PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° 42/08

Extraindo Metadados para a Captura da Carga de Trabalho e Planos de Execução de SGBDs

José Maria Monteiro
Sérgio Lifschitz
Ângelo Brayner

Departamento de Informática

PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO
RUA MARQUÊS DE SÃO VICENTE, 225 - CEP 22453-900
RIO DE JANEIRO - BRASIL

Monografias em Ciência da Computação, No. 42/08 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Julho, 2008

Extraindo Metadados para a Captura da Carga de Trabalho e Planos de Execução de SGBDs

José Maria Monteiro, Sérgio Lifschitz, Ângelo Brayner[1]

[1] Mestrado em Informática Aplicada - Universidade de Fortaleza (UNIFOR)

monteiro@inf.puc-rio.br, sergio@inf.puc-rio.br, brayner@unifor.br

Abstract. The performance of database servers is a key factor for the success of mission-critical applications. In order to ensure acceptable performances, we need to continuously monitor the database server's infrastructure. Whenever an unexpected event that reduces the system performance is detected, the DBMS must react immediately, solving the problems efficiently. The main source of information used for database performance monitoring is the DBMS metadata. However, checking these metadata information is highly dependent of the DBMS version. In this work we present a study of database metadata of some popular and widely used DBMSs: PostgreSQL 8, Oracle 10g and SQL Server 2005. We give a series of programming scripts that may be used to fetch the database workload, together with the corresponding I/O Cost and Execution Plan. Moreover, we also capture statistical information used in the process of performance analysis and solutions.

Keywords: Database Metadada, Database Tuning, SQL Workload, DBMS Statistics

Resumo. O desempenho dos servidores de bancos de dados é fator chave para o sucesso das aplicações de missão-crítica. A fim de assegurar um desempenho sempre aceitável torna-se necessário monitorar continuamente os servidores de bancos de dados. Caso seja detectada a ocorrência de um evento que possa comprometer o desempenho do sistema, deve-se reagir de forma imediata, solucionando-se os problemas encontrados com eficiência. A principal fonte de informações utilizada no monitoramento de desempenho de bancos de dados são os metadados do próprio SGBD. Entretanto, a forma de se obter estes metadados depende fortemente de cada fabricante do SGBD. Neste trabalho realizamos um estudo dos metadados de alguns dos SGBDs mais populares: PostgreSQL 8, Oracle 10g e SQL Server 2005. Além disso, elaboramos e apresentamos uma série de scripts capazes de capturar a carga de trabalho submetida ao SGBD, seus respectivos custos e planos de execução, e as principais informações estatísticas utilizadas no processo de análise e resolução de problemas de desempenho.

Palavras-chave: Metadados de SGBDs, Ajustes de Desempenho de SGBDs, Carga de Trabalho, Estatísticas dos SGBDs

In charge for publications:
Rosane Teles Lins Castilho
Assessoria de Biblioteca, Documentação e Informação
PUC-Rio Departamento de Informática
Rua Marquês de São Vicente, 225 - Gávea
22453-900 Rio de Janeiro RJ Brasil
Tel. +55 21 3114-1516 Fax: +55 21 3114-1530
E-mail: bib-di@inf.puc-rio.br
Web site: http://bib-di.inf.puc-rio.br/techreports/

1 Introdução

O desempenho dos servidores de bancos de dados é fator chave para o sucesso das aplicações de missão-crítica. Para estas aplicações, um baixo desempenho significa perdas de receita e de oportunidades de negócio. A fim de assegurar um desempenho sempre aceitável torna-se necessário gerenciar continuamente a infra-estrutura dos servidores de bancos de dados. Este gerenciamento envolve realizar tarefas regulares de administração, responder aos eventos de sistema, além de ajustar os diversos parâmetros de desempenho. Desta forma, praticamente todos os sistemas que armazenam grandes volumes de dados são gerenciados por administradores (DBAs - *Database Administrators*), os quais devem ser especialistas nas tarefas de administração e sintonia fina (*tuning*) de bancos de dados. Isto é, o ajuste dos parâmetros do SGBD (Sistema de Gerenciamento de Banco de Dados) de acordo com as características da carga de trabalho das aplicações.

Os DBAs são responsáveis pela análise contínua do funcionamento do sistema de banco de dados, pela descoberta de problemas de desempenho e, principalmente, pela solução rápida e eficaz destes problemas. O DBA procura identificar as causas dos gargalos de desempenho e os problemas de contenção de recursos. Para isso, dentre as principais fontes de informação do DBA temos os metadados do próprio SGBD. Neste sentido, tarefas como:

- capturar os comandos SQL executados,
- os custos estimados pelo otimizador,
- o tempo de resposta (tempo decorrido entre enviar a consulta SQL e o retorno do resultado),
- o número estimado de tuplas retornadas,
- o número de varreduras *sequential scans*, e
- dados de séries temporais *time-series* de parâmetros de desempenho

tornam-se fundamentais. Também é importante armazenar estes dados coletados em um repositório, seja para diagnósticos rápidos, ou mesmo para análises futuras e planejamento.

Para avaliar o desempenho de um SGBD, tempo de resposta (*elapsed time*) é a métrica mais importante, uma vez que este é o componente do desempenho percebido pelo usuário final. Logo, identificar as instruções SQL que incorrem nos maiores tempos de resposta é uma tarefa de extrema relevância. Por exemplo, é importante identificar quais instruções SQL foram responsáveis por picos intermitentes de consumo de CPU, pela maior quantidade de leituras em disco, entre outros.

As ações do DBA, quando detectado um problema de desempenho, são baseadas na análise do tempo de respostas das instruções SQL executadas. Depois de um exame das possíveis causas do problema detectado, pode ser necessário intervir, alterando os parâmetros do SGBD até que o efeito desejado seja alcançado.

Entretanto, assim como a maioria dos sistemas computacionais, os sistemas de bancos de dados são extremamente complexos e há uma boa quantidade de variações nas implementações dos diferentes fornecedores de produtos e soluções. Como resultado, há variações significativas de desempenho em diferentes tarefas. Um SGBD pode ser mais eficiente para uma determinada tarefa do que outro [10].

A seqüência de medições, ou os instantâneos (*snapshots*), examinados em um intervalo de tempo não-aleatório, são chamados de série de tempo. As medições feitas mais freqüentemente fornecem uma precisão maior sobre o tempo em que os eventos ocorrem, isto é, a série de tempo com taxa de amostragem mais elevada terá a definição muito melhor. Porém, na realidade, cada medição pode acrescentar um custo extra (/it overhead) ao sistema sendo medido.

Associar medidas de desempenho a uma combinação de tarefas de administração e sintonia fina de bancos de dados requer cuidado. A maneira correta de tirar a média dos números é tomar o tempo total para conclusão da carga de trabalho no lugar da produtividade média para cada tipo de transação. Portanto, uma única tarefa não é suficiente para quantificar o desempenho do sistema. Este deve ser medido por um conjunto selecionado de tarefas e dados mais conhecido como *benchmark* [10].

A maneira de se obter estes metadados depende do fabricante do SGBD. Para o DBA buscar as últimas cláusulas SQL executadas no Oracle 10g ele realiza consultas completamente diferentes da forma como esta informação é obtida através dos metadados do SQL Server 2005. Além disso, seria interessante (para os DBAs) armazenar os metadados capturados (e utilizados para análise de desempenho) em um repositório próprio, seja para diagnósticos rápidos, ou para análises futuras e planejamento.

Este trabalho discute como extrair os metadados dos principais SGBDs comerciais: Postgres, Oracle 10g e SQL Server 2005. As informações capturadas podem ser utilizadas pelos DBAs para a analise da performance dos SGBDs e para descobrir problemas de desempenho. Além disso, os metadados obtidos ajudam a guiar o DBA na seleção das possíveis soluções para os problemas encontrados.

O restante deste trabalho está organizado da seguinte maneira: a Seção 2 discute o conceito e a utilização dos metadados, na Seção 3 mostramos como extrair os metadados do PostgreSQL versão 8, a Seção 4 ilustra como capturar os metadados do Oracle 10g, na Seção 5 discutimos a obtenção dos metadados no Microsoft SQL Server 2005 e a Seção 6 conclui este trabalho.

2 Metadados

Os metadados são frequentemente descritos como "dados sobre dados". Ou seja, são informações adicionais, além da informação espacial e tabular, necessárias para que os dados se tornem úteis. Em outras palavras, são um conjunto de características e estatísticas sobre os dados que não estão normalmente incluídas nos dados do problema propriamente dito.

Assim, os metadados são comumente utilizados para:

- Adicionar contexto e conhecimento sobre os dados acessados pelos usuários.
- Esconder (abstrair) a complexidade dos dados dos usuários que não necessitam conhecer os detalhes técnicos dos dados.
- Descobrir os tipos dos dados, os relacionamentos entre os dados, o momento em que um dado foi lido ou atualizado pela última vez.
- Possibilitar às aplicações realizarem uma validação de tipo, validação, formatação, etc.

- Possibilitar a análise do desempenho do SGBD.

Alguns exemplos de metadados que podem ser citados: esquemas de dados, tabelas, colunas, índices, últimas cláusulas SQL executadas, os planos de execução das consultas executadas, e muito mais.

3 Metadados no PostgreSQL

Nesta seção iremos apresentar o SGBD PostgreSQL, discutir suas principais características e descrever como seus metadados estão organizados. Além disso, iremos mostrar como capturar a carga de trabalho submetida ao PostgreSQL, os planos de execução das consultas que compõem a carga de trabalho capturada, bem como informações estatísticas relevantes para a análise de desempenho.

O PostgreSQL é um sistema de banco de dados objeto relacional (SGDBOR), derivado do pacote PostgreSQL, desenvolvido pelo Departamento de Ciência da Computação da Universidade da Califórnia, em Berkeley. O PostgreSQL foi pioneiro em vários conceitos que somente se tornaram disponíveis muito mais tarde em alguns sistemas de banco de dados comerciais [5]. O PostgreSQL é um SGBD de código fonte aberto, que suporta grande parte do padrão SQL-2003, além de oferecer muitas funcionalidades modernas, tais como visões materializadas e controle de concorrência multiversão.

3.1 O Catálogo do Sistema

O catálogo do sistema é o local onde o gerenciador de banco de dados armazena os metadados dos seus esquemas. Estes metadados formam um conjunto de informações sobre as tabelas, colunas e índices do banco de dados, além de informações sobre as tarefas e funções executadas pelo SGBD.

No PostgreSQL, o catálogo do sistema é formado por um conjunto de tabelas de sistema. A Tabela 1 descreve brevemente este conjunto [7]. Já na Figura 1 podemos ter uma idéia da estrutura do catálogo de sistema do PostgreSQL.

3.2 Visões do Sistema

Assim como temos as tabelas de sistema, temos também as visões (*views*) internas do sistema. A partir destas visões, o PostgreSQL fornece um acesso conveniente às tabelas do catálogo do sistema mais freqüentemente utilizadas [7]. Estas visões estão listadas e brevemente descritas na Tabela 2.

3.3 O Coletor de Estatísticas

O coletor de estatísticas do PostgreSQL é um subsistema de apoio a coleta e relatório de informações sobre as atividades do servidor. Atualmente, este coletor pode contar acessos a tabelas e índices em termos de blocos de disco e linhas individuais. Também apóia a determinação exata do comando sendo executado no momento pelos outros processos servidores [8].

Uma vez que a coleta de estatísticas adiciona alguma sobrecarga à execução da carga de trabalho, o sistema pode ser configurado para coletar informações ou não. Isto é controlado

Nome do Catálogo	Finalidade
pg_aggregate	funções de agregação
pg_am	métodos de acesso de índice
pg_amop	operadores de método de acesso
pg_amproc	procedimentos de suporte de método de acesso
pg_attrdef	valor padrão das colunas
pg_attribute	colunas das tabelas (“atributos”)
pg_cast	casts (conversões de tipos de dado)
pg_class	tabelas, índices, seqüências, visões
pg_constraint	restrições de verificação, restrições de unicidade, restrições de chave primária, restrições de chave estrangeira
pg_conversion	informações sobre conversão de codificação
pg_database	bancos de dados que fazem parte deste agrupamento de bancos de dados
pg_depend	dependências entre objetos do banco de dados
pg_description	descrições ou comentários sobre os objetos do banco de dados
pg_group	grupos de usuários do banco de dados
pg_index	informações adicionais sobre índices
pg_inherits	hierarquia de herança de tabela
pg_language	linguagens para escrever funções
pg_largeobject	objetos grandes
pg_listener	suporte a notificação assíncrona
pg_namespace	esquemas
pg_opclass	classes de operador de método de acesso de índice
pg_operator	operadores
pg_proc	funções e procedimentos
pg_rewrite	regras de reescrita dos comandos
pg_shadow	usuários do banco de dados
pg_statistic	estatísticas do planejador
pg_tablespace	espaços de tabelas do agrupamento de bancos de dados
pg_trigger	gatilhos
pg_type	tipos de dado

Tabela 1: Descrição Parcial do Catálogo de Sistema do PostgreSQL

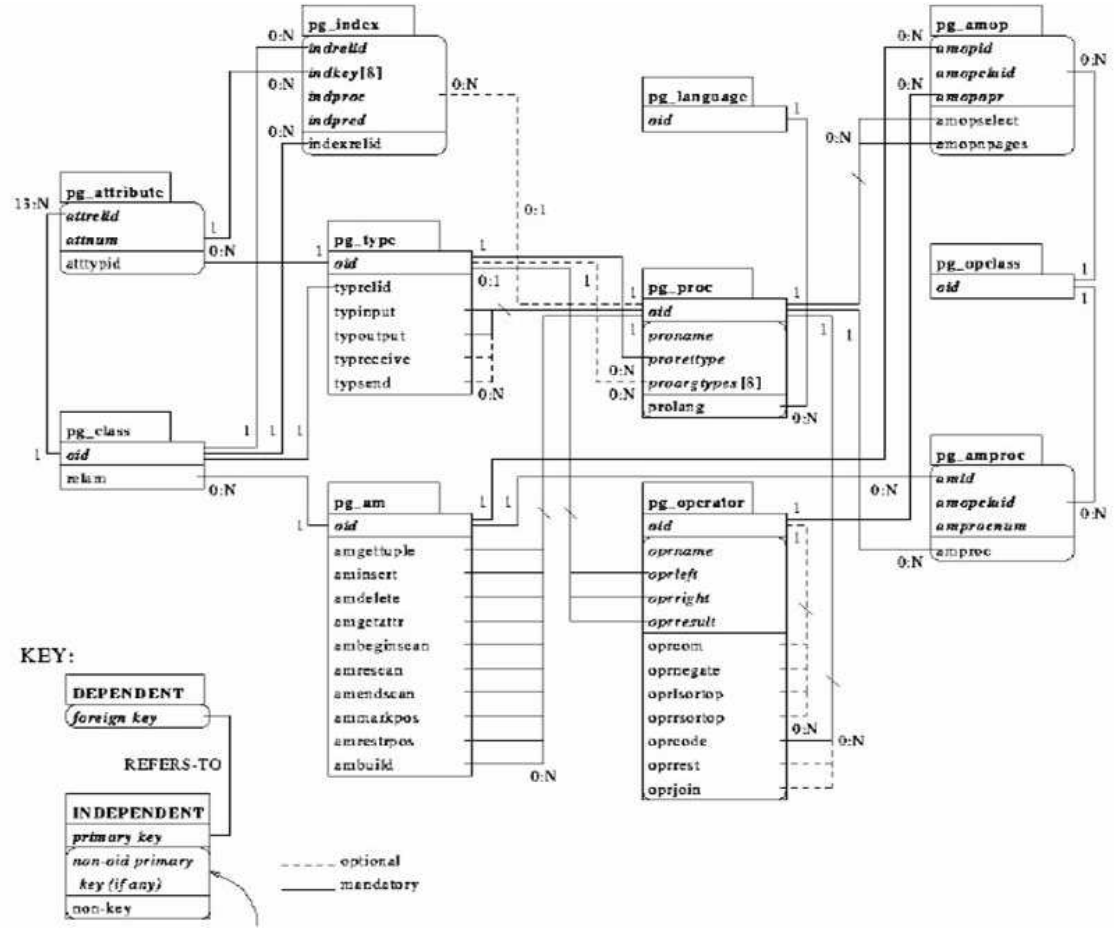

Figura 1: Estrutura do Catálogo de Sistema do PostgreSQL.

por parâmetros de configuração, normalmente definidos no arquivo postgresql.conf, que se encontra dentro do diretório *data* da instalação do PostgreSQL.

O parâmetro stats_start_collector deve ser definido como *true* para que o coletor de estatísticas seja ativado. Já os parâmetros stats_command_string, stats_block_level e stats_row_level controlam a quantidade de informação que é enviada para o coletor e, portanto, determinam quanta sobrecarga ocorre em tempo de execução. Determinam, respectivamente, se o processo servidor envia para o coletor a cadeia de caracteres do comando corrente, as estatísticas de acesso no nível de bloco de disco, e as estatísticas de acesso no nível de tupla.

A fim de ativar a coleta de estatísticas do PostgreSQL é necessário configurar o arquivo postgres.conf, na seção de RUNTIME STATISTICS, da seguinte maneira:

```
#---------------------------------------------------------------------------
# RUNTIME STATISTICS
#---------------------------------------------------------------------------

# - Statistics Monitoring -
```

Nome da Visão	Finalidade
pg_indexes	Índices
pg_locks	Bloqueios mantidos no momento
pg_rules	Regras
pg_settings	Configurações dos parâmetros
pg_stats	Estatísticas do otimizador
pg_tables	Tabelas
pg_user	Usuários do banco de dados
pg_views	Visões

Tabela 2: Visões de Sistema do PostgreSQL.

```
#log_parser_stats = on
#log_planner_stats = on
#log_executor_stats = on
log_statement_stats = on

# - Query/Index Statistics Collector -

stats_start_collector = on
stats_command_string = on
stats_block_level = on
stats_row_level = on
stats_reset_on_server_start = on
```

3.4 Visões de Estatísticas

O PostgreSQL disponibiliza diversas visões pré-definidas para mostrar os resultados das estatísticas coletadas pelo coletor de estatísticas. Como alternativa, podem ser construídas visões personalizadas utilizando as funções de estatísticas subjacentes [7, 4].

Ao se utilizar as estatísticas para monitorar a atividade corrente, é importante ter em mente que as informações não são atualizadas instantaneamente. Portanto, um comando (ou transação) ainda em progresso não afeta os totais exibidos. Também, o próprio coletor emite um novo relatório no máximo uma vez a cada pgstat_stat_interval milissegundos (500 por padrão). Assim, as informações mostradas são anteriores à atividade corrente.

Outro ponto importante é que, quando se solicita a um processo servidor para mostrar uma destas estatísticas, este busca primeiro os relatórios mais recentes emitidos pelo processo coletor e, depois, continua utilizando este instantâneo para todas as visões e funções de estatística até o término da transação corrente. Portanto, as estatísticas parecem não mudar enquanto se permanece na transação corrente. Isto é uma característica, e não um erro, pois permite realizar várias consultas às estatísticas e correlacionar os resultados. A Tabela 3 mostra as principais visões de estatísticas do PostgreSQL.

Nome da Visão	Descrição
pg_stat_activity	Uma linha por processo servidor, mostrando o ID do processo, o banco de dados, o usuário, o comando corrente e a hora em que o comando corrente começou a executar. As colunas que mostram os dados do comando corrente somente estão disponíveis quando o parâmetro stats_command_string está habilitado. Além disso, estas colunas mostram o valor nulo a menos que o usuário consultando a visão seja um super-usuário, ou o mesmo usuário dono do processo sendo mostrado (Deve ser observado que devido ao retardo do que é informado pelo coletor, o comando corrente somente será mostrado no caso dos comandos com longo tempo de execução).
pg_stat_database	Uma linha por banco de dados, mostrando o número de processos servidor ativos, total de transações efetivadas e total de transações canceladas neste banco de dados, total de blocos de disco lidos e total de acertos no *buffer* (ou seja, solicitações de leitura de bloco evitadas por encontrar o bloco no cache do *buffer*).
pg_stat_all_tables	Para cada tabela do banco de dados corrente, o número total de: varreduras seqüenciais e de índice; linhas retornadas por cada tipo de varredura; linhas inseridas, atualizadas e excluídas.
pg_stat_all_indexes	Para cada índice do banco de dados corrente, o total de varreduras de índice que utilizaram este índice, o número de linhas do índice lidas, e o número de linhas da tabela buscadas com sucesso (Pode ser menor quando existem entradas do índice apontando para linhas da tabela expiradas).
pg_statio_all_tables	Para cada tabela do banco de dados corrente, o número total de blocos de disco da tabela lidos, o número de acertos no *buffer*, o número de blocos de disco lidos e acertos no buffer para todos os índices da tabela, o número de blocos de disco lidos e acertos no *buffer* para a tabela auxiliar TOAST da tabela (se houver), e o número de blocos de disco lidos e acertos no buffer para o índice da tabela TOAST.
pg_statio_all_indexes	Para cada índice do banco de dados corrente, o número de blocos de disco lidos e de acertos no *buffer* para o índice.

Tabela 3: Principais Visões de Estatísticas do PostgreSQL.

3.5 Funções de Acesso às Estatísticas

O PostgreSQL fornece um conjunto de funções que oferecem acesso às informações estatísticas. Estas funções geram uma espécie de relatório, compilando informações relevantes. Algumas destas funções geram informações estatísticas (relatórios) por banco de dados, por tabelas, por índices e até por processos. As funções de acesso por banco de dados devem receber como argumento o OID do banco de dados para o qual se deseja obter as informações (relatório). Já as funções de acesso por tabela e por índice recebem o OID da tabela ou do índice, respectivamente.

Vale ressaltar que somente podem ser visualizadas por estas funções as tabelas e índices presentes no banco de dados corrente. As funções de acesso por processo servidor recebem o número de ID do processo servidor, que varia entre um (1) e o número de processos servidores ativos no momento [7, 4]. A Tabela 4 mostra as principais funções de acesso às estatísticas do PostgreSQL.

3.6 Extraindo a Carga de Trabalho Submetida ao PostgreSQL

Desejamos extrair do catálogo de sistemas (metadados) do PostgreSQL a carga de trabalho a este submetida. Assim, para cada instrução SQL executada, pretende-se obter: a própria cláusula SQL, seu plano de execução, o custo total, o custo de I/O e o número de execuções deste comando SQL (ou seja, o número de vezes que o comando SQL foi executado). A seguir descrevemos como obter essas informações a partir dos metadados do PostgreSQL.

A fim de capturar a cláusula SQL em execução, para cada processo servidor ativo utilizamos a consulta mostrada a seguir. Observe que esta consulta gera uma linha para cada processo servidor, exibindo o seu PID (identificação do processo) e o seu comando corrente.

```
SELECT pg_stat_get_backend_pid(s.backendid) AS procpid,
      pg_stat_get_backend_activity(s.backendid) AS current_query
FROM (SELECT pg_stat_get_backend_idset() AS backendid) AS s;
```

Para obter o plano de execução de uma determinada clásula SQL usamos o comando "EXPLAIN + instruções SQL capturada", da seguinte forma:

```
EXPLAIN + INSTRUÇÃO SQL EXECUTADA DURANTE A CONEXÃO DO SGBD
```

3.7 Extraindo Informações Estatísticas do PostgreSQL

Com a finalidade de fornecer um suporte mais adequado para a análise da carga de trabalho capturada é necessário recuperar algumas informações estatísticas, tais como: o número de blocos (páginas de dados) ocupados por uma determinada relação, os índices existentes sobre uma determinada tabela, o método de acesso de um determinado índice, a altura de um determinado índice (árvore B^+), dentre outros.

Para obter essas informações foi necessário o estudo e utilização do catalogo de sistema e das visões de sistema, anteriormente descritas. Dentre as tabelas e visões existentes no catálogo dos sistemas, uma tabela e uma visão são suficientes para retornar os dados desejados. São eles: a tabela de sistema PG_CLASS e a visão de sistema PG_INDEXES. A

Nome da Função	Descrição
pg_stat_get_db_numbackends(oid)	Número de processos servidor ativos conectados ao banco de dados
pg_stat_get_numscans(oid)	Número de varreduras seqüenciais realizadas quando o argumento é uma tabela, ou o número de varreduras de índice quando o argumento é um índice
pg_stat_get_tuples_returned(oid)	Número de linhas lidas por varreduras seqüenciais quando o argumento é uma tabela, ou o número de linhas do índice lidas quando o argumento é um índice
pg_stat_get_tuples_inserted(oid)	Número de linhas inseridas na tabela
pg_stat_get_tuples_updated(oid)	Número de linhas atualizadas na tabela
pg_stat_get_tuples_deleted(oid)	Número de linhas excluídas da tabela
pg_stat_get_backend_idset()	Conjunto de IDs de processos servidor ativos no momento (de 1 ao número de processos servidores ativos)
pg_backend_pid()	ID de processo do processo servidor conectado à sessão corrente
pg_stat_get_backend_pid(integer)	ID de processo do processo servidor especificado
pg_stat_get_backend_activity(integer)	Comando ativo do processo servidor especificado (nulo se o usuário corrente não for um superusuário nem o mesmo usuário da sessão sendo consultada, ou se stats_command_string não estiver habilitado)
pg_stat_get_backend_activity_start(integer)	A hora em que o comando executando no momento, no processo servidor especificado, começou (nulo se o usuário corrente não for um superusuário nem o mesmo usuário da sessão sendo consultada, ou se stats_command_string não estiver habilitado)

Tabela 4: Principais Funções de Acesso às Estatísticas do PostgreSQL.

Coluna	Tipo de dado	Descrição
relname	name	Nome da tabela, índice, visão, etc.
relpages	int4	Tamanho da representação em disco desta tabela em páginas (com tamanho BLCKSZ). Esta é apenas uma estimativa utilizada pelo otimizador de planos. Atualizado pelos comandos VACUUM, ANALYZE e uns poucos comandos DLL como o CREATE INDEX.
reltuples	float4	Número de linhas na tabela. Esta é apenas uma estimativa utilizada pelo otimizador. Atualizado pelos comandos VACUUM, ANALYZE e uns poucos comandos de DLL como o CREATE INDEX.
relhasindex	bool	Verdade se for uma tabela e possui (ou possuía recentemente) algum índice. É definido por CREATE INDEX, mas não é limpo imediatamente por DROP INDEX. O comando VACUUM limpa esta coluna quando descobre que a tabela não possui índices.
relhaspkey	bool	Verdade se a tabela possui uma chave primária.

Tabela 5: Principais Campos da Tabela de Sistema PG_CLASS.

Coluna	Tipo de dado	Descrição
schemaname	name	Nome do esquema que contém a tabela e o índice
tablename	name	Nome da tabela para a qual o índice se destina
indexname	name	Nome do índice
tablespace	name	Nome do espaço de tabelas contendo o índice (NULL se for o padrão para o banco de dados)
indexdef	text	Definição do índice (o comando de criação reconstruído)

Tabela 6: Principais atributos da visão PG_INDEXES.

descrição desses objetos, seus atributos mais importantes, além do relacionamento entre eles, são mostrados a seguir:

A tabela pg_class cataloga as tabelas e qualquer outro objeto do SGBD que possui colunas, ou que de alguma forma seja semelhante a uma tabela. Isto inclui os índices, seqüências, visões, tipos compostos, dentre outros. A descrição de seus principais campos, pode ser vista na Tabela 5.

A visão PG_INDEXES fornece acesso a informações úteis sobre cada índice do banco de dados. Ela possui informações como nome do esquema (*schema*), nome da tabela, nome do índice, e a definição do índice. A Tabela 6 descreve os principais campos da visão PG_INDEXES.

Para se obter informações sobre uma determinada tabela ou visão utilizamos a tabela de sistema PG_CLASS e a visão PG_INDEXES. Apresentamos agora algumas cláusulas SQL que podem ser utilizadas para buscar as informações estatísticas do PostgreSQL. A consulta a seguir obtém a quantidade de tuplas de uma determinada tabela.

```
SELECT RELTUPLES AS NUM_LINHAS
FROM PG_CLASS
WHERE RELNAME IN (
  SELECT TABLENAME
FROM PG_TABLES
WHERE SCHEMANAME LIKE '"+SCHEMA+"'
  AND TABLENAME LIKE '" + NOMETABELA + "'
);
```

Já a consulta a seguir obtém a quantidade de blocos ("paginas") de uma determinada tabela.

```
SELECT RELPAGES AS NUM_PAGINAS
FROM PG_CLASS
WHERE RELNAME IN (
  SELECT TABLENAME
FROM PG_TABLES
WHERE SCHEMANAME LIKE '"+SCHEMA+"'
  AND TABLENAME LIKE '" + NOMETABELA + "'
);
```

Para se retornar os indices definidos sobre uma determinada tabela podemos utilizar a seguinte expressão SQL:

```
SELECT INDEXNAME
FROM PG_INDEXES"
WHERE SCHEMANAME LIKE '"+SCHEMA+"'
    AND TABLENAME LIKE '" + NOMETABELA + "';
```

Para se obter o método de acesso de um determinado índice podemos utilizar a expressão SQL apresentada a seguir:

```
SELECT  (SELECT AMNAME
         FROM PG_AM
         WHERE PG_AM.OID= PG_CLASS.RELAM
         ) AS NOME
FROM PG_CLASS
WHERE RELNAME LIKE (SELECT INDEXNAME
                    FROM PG_INDEXES
                    WHERE SCHEMANAME LIKE '"+SCHEMA+"'
```

```
 AND INDEXNAME LIKE '" + NOMEINDICE + "'
);
```

4 Oracle 10g

Nesta seção iremos apresentar o SGBD Oracle 10g, discutir algumas de suas principais características e descrever como seus metadados estão organizados. Além disso, iremos mostrar como capturar a carga de trabalho submetida ao Oracle 10g, os planos de execução das consultas que compõem a carga de trabalho capturada, bem como informações estatísticas relevantes para a análise de desempenho.

O Oracle 10g oferece suporte para diferentes plataformas, fornece aos usuários a possibilidade de começar com uma solução básica e migrar para outras versões mais complexas, quando necessário. Proporciona ainda facilidades para o desenvolvimento de aplicações web através do Oracle Application Express [2, 1].

A versão 10g está voltada para o conceito de computação em grade *grid*. Neste sentido, busca facilitar a implementação de soluções de alta escalabilidade, fornecendo suporte para *clusters* de dados e bancos de dados distribuídos.

4.1 Dicionário de Dados

O Oracle 10g possui um dicionário de dados o qual consiste em um conjunto de tabelas especiais, de propriedade do usuário *SYS*, que têm como função registrar todas as informações de todos os objetos criados no banco de dados. Esses objetos podem ser tabelas, usuários, objetos, *sequences*, *tablespaces*, *views*, *constraints*, entre outros. Para termos acesso ao dicionário de dados do Oracle 10g podemos acessar as visões internas do Oracle, que estão separadas em três grandes grupos: *USER_*, *ALL_* e *DBA_*. Cada grupo possui seu escopo próprio que determinará o nível de acesso que cada um abrange [6]. A seguir uma breve descrição de cada grupo:

- *USER_*: exibe todos os objetos criados no próprio esquema (*schema*) do usuário.
- *ALL_*: lista todos os objetos, acessíveis ao usuário, associados a um ou vários esquemas.
- *DBA_*: mostra todos os objetos do banco de dados.

Na Figura 2 podemos visualizar, de forma simplificada, a estrutura do dicionário de dados do Oracle 10g.

Na tabela 7 descrevemos as principais visões pertencentes ao grupo *DBA_*. Através destas visões podemos obter os metadados dos objetos armazenados no banco de dados.

Outro grupo de visões de extrema importância que compõe o dicionário de dados do Oracle 10g é o grupo das visões dinâmicas V$. Estas visões exercem um papel importantíssimo no monitoramento das atividades do SGBD. Elas constituem as fontes para se obter a estimativa real da carga de trabalho submetida ao banco de dados. Como exemplo, no Oracle 10g temos a visão V$ACTIVE_SESSION_HISTORY, que contém o histórico das seções ativas no banco de dados. As visões V$ estão divididas em grupos e subgrupos, e são organizadas pelas funcionalidade de cada grupo [6].

Visão	Descrição
Objects	mostra todos os objetos do banco de dados, como tabelas, índices, visões, partições, etc.
Tables	contém a descrição de todas as tabelas do banco de dados.
Tab_Columns	mostra a descrição de todas as colunas de todas as tabelas, visões e *clusters* do banco de dados.
Tab_Partitions	contém informações sobre as partições de tabelas.
Indexes	mostra todos os índices do banco de dados.
Ind_Columns	fornece as informações exatas sobre as colunas que são chaves dos índices, como por exemplo: nome, tabela, localização, tamanho, dentre outras.
Ind_Partitions	contém informações sobre as partições de índices.
Constraints	possui todas as regras de integridade ou validação dos dados de entrada que estão nas tabelas.
Cons_Columns	fornece as informações sobre as colunas e definição de regras de integridade, as *constraints*.
Users	mostra todos os usuários do banco de dados e passa informações precisas como *USERNAME*, *USER_ID*, *ACCOUNT_STATUS*, *SESSION*, *PROCESS*.
Views	lista todas as visões do banco de dados.
Updatable_Columns	permite saber se as colunas de uma tabela podem ser alteradas, inseridas ou excluídas.

Tabela 7: Principais Visões do Grupo *DBA_*.

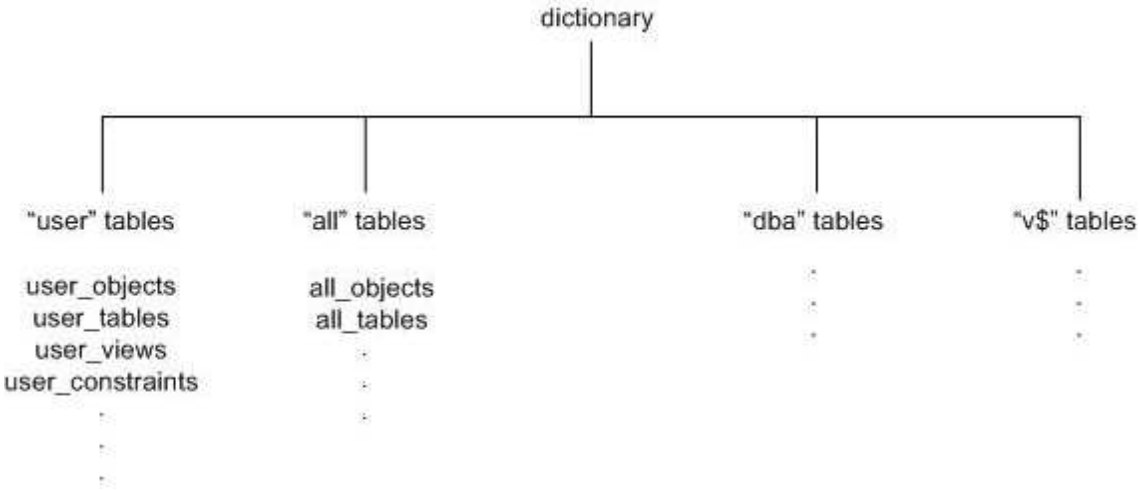

Figura 2: Estrutura do Dicionário de Dados do Oracle.

Uma característica importante das visões dinâmicas do Oracle 10g é que suas informações são armazenadas temporariamente na SGA e são eliminadas periodicamente. Além disso, a cada reinício do banco essas informações também são perdidas.

4.2 Extraindo a Carga de Trabalho Submetida ao Oracle 10g

Desejamos extrair do dicionário de dados (metadados) do Oracle 10g a carga de trabalho a este submetida. Assim, para cada instrução SQL executada, pretende-se obter: a própria cláusula SQL, seu plano de execução, o custo total, o custo de I/O e o número de execuções deste comando SQL (ou seja, o número de vezes que o comando SQL foi executado).

Para isso, o estudo do dicionário de dados e, principalmente, das visões dinâmicas V$, foi indispensável. Como o dicionário de dados do Oracle é composto por diversos grupos, os quais podem conter vários sub-grupos (os quais abrangem um determinado número de visões), esta pesquisa selecionou algumas visões, de grupos distintos, as quais, em conjunto, fornecem os dados desejados. As visões selecionadas foram: V$ACTIVE_SESSION_HISTORY, V$SQL e V$SQL_PLAN. A seguir, descreveremos essas visões e seus atributos mais importantes.

A visão V$ACTIVE_SESSION_HISTORY contém o histórico de cada sessão ativa do banco. Uma sessão da base de dados é considerada ativa se estiver no processador central ou estiver esperando um evento que não pertença à classe inativa de espera. Esta visão guarda os registros de todas as instruções SQL executadas pelas sessões ativas. Os principais campos dessa visão estão descritos na Tabela 8.

Já a visão V$SQL lista as informações estatísticas e o texto completo de cada cláusula SQL executada. A Tabela 9 mostra, em detalhes, os principais campos desta visão.

A Visão V$SQL_PLAN contém a informação do plano de execução de cada instrução SQL submetida ao SGBD. Esta é considerada a principal visão, pois é nela que se encontra o plano de execução, custo de E/S, número de linhas e outros detalhes das instruções SQL.
A Figura 3 mostra o relacionamento entre as três visões utilizadas (V$ACTIVE_SESSION_HISTORY, V$SQL e V$SQL_PLAN).

A Tabela 10 mostra, em detalhes, os campos desta visão que foram utilizados.

Coluna	Tipo de dado	Descrição
SQL_ID	VARCHAR2(13)	Identificador da instrução SQL carregada na seção durante sua execução.
SQL_PLAN_HASH_VALUE	NUMBER	Representação numérica do plano da instrução SQL.

Tabela 8: Principais Campos da Visão V$ACTIVE_SESSION_HISTORY.

Coluna	Tipo de dado	Descrição
ADDRESS	RAW(4—8)	Endereço para manipular o cursor.
HASH_VALUE	NUMBER	Valor hash para a instrução no *cache* de biblioteca.
SQL_ID	VARCHAR2(13)	Identificador da instrução SQL.
PLAN_HASH_VALUE	NUMBER	Representação numérica do plano de execução para o cursor. A comparação de um PLAN_HASH_VALUE a outro facilmente identifica se os dois planos são os mesmos ou não, ao invés de comparar os dois planos linha a linha.
SQL_FULLTEXT	CLOB	O texto completo de uma instrução SQL pode ser recuperado usando apenas esta coluna como alternativa à concatenação das partes da instrução, presentes na coluna sql_text da visão v$sql_text.
EXECUTIONS	NUMBER	Número de vezes que uma instrução SQL foi executada durante a conexão com o banco de dados.

Tabela 9: Principais Campos da Visão V$SQL.

Coluna	Tipo de dado	Descrição
ADDRESS	RAW(4—8)	Endereço para manipular o cursor.
HASH_VALUE	NUMBER	Valor *hash* da instrução no *cache* de biblioteca. As colunas ADDRESS e HASH_VALUE podem ser usadas para adicionar informação específica através da visão V$SQLAREA.
SQL_ID	VARCHAR2(13)	Identificador SQL do cursor no *cache* de biblioteca.
PLAN_HASH_VALUE	NUMBER	Representação numérica do plano de execução para o cursor. A comparação de um PLAN_HASH_VALUE a outro facilmente identifica se os dois planos são o mesmo ou não (ao invés de comprar os dois planos linha a linha).
OPERATION	VARCHAR2(30)	Nome da operação interna executada neste passo, por exemplo, TABLE ACCESS.
OPTIONS	VARCHAR2(30)	Uma variação da operação descrita na coluna OPERATION, por exemplo, FULL.
OBJECT_NAME	VARCHAR2(31)	Nome da tabela ou índice.
ID	NUMBER	Número atribuído a cada passo no plano de execução.
COST	NUMBER	Custo da operação como estimado pela abordagem baseada no custo realizada pelo otimizador.
CARDINALITY	NUMBER	Estimativa, baseada no custo, do número de linhas produzidas pela operação.
IO_COST	NUMBER	Custo de I/O da operação, como estimado pela abordagem baseada em custos. Para instruções que usam a abordagem baseada em regras, a coluna terá valor NULL.

Tabela 10: Principais Campos da Visão V$SQL_PLAN.

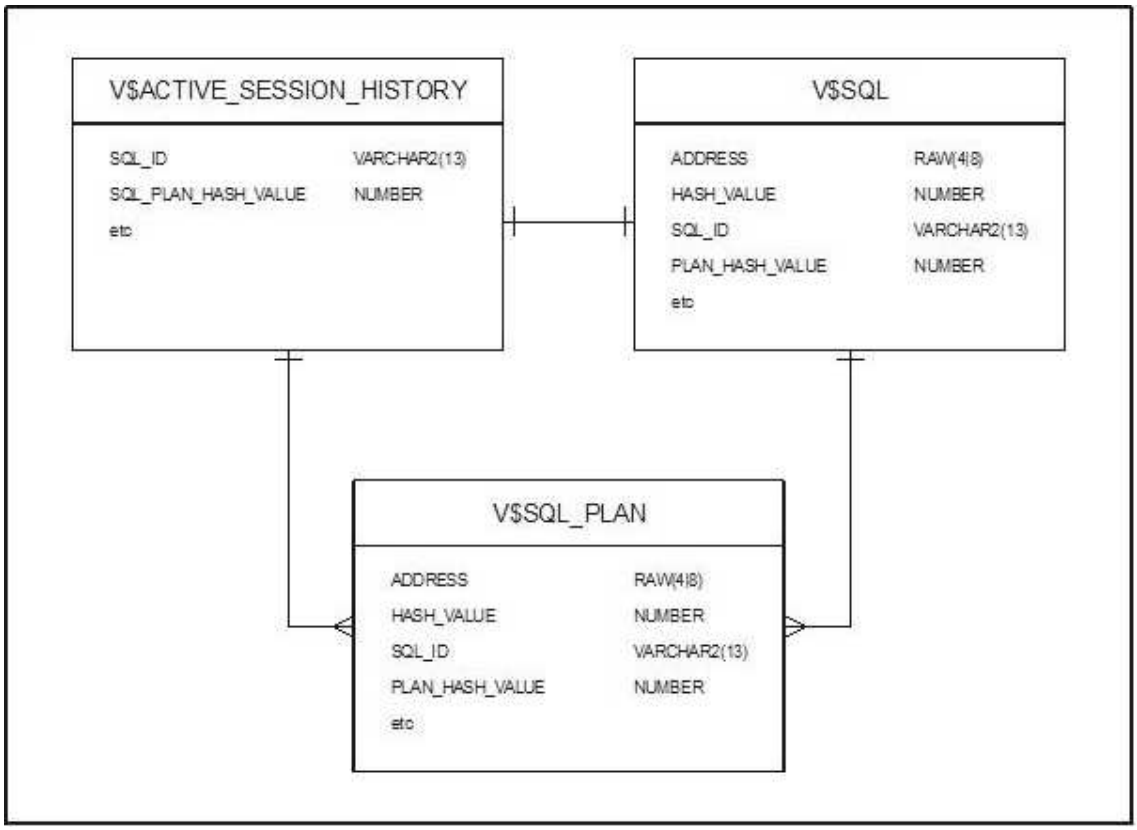

Figura 3: Relacionamento Entre as Visões V$ Utilizadas.

Cláusulas SQL Utilizadas para Extrair a Carga de Trabalho

A seguir apresentamos uma cláusula SQL que permite obter as informações sobre as últimas consultas executadas no Oracle 10g. Vale ressaltar que somente as consultas presentes na visão V$ACTIVE_SESSION_HISTORY são capturadas.

```
SELECT SQL.ADDRESS, SQL.HASH_VALUE, SQL.SQL_FULLTEXT, SQL.EXECUTIONS
FROM V$SQL SQL
WHERE EXISTS (SELECT ASH.SQL_ID, ASH.SQL_PLAN_HASH_VALUE
               FROM V$ACTIVE_SESSION_HISTORY ASH
               WHERE ASH.SQL_PLAN_HASH_VALUE = SQL.PLAN_HASH_VALUE
                     AND ASH.SQL_ID = SQL.SQL_ID
               )
      AND SQL.SQL_FULLTEXT NOT LIKE '%FROM V$SQL SQL%'
```

A seguir apresentamos uma cláusula que busca no dicionário de dados os planos de execução das últimas consultas executadas no Oracle 10g (e anteriormente capturadas).

```
SELECT SQL.ADDRESS, SQL.HASH_VALUE, SP.OPERATION, SP.OPTIONS,
       SP.IO_COST, SP.CARDINALITY, SQL.EXECUTIONS, SP.ID
```

Coluna	Tipo de dado	Descrição
OWNER	VARCHAR2(30)	Proprietário da tabela
TABLE_NAME	VARCHAR2(30)	Nome da tabela
NUM_ROWS*	NUMBER	Número de linhas da tabela
BLOCKS*	NUMBER	Número de blocos de dados usados

Tabela 11: Principais Campos da Visão DBA_TABLES.

```
FROM V$SQL SQL, V$SQL_PLAN SP
WHERE SP.PLAN_HASH_VALUE = SQL.PLAN_HASH_VALUE
      AND SP.SQL_ID = SQL.SQL_ID
      AND EXISTS (SELECT ASH.SQL_ID, ASH.SQL_PLAN_HASH_VALUE
                  FROM V$ACTIVE_SESSION_HISTORY ASH
                  WHERE ASH.SQL_PLAN_HASH_VALUE = SQL.PLAN_HASH_VALUE
                  AND ASH.SQL_ID = SQL.SQL_ID
                 )
      AND SQL.SQL_FULLTEXT NOT LIKE '%FROM V$SQL SQL%'
```

4.3 Extraindo Informações Estatísticas do Oracle 10g

Com a finalidade de fornecer um suporte mais adequado para a análise da carga de trabalho capturada é necessário obter algumas informações estatísticas, tais como: o número de blocos (páginas de dados) ocupados por uma determinada relação, os índices existentes sobre uma determinada tabela, o método de acesso de um determinado índice, a altura de um determinado índice (árvore B^+), dentre outros.

Para obter essas informações, foi necessário o estudo e utilização do dicionário de dados, anteriormente descrito. Neste sentido, exploramos as visões do grupo DBA_. Dentre as visões existentes no grupo DBA_, duas delas podem ser utilizadas para retornar os dados desejados: DBA_TABLES e DBA_INDEXES. A descrição dessas visões, seus atributos mais importantes e o relacionamento entre elas, são mostrados a seguir.

A visão DBA_TABLES contém a descrição de todas as tabelas do banco de dados. Ela informa em qual *tablespace* a tabela está localizada, o número de linhas, blocos de dados alocados, entre outras informações. A descrição de seus principais campos, pode ser vista na Tabela 11.

A visão DBA_INDEXES mostra todos os índices do banco de dados. Ela possui informações como nome do índice, ordem da árvore B^+, tabela, tamanho, tipo de tabela, localização, *tablespace*, entre outras. A Tabela 12 mostra, em detalhes, os campos que foram utilizados.

Cabe observar que no caso do BLEVEL*, uma profundidade 0 indica que a raiz e a folha são o mesmo nó. Já a Figura 4 mostra o relacionamento entre as visões DBA_TABLES e DBA_INDEXES.

Cláusulas SQL Utilizadas para Obter os Dados Estatísticos de uma Tabela

Coluna	Tipo de dado	Descrição
OWNER	VARCHAR2(30)	Proprietário do índice
INDEX_NAME	VARCHAR2(30)	Nome do índice
TABLE_OWNER	VARCHAR2(30)	Proprietário da tabela indexada
TABLE_NAME	VARCHAR2(30)	Nome da tabela indexada
BLEVEL*	NUMBER	Nível da árvore B^+ : profundidade da raiz aos nós folha.

Tabela 12: Principais Campos da Visão DBA_INDEXES.

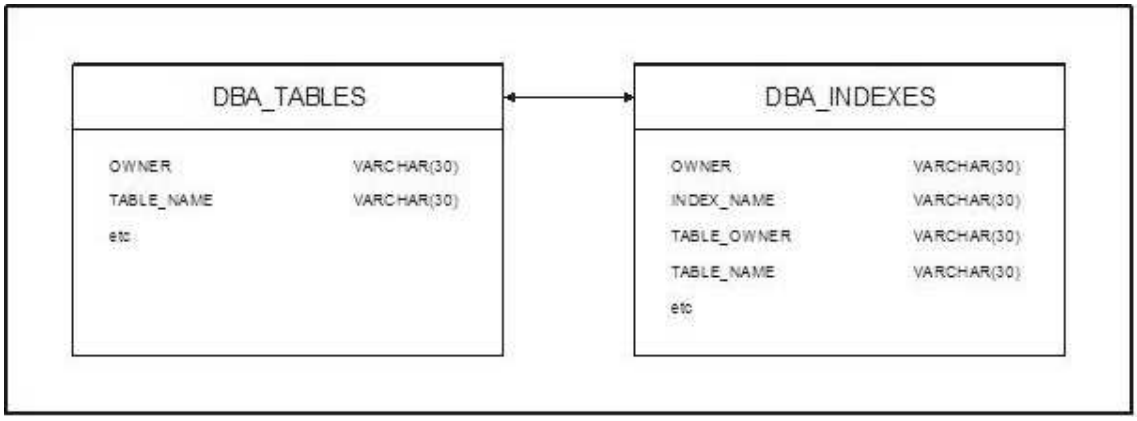

Figura 4: Relacionamento Entre as Visões V$ Utilizadas.

Para se obter as informações estatísticas sobre uma determina tabela, foi criada uma cláusula SQL que utiliza as visões DBA_TABLES e DBA_INDEXES (as quais se relacionam através dos campos TABLE_OWNER e TABLE_NAME), para retornar os dados desejados: nome da tabela, número de linhas, número de blocos, nome do índice e altura da árvore. A consulta faz uma busca nestas duas visões, através do nome da tabela. Esta cláusula está descrita a seguir.

```
SELECT T.TABLE_NAME nomeTabela, T.NUM_ROWS as numeroLinhas,
       T.BLOCKS as numeroBlocos, I.INDEX_NAME nomeIndice,
       I.BLEVEL as alturaArvore
FROM DBA_TABLES T, DBA_INDEXES I
WHERE T.TABLE_NAME = 'nomeTabela'
      AND T.OWNER = I.TABLE_OWNER(+)
      AND T.TABLE_NAME = I.TABLE_NAME(+)
```

5 SQL Server 2005

Nesta seção iremos apresentar o SGBD *SQL Server* 2005, discutir suas principais características e descrever como seus metadados estão organizados. Além disso, iremos mostrar como capturar a carga de trabalho submetida ao *SQL Server* 2005, os planos de execução das consultas que compõem a carga de trabalho capturada, bem como informações estatísticas relevantes para a análise de desempenho.

A *Microsoft* oferece a família de produtos *SQL Server* 2005 em quatro edições: *Express*, *Workgroup*, *Standard* e *Enterprise* [3, 9].

5.1 O Catálogo do Sistema

Quando criamos uma tabela no *SQL Server*, com suas colunas, índices e tipos de dados, estas informações (metadados) são armazenadas no *Database Catalog*, um conjunto de tabelas e visões que armazena informações sobre os objetos que o usuário criou no banco de dados, como por exemplo, tabelas, procedimentos armazenados ou visões [11].

Existem basicamente cinco maneiras de retornar metadados no *SQL Server*:

1. Através de tabelas de sistema (*System Tables*)
2. Através das visões do sistema (*System Views*)
3. Através de funções e procedimentos armazenados do *SQL Server*
4. *OLE DB schema rowsets*
5. *ODBC catalog functions*

A seguir vamos descrever cada um destes casos.

Tabela de Sistema	Descrição
sysobjects	Armazena informações sobre todos os objetos do B.D.
sysindexes	Armazena informações específicas sobre índices do BD.
syscolumns	Armazena todas as informações sobre todas as tabelas do BD.
syscomments	Armazena o código fonte de procedimentos armazenados (stored procedures) e funções do BD.
syslocks*	Armazena informações sobre os locks (bloqueios) dos objetos do BD.
sysdatabases*	Armazena informações sobre os Bancos de dados do Servidor SQL Server.

Tabela 13: Principais Tabelas de Sistema do SQL Server

5.1.1 Tabelas de Sistema

Os metadados no *SQL Server* são armazenados em tabelas de sistema. Todas as tabelas de sistema começam pelo prefixo sys e em hipótese alguma devem ser modificadas, pois caso alguma coisa errada ocorra com elas, o banco de dados inteiro pode deixar de funcionar. Não se recomenda o acesso direto (isto é , dar um SELECT nestas tabelas) a estas tabelas, pois elas podem mudar de nome nas próximas versões do produto, tornando seu código inválido e sem nenhuma portabilidade [11].

Na Tabela 13 temos algumas das principais tabelas de sistema e um breve comentário sobre cada uma delas:

Vale ressaltar que as tabelas de sistema marcadas com * somente existem no B.D. (banco de dados) *MASTER*.

5.1.2 Visões de Sistema

O método de acesso aos metadados mais recomendado pela *Microsoft* consiste na utilização das Visões de Sistema. Estas encapsulam o uso das tabelas de sistema. Estas visões só foram implementadas a partir do *SQL Server 7.0* e para os usuários *Oracle*, as Visões de Sistema são muito parecidas com as visões que começam com V$ (*Dynamic Performance Views* do *Oracle*) [11].

No *SQL Server* 2005 estas visões estão agrupadas em coleções (ou esquemas):

- Information Schema Views
- Compatibility Views
- Catalog Views
- Replication Views
- Dynamic Management Views and Functions

Desta forma, no *SQL Server* 2005 as tabelas de sistema estão escondidas e o acesso à elas é restrito. Assim, recomenda-se que o acesso aos metadados seja realizado através

das visões de sistema. Algumas destas visões foram incorporadas na versão 2005, outras existem desde as versões anteriores. A vantagem das visões é que estas possuem uma leitura mais fácil e são auto-descritivas. Para facilitar a conversão de *scripts* legados baseados nas tabelas de sistema o *SQL Server 2005* fornece um conjunto de visões que substituem diretamente as tabelas de sistema. Estas visões são denominadas de *compatibility view*. No *SQL Server 2005* existem mais de 230 visões de sistema, lembrando que no *SQL Server* 2000 haviam apenas pouco mais de 50 tabelas de sistema [11].

As visões de sistema são automaticamente inseridas em qualquer banco de dados criado pelos usuários. Estas visões são agrupadas em diferentes esquemas (coleções). Logo, um mesmo banco de dados vai conter diferentes esquemas (contendo as visões do sistema). A seguir descrevemos alguns dos principais grupos (ou esquema) de visões de sistema.

Information Schema Views

As visões do grupo *Information schema views* baseiam-se nas especificações de catálogo do padrão SQL-92. Elas apresentam as informações do catálogo em um formato independente das tabelas de sistema que armazenam os metadados. Os usuários e/ou aplicações podem utilizar estas visões e obter portabilidade entre diferentes SGBDs que sigam o padrão SQL-92 [11].

Existem vinte diferentes visões neste esquema (grupo). Estas visões são utilizadas para se recuperar os aspectos físicos de um banco de dados, tais como as tabelas existentes, as colunas de uma determinada tabela, etc.

A seguir mostramos alguns exemplos da utilização destas visões:

```
SELECT *
FROM INFORMATION_SCHEMA.TABLES
-- Retorna as informações armazenadas no catálogo de todas as tabelas existentes

SELECT *
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'MyTable'
-- Retorna as informações sobre as colunas da tabela "MyTable"

SELECT COLUMN_NAME ,IS_NULLABLE , DATA_TYPE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE TABLE_NAME = 'TABELA1'
-- Retorna o nome, se a coluna permite null e o tipo de dados de todas as colunas
-- da tabela chamada 'TABELA1'

WITH ENCRIPTION
SELECT SPECIFIC_NAME , ROUTINE_TYPE , ROUTINE_DEFINITION
FROM INFORMATION_SCHEMA.ROUTINES
-- Retorna o nome , o tipo de rotina ( Stored Procedure ou função ) e o código fonte da
-- rotina , desde que a mesma não tenha sido criada com a opção

SELECT TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
```

```
WHERE TABLE_TYPE = 'VIEW'
-- Retorna todas as tabelas do banco de dados atual, que na verdade são views
```

Como discutimos anteriormente, muitas das tabelas de sistema existentes nas versões anteriores do *SQL Server* são agora implementadas (no *SQL server 2005*) como um conjunto de visões. Estas visões são denominadas visões de compatibilidade (*compatibility views*), e existem somente por uma questão de compatibilidade. Elas mostram os mesmos metadados que eram disponibilizados no *SQL Server 2000*. Entretanto, elas não mostram os metados relacionados com as novas características adicionadas pelo *SQL Server 2005* [11].

O *SQL Server* 2005 introduziu as visões de catálogo (*catalog views*) como uma interface completa para se acessar os metadados do sistema. Estas visões proporcionam o acesso aos metadados armazenados em todos os bancos de dados (*databases*) do servidor [11].

5.1.3 Através de Funções do SQL Server

Uma outra maneira mais segura de se obter metadados é utilizando algumas funções já prontas do *SQL Server* para acessar essas informações. No exemplo a seguir, utilizamos duas funções: primeiro a função OBJECT_ID() que retorna um identificador interno do *SQL Server* para um objeto e depois a função OBJECTPROPERTY() para retornar se o objeto é uma tabela ou não:

```
SELECT OBJECTPROPERTY(OBJECT_ID('TABELA1'),'isTable')
```

O retorno da função depende de qual propriedade do objeto se está consultando. Neste caso, a propriedade chama-se 'isTable' e a função OBJECTPROPERTY() retorna o valor 1 se o objeto chamado TABELA1 for uma tabela, 0 se não for uma tabela e NULL se o objeto não existir no banco de dados atual.

5.1.4 OLE DB Schema Rowsets

A especificação OLE DB define uma interface denominada IDBSchemaRowset que mostra o conjunto dos esquemas que contêm as informações do catálogo. O OLE DB *schema rowsets* constitui um padrão para apresentar as informações de catálogo suportadas por diferentes provedores OLE DB. Estes *rowsets* são independentes da estrutura das tabelas de sistema [11].

5.1.5 ODBC Catalog Functions

A especificação ODBC define um conjunto de funções de catálogo que retornam cursores (*result sets*) contendo as informações do catálogo. Estas funções constituem um método padrão para representar as informações do catálogo que são suportadas por diferentes *drivers* ODBC. Os cursores retornados por estas funções são independentes da estrutura das tabelas de sistema [11].

5.2 Extraindo a Carga de Trabalho Submetida ao SQL Server 2005

Desejamos extrair do catálogo de sistemas (metadados) do *SQL Server* 2005 a carga de trabalho a este submetida. Assim, para cada instrução SQL executada, pretende-se obter: a própria cláusula SQL, seu plano de execução, o custo total, o custo de I/O e o número de execuções deste comando SQL (ou seja, o número de vezes que o comando SQL foi executado). Para isso, o estudo do catálogo do sistema, das visões de sistema, além das visões e funções de acesso às estatísticas foi indispensável.

Para se capturar as cláusulas SQL submetidas ao SQL Server utilizamos a tabela de sistema syscacheobjects (Tabela 14), a qual pertence ao database master. A seguir descrevemos os principais campos desta tabela.

Após uma análise criteriosa dos campos da tabela de sistema syscaheobjects identificamos os campos necessários para se obter as cláusulas SQL submetidas ao *SQL Server*. Com base nestes campos elaboramos a consulta mostrada a seguir, a qual obtém os últimos comandos SQL executados pelo *SQL Server*, juntamente com a quantidade de vezes que o comando foi executado.

```
SELECT UPPER(sql) as sql, sum(usecounts) as usecounts
FROM master.dbo.syscacheobjects
WHERE objtype='Prepared' and cacheobjtype='Executable Plan' and sql like '()%'
GROUP BY UPPER(sql)
```

Para obter o plano de execução de uma determinada cláusula SQL previamente capturada utilizamos o comando set showplan_all on seguido da cláusula SQL capturada. A seguir, mostramos como obter o plano de execução para uma determinada cláusula SQL no *SQL Server* 2005.

```
SET SHOWPLAN_ALL ON
INSTRUÇÃO SQL EXECUTADA DURANTE A CONEXÃO DO SGBD
SET SHOWPLAN_ALL OFF
```

5.3 Extraindo Informações Estatísticas do SQL Server 2005

Com a finalidade de fornecer um suporte mais adequado para a análise da carga de trabalho capturada é necessário recuperar algumas informações estatísticas, tais como: o número de blocos (páginas de dados) ocupados por uma determinada relação, os índices existentes sobre uma determinada tabela, o método de acesso de um determinado índice, a altura de um determinado índice (árvore B^+), dentre outros.

Para obter essas informações, foi necessário o estudo e utilização do catalogo do sistema, anteriormente descritas. Dentre as tabelas e visões existentes no catálogo de sistemas, duas tabelas são suficientes para retornar os dados desejados. São elas: sysobjects e sysindexes. A tabela sysobjects cataloga as informações sobre os objetos do SGBD (tabelas, visões, procedimentos armazenado, etc). Já a tabela de sistema sysindexes fornece acesso a informações úteis sobre cada índice do banco de dados. Ela possui informações como

Coluna	Tipo de dado	Descrição
cacheobjtype	nvarchar(34)	Tipo do objeto na cache: Compiled Plan Executable Plan Parse Tree Cursor Parse Tree Extended Stored Procedure
objtype	nvarchar(16)	Tipo do objeto: Stored Procedure Prepared statement Ad hoc query ReplProc (replication procedure) Trigger View Default User table System table Check Rule
objid	int	ID do objeto armazenado na tabela de sistema sysobjects, quando o objeto em *cache* for um objeto do banco de dadoss (*procedures*, *views*, *triggers*, etc). Para objetos *ad hoc* ou *prepared* SQL, objid é um valor gerado internamente.
dbid	smallint	ID do banco de dados no qual o objeto em *cache* foi compilado.
usecounts	int	Número de vezes que o objeto em *cache* foi usado desde a ultima inspeção.
pagesused	int	Número de páginas de memória ocupadas pelo objeto na *cache*.
sqlbytes	int	Tamanho do nome do objeto. Esse valor pode ser usado para distinguir dois objetos cujos primeiros 128 caracteres dos seus nomes sejam iguais.
sql	nvarchar(256)	Texto da cláusula SQL.

Tabela 14: Principais Campos da Tabela de Sistema Syscacheobjects.

nome do esquema (*schema*), nome da tabela, nome do índice, e a definição do índice. A descrição dessas duas tabelas e de seus atributos mais importantes são mostradas nas Tabelas 15 e 16.

Apresentamos agora um conjunto de cláusulas SQL, construídas a partir das tabelas de sistema Sysobjects e Sysindexes, que buscam as informações estatísticas desejadas: A cláusula SQL a seguir obtém quantidade de *tuplas* de uma determinada tabela.

```
SELECT I.ROWS
FROM SYSOBJECTS T
JOIN  SYSINDEXES I ON I.ID=T.ID
WHERE T.NAME='TABLE1' AND T.TYPE='U' AND I.INDID<=1
```

Já cláusula SQL a seguir obtém a quantidade de blocos de uma determinada tabela seria:

```
SELECT I.DPAGES
FROM SYSOBJECTS T
JOIN  SYSINDEXES I ON I.ID=T.ID
WHERE T.NAME='TABLE1' AND T.TYPE='U' AND I.INDID<=1
```

Para se obter os índices definidos sobre uma determinada tabela, podemos utilizar a expressão mostrada a seguir:

```
SELECT I.NAME, I.KEYS
FROM SYSOBJECTS T
JOIN  SYSINDEXES I ON I.ID=T.ID
WHERE T.NAME='TABLE1' AND T.TYPE='U' AND I.INDID>=1
```

6 Conclusões

O desempenho dos servidores de bancos de dados é fator chave para o sucesso das aplicações de missão-crítica. Para estas aplicações, um baixo desempenho significa perdas de receita e de oportunidades de negócio. A fim de assegurar um desempenho sempre aceitável torna-se necessário monitorar continuamente a infra-estrutura dos servidores de bancos de dados, e, em caso de eventos inesperados que possam comprometer o desempenho do sistema, deve-se reagir de forma imediata, solucionando-se os problemas encontrados no menor espaço de tempo possível, com rapidez e eficiência. Os DBAs são os profissionais responsáveis por esta importante e complexa missão.

Durante a análise do SGBD, o DBA procura identificar as causas dos gargalos de desempenho e os problemas de contenção de recursos. Para isso, a principal fonte de informações do DBA são os metadados do próprio SGBD. Neste sentido, capturar a carga de trabalho submetida ao banco de dados (cláusulas SQL, planos de execução, custos,

Coluna	Tipo de dado	Descrição
name	sysname	Nome do objeto.
Id	int	Identificador do objeto.
xtype	char(2)	Tipo do objeto. Pode ser um dos seguintes valores: C = CHECK constraint D = Default or DEFAULT constraint F = FOREIGN KEY constraint L = Log FN = Scalar function IF = Inlined table-function P = Stored procedure PK = PRIMARY KEY constraint (type is K) RF = Replication filter stored procedure S = System table TF = Table function TR = Trigger U = User table UQ = UNIQUE constraint (type is K) V = View X = Extended stored procedure
uid	smallint	ID do usuário proprietário do objeto.
crdate	datetime	Data em que o objeto foi criado.
type	char(2)	Tipo do objeto. Pode ser um dos seguintes valores: C = CHECK constraint D = Default or DEFAULT constraint F = FOREIGN KEY constraint FN = Scalar function IF = Inlined table-function K = PRIMARY KEY or UNIQUE constraint L = Log P = Stored procedure R = Rule RF = Replication filter stored procedure S = System table TF = Table function TR = Trigger U = User table V = View X = Extended stored procedure

Tabela 15: Principais Campos da Tabela de Sistema Sysobjects.

Coluna	Tipo de dado	Descrição
id	int	ID da tabela (para indid = 0 or 255). Caso contrário, ID da tabela ao qual o índice pertence.
first	binary(6)	Ponteiro para a primeira página ou para o nó Raíz.
indid	smallint	ID do índice: 1 = Clustered index ¿1 = Nonclustered 255 = Entrada para tabelas que tenham dados texto ou imagem.
root	binary(6)	Para indid ¿= 1 e ¡ 255, root é um ponteiro para o nó raiz do índice. Para indid = 0 or indid = 255, root é um ponteiro para a última página de dados.
minlen	smallint	Tamanho mínimo de uma.
keycnt	smallint	Número de chaves.
dpages	int	Para indid = 0 ou indid = 1, dpages representa o número de páginas de dados. Para indid=255, este campo contém valor 0. Para outros casos, este valor representa o número de páginas de índices.
rowmodctr	int	Total de linhas inseridas, excluídas ou atualizadas desde a última vez que as estatísticas foram atualizadas.
xmaxlen	smallint	Tamanho máximo de uma linha.
maxirow	smallint	Tamanho máximo de uma linha referente a um nó não folha.
OrigFillFactor	tinyint	*Fillfactor* utilizado quando o índice foi criado. Pode ser útil quando se deseja recriar o índice e não se recorda do *fillfactor* utilizado quando o índice foi criado.
keys	varbinary(816)	Lista com os IDs das colunas que compõem a chave do índice.
name	sysname	Nome da tabela (para indid = 0 or 255). Caso contrário, nome do índice.
rows	int	Número de linhas da tabela (se indid = 0 and indid = 1). Para indid = 255, rows recebe valor 0.

Tabela 16: Principais Campos da Tabela de Sistema Sysindexes.

etc.), bem como informações estatísticas (número estimado de linhas e de páginas de uma tabela, índices existentes, etc.), torna-se de fundamental importância.

Contudo, a forma de se obter (consultar) estes metadados depende do fabricante do SGBD. Assim, a maneira como o DBA recupera as últimas cláusulas SQL executadas no *Oracle* 10g é completamente diferente da forma como esta informação é obtida através dos metadados do *SQL Server* 2005.

Neste trabalho realizamos um estudo dos metadados dos principais SGBDs comerciais: *PostgreSQL*, *Oracle* 10g e *SQL Server* 2005. Além disso, elaboramos e apresentamos uma série de *scripts* capazes de capturar os principais metadados e informações estatísticas utilizadas no processo de sintonia (*tuning*), análise e resolução de problemas de desempenho. Desta forma, este trabalho constitui um guia para facilitar a identificação e utilização dos metadados e estatísticas necessários ao processo de identificação e solução de problemas de desempenho.

Referências

[1] D. Burleson. *Creating a Self-Tuning Oracle Database*. Rampant, 2004.

[2] B. Dageville, D. Das, K. Dias, K. Yagoub, M. Zait, and M. Ziauddin. Automatic sql tuning in oracle 10g. In *In Proceedings of the International Conference on Very Large Databases (VLDB)*, pages 1098–1109, 2004.

[3] K. Delaney. *Inside Microsoft SQL Server 2005*. Microsoft Press, 2006.

[4] E. Geschwinde and H. Jungerschoning. *PostgreSQL Developer's Handbook*. Sams Publishing, 2002.

[5] B. Momjiam. *PostgreSQL - Introduction and Concepts*. Addison-Wesley, 2001.

[6] Oracle v$ view list. `http://www.dba-oracle.com/menu`, 2008.

[7] Postgresql. `http://www.postgresql.org`.

[8] O coletor de estatísticas do postgresql. `http://www.javalinux.com.br/javalinux/pg74/monitoring-stats.html`, 2008.

[9] J. Shapiro. *Microsoft SQL Server 2005 The Complete Reference*. Osborne, 2006.

[10] A. Silberschartz, H. F. Korth, and S. Sudarshan. *Sistema de Banco de Dados*. Campus, 2006.

[11] Sql server books on-line. `http://www.microsoft.com/`, 2008.