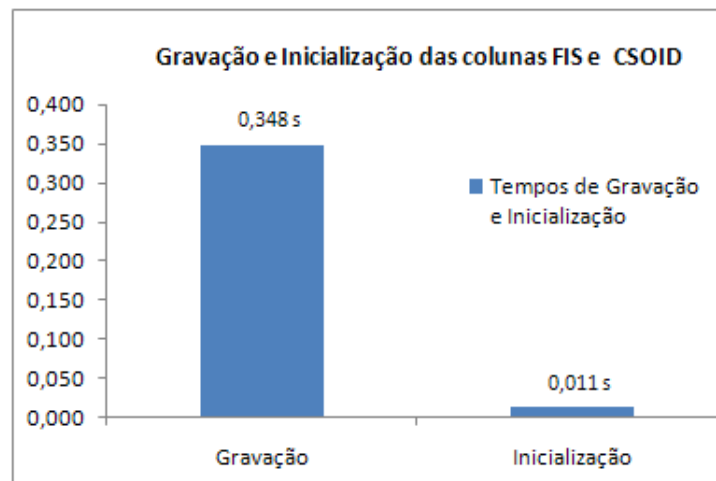


Figura 48: Tempo de gravação e inicialização das colunas CSOID e FIS com páginas de 4 KB.

Somando-se os tempos de mapeamento e os tempos de criação e inicialização, tem-se o tempo total para o gerenciamento dos 31,5 milhões de OIDs, como é ilustrado na figura 49.

Tempo das operações	Gerenciamento dos OIDs	Porcentagem
46,983 s	0,905 s	1,926 %

Figura 49: Tempo total de gerenciamento dos 31,5 milhões de OIDs.

A utilização de memória para páginas de 4 KB e de 1 KB foram as mesmas, conforme ilustrado na figura 50.

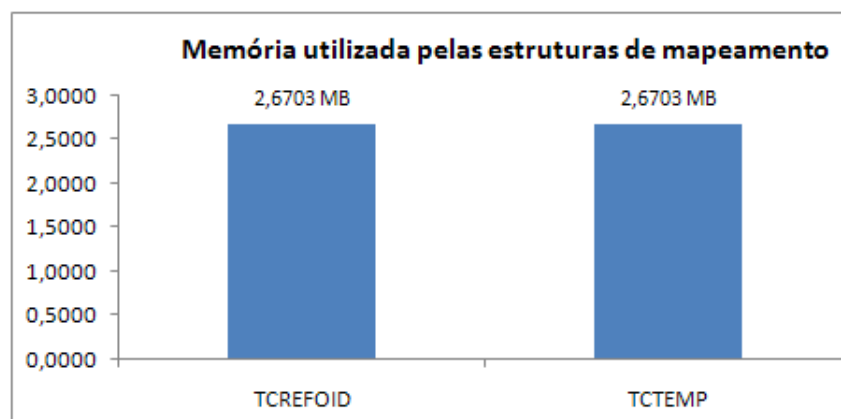


Figura 50: Utilização de memória principal para páginas de 4 KB.

Utilizando-se a mesma quantidade de memória que a abordagem de 1 KB foi possível gerenciar um número quatro vezes maior de OIDs. Além disso, o tempo para a inicialização

e gravação da TCREFOID foi menor com páginas de 4 KB, reduzindo o tempo total de gerenciamento dos OIDs, fazendo com que para a manipulação dos 31,5 milhões de OIDs utilizasse 1,926% do tempo de processamento total do teste realizado.

A utilização de memória e o custo de 1,926% de mapeamento aferido nos testes com páginas de 4 KB não são apresentados pela abordagem de mapeamento direto. Entretanto, a um baixo custo de processamento e utilização de memória, tem-se uma abordagem para o gerenciamento de OIDs que se tenha complexidade constante para o mapeamento dos OIDs e ao mesmo tempo permita a flexibilidade de relocação de todas as páginas da base de dados, suportando operações de *clustering*, *garbage collection*, compactação e qualquer outra operação que exija a relocação de páginas em disco.

4.6 Considerações finais

Inicialmente foram apresentados os aplicativos utilizados para a realização dos testes. Em seguida, foi avaliado o ganho de desempenho em termos de processamento e acessos ao disco da nova abordagem em relação a abordagem utilizada anteriormente para o gerenciamento de OIDs no NUGEM. Por fim, foram avaliados o tempo de mapeamento e utilização de memória pela estratégia de mapeamento indireto para páginas de 1 KB e de 4 KB. Com páginas de 4 KB, o total de operações de mapeamento foi de apenas 1,926% do tempo total das operações realizadas.

5 *Conclusões e trabalhos futuros*

Foram discutidos os principais conceitos e técnicas referentes à área de identificadores de objetos em sistemas gerenciadores de bancos de dados não convencionais. Pôde-se notar uma carência na literatura em relação às abordagens de gerenciamento de OIDs, sendo as técnicas mais relevantes as de árvore-B, *hashing* e mapeamento direto. Um trabalho mais recente tratou do problema de esgotamento de OIDs (SHI et al., 2006), problema que também foi avaliado para o mapeamento indireto.

Este trabalho contribuiu para a técnica já utilizada no NUGEM (VALÊNCIO, 2000), uma vez que reduziu o número de acessos ao disco e o tempo de processamento durante a manipulação de identificadores, caracterizando-a como uma nova abordagem para o gerenciamento de OIDs lógicos. Embora desenvolvida e testada no NUGEM, esta abordagem pode ser utilizada por outras arquiteturas de SGBDs que façam uso das estruturas e funcionalidades aqui descritas. Os objetivos atingidos neste trabalho foram:

- *Proposta de uma nova estrutura para gerenciamento de identificadores de objetos* - uma estrutura de dados foi desenvolvida, representada por uma tabela com duas colunas, as quais referenciam páginas em disco contendo OIDs, bem como dados. Foi também realizada uma nova abordagem para a coluna CSOID da TCREFOID, que passou a referenciar páginas na base de dados contendo contOIDs consistentes. Além disso, foram otimizadas as listas usadas pelas funcionalidades do NUGEM para o gerenciamento de endereços, com a utilização de árvore vermelho-preto.

Os resultados dos testes comprovaram a eficiência proporcionada pela estrutura

proposta (TCTEMP), que reduziu o número de acessos ao disco nas operações de criação e remoção de páginas. Para a criação de uma nova página, as operações de escrita em disco foram reduzidas em 83,55% e as de leitura em 67,1%. Para a remoção de páginas, as operações de leitura foram reduzidas em 100%, enquanto as operações de escrita se mantiveram nulas. Em relação ao tempo de processamento das páginas de OIDs e de dados, obteve-se 39,74% de ganho de desempenho para a operação de criação e 20,38% para a operação de remoção. Os testes na camada lógica do NUGEM, a SGLog, comprovaram que o ganho de desempenho eliminando-se as operações de busca pelo máximo e espalhamento dos contOIDs, embora exista, não foi acentuado. Notou-se também um ganho de desempenho no gerenciamento das listas de endereços e de transações, que foram otimizadas com árvore vermelho-preto;

- *Avaliação do problema de esgotamento de OIDs* - foi proposta uma fórmula simples, porém que permite avaliar o esgotamento de OIDs para a abordagem de mapeamento indireto. Se cada endereço lógico fosse removido um milhão de vezes por dia, seriam necessários quatro mil anos para se esgotar o seu OID. Assim, concluiu-se que o esgotamento de OIDs não representa um problema para a abordagem de mapeamento indireto;
- *Comparação com as abordagens hashing, árvore-B e mapeamento direto* - dentre as abordagens propostas na literatura, mapeamento indireto se mostrou a mais completa. Enquanto *hashing* apresenta o problema de colisões e baixa ocupação das páginas em disco, mapeamento direto não permite a relocação total de suas páginas e árvore-B utiliza muita memória e possui tempo de mapeamento logarítmico, a abordagem de mapeamento indireto não apresenta colisões, possui baixa utilização de memória, tempo de mapeamento constante e as páginas podem ter ocupação de 100% e ser relocadas livremente. Nos testes realizados com páginas de 4 KB, constatou-se que o tempo de mapeamento foi de 1,926% do tempo total das operações realizadas com os OIDs. Este percentual pode ser visto como o custo adicional à

abordagem de mapeamento direto para permitir-se a relocação das páginas na base de dados, pois as abordagens de mapeamento direto e indireto utilizam o conceito de contador para cada endereço de objeto, com o diferencial da estratégia de mapeamento indireto usar tabelas de indireção em memória (TCREFOID e TCTEMP).

Com base nas estruturas e funcionalidades da abordagem de mapeamento indireto, diversas outras propostas podem ser desenvolvidas abrangendo áreas não aprofundadas neste trabalho. Algumas sugestões são:

- *Suporte distribuído ao gerenciamento de OIDs* - para isso seria necessário incluir mais um campo no OID indicando a base de dados do objeto, para que cada objeto possa ser identificado unicamente (EICKLER; KEMPER; KOSSMANN, 1997). Seria também necessário efetuar um estudo das estruturas e demais funcionalidades do NUGEM para adequá-las ao suporte distribuído;
- *Estudos comparativos com técnicas de controle de concorrência e recuperação* - outra proposta seria implementar diferentes controles de concorrência e recuperação e compará-los nas técnicas de mapeamento direto e mapeamento indireto;
- *Otimização da utilização de memória* - pode-se avaliar como as estruturas de dados TCREFOID e TCTEMP poderiam ser otimizadas para ocupar menos espaço em memória;
- *Comparação das listas de endereços com outras estruturas de dados* - neste trabalho considerou-se árvores vermelho-preto para a representação das listas do NUGEM. Entretanto, outras estruturas poderiam ser implementadas e avaliadas, como por exemplo árvore-B;
- *Avaliação do problema da linearidade na utilização dos endereços físicos* - a estratégia de mapeamento indireto permite a utilização linear para o subconjunto de endereços físicos armazenados na LRRF. Embora o subconjunto de endereços armazenado na TCTEMP também seja utilizado, ele não é feito de forma linear para

obter-se um desempenho melhor durante a execução das transações. Poderiam ser propostas estratégias alternativas para se assegurar a linearidade completa da utilização dos endereços físicos, e, ao mesmo tempo, garantir-se um bom desempenho do gerenciamento de OIDs.

Referências

- ABITEBOUL, S.; KANELLAKIS, P. C. Object identity as a query language primitive. *J. ACM*, ACM, New York, NY, USA, v. 45, n. 5, p. 798–842, 1998.
- BERLER, M.; EASTMAN, J.; JORDAN, D.; RUSSELL, C.; SCHADOW, O.; STANIENDA, T.; VELEZ, F. *The object data standard: ODMG 3.0*. 1. ed. San Francisco, CA, USA: Morgan Kaufmann, 2000.
- BRAUMANDL, R.; CLAUSSEN, J.; KEMPER, A.; KOSSMANN, D. Functional-join processing. *The VLDB Journal*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, v. 8, n. 3-4, p. 156–177, 2000.
- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. *Algoritmos: Teoria e Prática*. 2. ed. Rio de Janeiro: Campus, 2002.
- DEUX, O. The story of o2. *IEEE Trans. on Knowl. and Data Eng.*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 2, n. 1, p. 91–108, 1990.
- EICKLER, A.; GERLHOF, C. A.; KOSSMANN, D. A performance evaluation of oid mapping techniques. In: *VLDB '95: Proceedings of the 21th International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995. p. 18–29.
- EICKLER, A.; KEMPER, A.; KOSSMANN, D. Finding data in the neighborhood. In: *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1997. p. 336–345.
- ELMASRI, E. R.; NAVATHE, S. *Sistemas de Bancos de Dados*. 4. ed. São Paulo: Addison Wesley, 2005.
- EMBODY, R. J.; DU, H. C. Dynamic hashing schemes. In: *ACM Computing Surveys*. [S.l.: s.n.], 1988. p. 85–113.
- FERRIZZI, A. C. *Suporte ao gerenciamento de objetos e esquemas junto ao Núcleo Gerenciador de Dados Multimídia*. 63 f. Monografia (Bacharelado em Ciência da Computação) — Instituto de Biociências, Letras e Ciências Exatas, Universidade Estadual Paulista Júlio de Mesquita Filho, São José do Rio Preto, São Paulo, 2006.
- FERRIZZI, A. C.; JARDINI, T.; COSTA, L. R.; COLOMBO, J.; MAUAD, E. C.; KERR, L. M.; HIDALGO, G. S.; RAHAL, P.; VALÊNCIO, C. R. A clinical decision support system for cancer diseases. In: *EATIS '08: Proceedings of the 2008 Euro American Conference on Telematics and Information Systems*. Sergipe, Brasil: ACM, 2008. p. 1–4.

- FERRIZZI, A. C.; JARDINI, T.; COSTA, L. R.; COLOMBO, J.; RAHAL, P.; VALÊNCIO, C. R.; MAUAD, E. C.; KERR, L. M.; HIDALGO, G. S. A medical information system to manage a cancer database. In: *ICSOF (ISDM/ABF)*. Porto, Portugal: [s.n.], 2008. p. 268–271.
- GERLHOF, C. A.; KEMPER, A.; MOERKOTTE, G. On the cost of monitoring and reorganization of object bases for clustering. *SIGMOD Rec.*, ACM, New York, NY, USA, v. 25, n. 3, p. 22–27, 1996.
- GOUTIER, A. S. *Gerenciamento de Sobrestantes no Núcleo Gerenciador de Dados Multimídia*. 55 f. Monografia (Bacharelado em Ciência da Computação) — Instituto de Biociências, Letras e Ciências Exatas, Universidade Estadual Paulista Júlio de Mesquita Filho, São José do Rio Preto, São Paulo, 2004.
- KHOSHAFIAN, S.; COPELAND, G. P. Object identity. In: *OOPSLA*. Portland, Oregon, USA: [s.n.], 1986. p. 406–416.
- KHOSHFIAN, S.; BAKER, A. B. *Multimedia and Imaging Databases*. 1. ed. San Francisco, CA, USA: Morgan Kaufmann, 1995.
- LAKHAMRAJU, M. K.; RASTOGI, R.; SESHADRI, S.; SUDARSHAN, S. On-line reorganization in object databases. *SIGMOD Rec.*, ACM, New York, NY, USA, v. 29, n. 2, p. 58–69, 2000.
- LEE, W.-C.; LEE, D. L. On processing nested queries in distributed object-oriented database systems. In: *RIDE '95: Proceedings of the 5th International Workshop on Research Issues in Data Engineering-Distributed Object Management (RIDE-DOM'95)*. Washington, DC, USA: IEEE Computer Society, 1995. p. 10.
- LIMA, W. d. S. *Suporte à estratégia de clustering para o Núcleo Gerenciador de Dados Multimídia*. 99 f. Monografia (Bacharelado em Ciência da Computação) — Instituto de Biociências, Letras e Ciências Exatas, Universidade Estadual Paulista Júlio de Mesquita Filho, São José do Rio Preto, São Paulo, 2006.
- NØRVÅG, K. The vagabond temporal oid index: An index structure for oid indexing in temporal object database systems. In: *IDEAS '00: Proceedings of the 2000 International Symposium on Database Engineering & Applications*. Washington, DC, USA: IEEE Computer Society, 2000. p. 158–166.
- OLIVEIRA, W. D. *Benchmark para um Núcleo Gerenciador de Dados Multimídia*. 70 f. Monografia (Bacharelado em Ciência da Computação) — Instituto de Biociências, Letras e Ciências Exatas, Universidade Estadual Paulista Júlio de Mesquita Filho, São José do Rio Preto, São Paulo, 2006.
- ORACLE. *Oracle Database*. Disponível em: <<http://www.oracle.com>>. Acesso em: 5 mar. 2010.
- SHI, Y.; ZHAI, B.; ZHOU, X.; PENG, Z. Object identifier reuse mechanism for totem. In: *ICDEW '06: Proceedings of the 22nd International Conference on Data Engineering Workshops*. Washington, DC, USA: IEEE Computer Society, 2006. p. 127.

- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. *Sistema de banco de dados*. 3. ed. São Paulo: Makron Books, 1999.
- SOCKUT, G. H.; IYER, B. R. Online reorganization of databases. *ACM Comput. Surv.*, ACM, New York, NY, USA, v. 41, n. 3, p. 1–136, 2009.
- SOUSA P., S. A. M. J. Object identifiers and identity: a naming issue. In: *Fourth International Workshop on Object-Orientation in Operating Systems*. Los Alamitos, CA, USA: [s.n.], 1995. p. 127–129.
- STEIN, L. D.; THIERRY-MIEG, J. Acedb: A genome database management system. *Computing in Science and Engineering*, IEEE Computer Society, Los Alamitos, CA, USA, v. 1, p. 44–52, 1999.
- VALÊNCIO, C. R. *Núcleo Gerenciador de Objetos - Compatibilizando Eficiência e Flexibilidade*. 149 f. Tese (Doutorado em Ciências: Física Aplicada - opção Física Computacional) — Instituto de Física de São Carlos, Universidade de São Paulo, São Carlos, 2000.
- VIARA, E.; BARILLOT, E.; VAYSSEIX, G. The eyedb oodbms. In: *IDEAS '99: Proceedings of the 1999 International Symposium on Database Engineering & Applications*. Washington, DC, USA: IEEE Computer Society, 1999. p. 390.
- WATANABE, L. S.; PEREIRA, R. F. *Otimização das estruturas e benchmark para o Núcleo Gerenciador de Objetos - NuGO*. 61 f. Monografia (Bacharelado em Ciência da Computação) — Instituto de Biociências, Letras e Ciências Exatas, Universidade Estadual Paulista Júlio de Mesquita Filho, São José do Rio Preto, São Paulo, 2003.
- WIETRZYK, V. S.; ORGUN, M. A. Versant architecture: Supporting high-performance object databases. In: *IDEAS '98: Proceedings of the 1998 International Symposium on Database Engineering & Applications*. Washington, DC, USA: IEEE Computer Society, 1998. p. 141.