PUC

ISSN 0103-9741

Monografias em Ciência da Computação
n° XX/07

Extraindo Metadados 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. XX/07 ISSN: 0103-9741
Editor: Prof. Carlos José Pereira de Lucena Dezembro, 2007

Extraindo Metadados 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 the database servers is a key factor for the mission-critical application success. In order to assure an always acceptable performance becomes necessary continuously to monitor the database server's infrastructure. Case an unexpected event, that decree the system performance, be detected, it must be reacted of immediate form, solving the problems found with rapidity and efficiency. The main source of information used in the database performance monitoring is the proper SBMS metadata. However, the form of obtain (to consult) these metadata depend on the DBMS vendor. In this work we carry through an exhausting study of the metadata of the main commercial DBMSs: Postgres 8.2, Oracle 10g and SQL Server 2005. Moreover, we elaborate and present a series of scripts capable to capture the main metadata and statistical information used in the performance problems analysis and resolution process.

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 uma performance sempre aceitável torna-se necessário monitorar continuamente a infra-estrutura dos servidores de bancos de dados. Caso seja detectada a ocorrência de um evento inesperado que possa comprometer a performance do sistema, deve-se reagir de forma imediata, solucionando-se os problemas encontrados com rapidez e eficiência. A principal fonte de informações utilizada no monitoramento da performance de bancos de dados são os metadados do próprio SGBD. Porém, a forma de se obter (consultar) estes metadados depende do fabricante do SGBD. Neste trabalho realizamos um estudo exaustivo dos metadados dos principais SGBDs comerciais: Postgres 8.2, 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 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 of 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, uma baixa performance significa perdas de receita e de oportunidades de negócio. A fim de assegurar uma performance 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 à eventos de sistema inesperados, além de ajustar (sintonizar) os diversos parâmetros de performance. Desta forma, praticamente todos os bancos de dados, que armazenam grandes volumes de dados, são gerenciados por administradores (DBAs - *Database Administrators*), os quais devem ser especialistas nas tarefas de administração e sintonia (*tuning*) de bancos de dados (ajuste dos parâmetros do SGBD (Sistema de Gerenciamento de Banco de Dados) de acordo com as características da carga de trabalho das aplicações).

Assim, os DBAs são responsáveis pela análise contínua do funcionamento do SGBD, descoberta de problemas de performance e, principalmente, pela solução rápida e eficiente destes problemas. Durante a análise do SGBD, o DBA procura identificar as causas dos gargalos de performance 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 os comandos SQL executados, custo estimado, tempo de resposta (tempo decorrido entre enviar a consulta SQL e o retorno do resultado), número estimado de linhas retornadas, ou número de *sequential scans*, dados *time-series* de parâmetros de performance e armazenar estes dados em um repositório, para diagnósticos rápidos, ou para análises futuras e planejamento, torna-se de fundamental importância.

Para avaliar a performance de um SGBD o tempo de resposta é o métrica mais importante, uma vez que este é componente do desempenho que é percebido pelo usuário final. Logo, identificar as instruções SQL que proporcionam os maiores tempos de resposta é uma tarefa fundamental. Por exemplo, é importante identificar quais instruções SQL foram responsáveis por picos intermitentes de consumo de CPU, leituras em disco, etc.

As ações do DBA, para intervir quando detectado um problema de performance, 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 de software, também os bancos de dados são complexos: há uma boa quantidade de variações pelas implementações dos diferentes fornecedores. Como resultado, há significativas variações de desempenho em diferentes tarefas. Um sistema pode ser o mais eficiente para uma determinada tarefa; outro pode ser o mais eficiente para uma tarefa diferente [22].

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. Seria natural supor que uma taxa mais elevada de amostragem fosse melhor, mas na realidade cada medição pode acrescentar um custo extra ao sistema sendo medido.

Entretanto, associar medidas de desempenho a uma combinação de tarefas 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 chamado nível de referência de desempenho, mais conhecido como *benchmark* [22].

Neste contexto, a fim de descobrir problemas de performance e assegurar que os SGBDs mantenham sempre um desempenho adequado, os DBAs necessitam realizar uma análise contínua do funcionamento do SGBD. Para isso, é necessário medir (e capturar) periodicamente a performance do SGBD. Essas medições ("*snapshots*") são realizadas através de consultas aos metadados. Assim, é possível recuperar, por exemplo, as instruções SQL executadas, os planos de execução gerados para estas consultas, o custo estimado, o tempo de resposta, o número estimado de linhas retornadas, os índices existentes, o nível de fragmentação de um determinado índice, o número de páginas (blocos) de uma determinada tabela, etc.

Porém, a forma de se obter (consultar) estes metadados depende do fabricante do SGBD. Desta forma, a maneira como o DBA recupera as últimas cláusulas SQL executadas no Oracle 10g é completamente diferente da forma como esta informação é recuperada 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, 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 recuperados 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 Postgres 8.2, a seção 4 ilustra como recuperar os metadados do Oracle 10g, na seção 5 discutimos a extração dos metadados no SQL Server 2005 e a seção 6 conclui este trabalho.

2 Metadados

Os metadados são frequentemente descritos como “dados sobre dados”. Ou seja, não são mais do que informações adicionais (além da informação espacial e tabular) que é necessária para que os dados se tornem úteis. Por outras palavras, são um conjunto de características sobre os dados que não estão normalmente incluídas nos dados propriamente ditos.

A partir desta definição pode então concluir-se os usos mais significativos dos metadados. São eles:

- 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 dato foi lido ou atualizado pela última vez.

- Possibilitar às aplicações executarem checagem de tipo, validação, formatação, etc.

- Possibilitar a análise do desempenho do SGBD.

Alguns exemplos de metadados incluem: modelos de dados, esquemas, tabelas, colunas, índices, últimas cláusulas SQL executadas, planos de execução das consultas executadas, etc.

3 Postgresql

3.1 Conceitos e Características

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 [15].

O *Postgresql* é um banco de dados de código fonte aberto, o qual suporta grande parte do padrão SQL-2003, além de oferecer muitas funcionalidades modernas, tais como:

- Comandos Complexos;
- Gatilhos (*Triggers*)
- Visões Materializadas
- Controle de Concorrência Multiversão.

Além disso o *Postgresql* pode ser estendido pelos seus usuários de diversas maneiras, por exemplo:

- Tipos de Dados;
- Funções
- Regras (*Rules*)
- Estruturas de Índices.
- Linguagens Procedurais.

Devido o *Postgresql* ser distribuído pela a licença BSD (*Berkeley Software Distribution*), ela permite que usuários façam qualquer coisa com o código, incluindo vender os binários sem os códigos-fontes. Assim, o *Postgresql* pode ser utilizado, modificado e distribuído para qualquer finalidade, seja ela privada, comercial ou acadêmica.

3.2 Catálogo do Sistema

O catálogo do sistema é o local onde o gerenciador de banco de dados armazena os meta-dados dos seus esquemas. Estes meta-dados 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 e visões. A seguir, apresentaremos uma breve descrição de cada uma destas tabelas e visões.

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 de tabela ("atributos")
pg_cast	casts (conversões de tipos de dado)
pg_class	tabelas, índices, seqüências, visões ("relações")
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 3.1 - Catálogo de Sistema do *Postgresql*

Na Figura 3.1 podemos visualizar, de forma simplificada, a estrutura do catálogo de sistema do *Postgresql*.

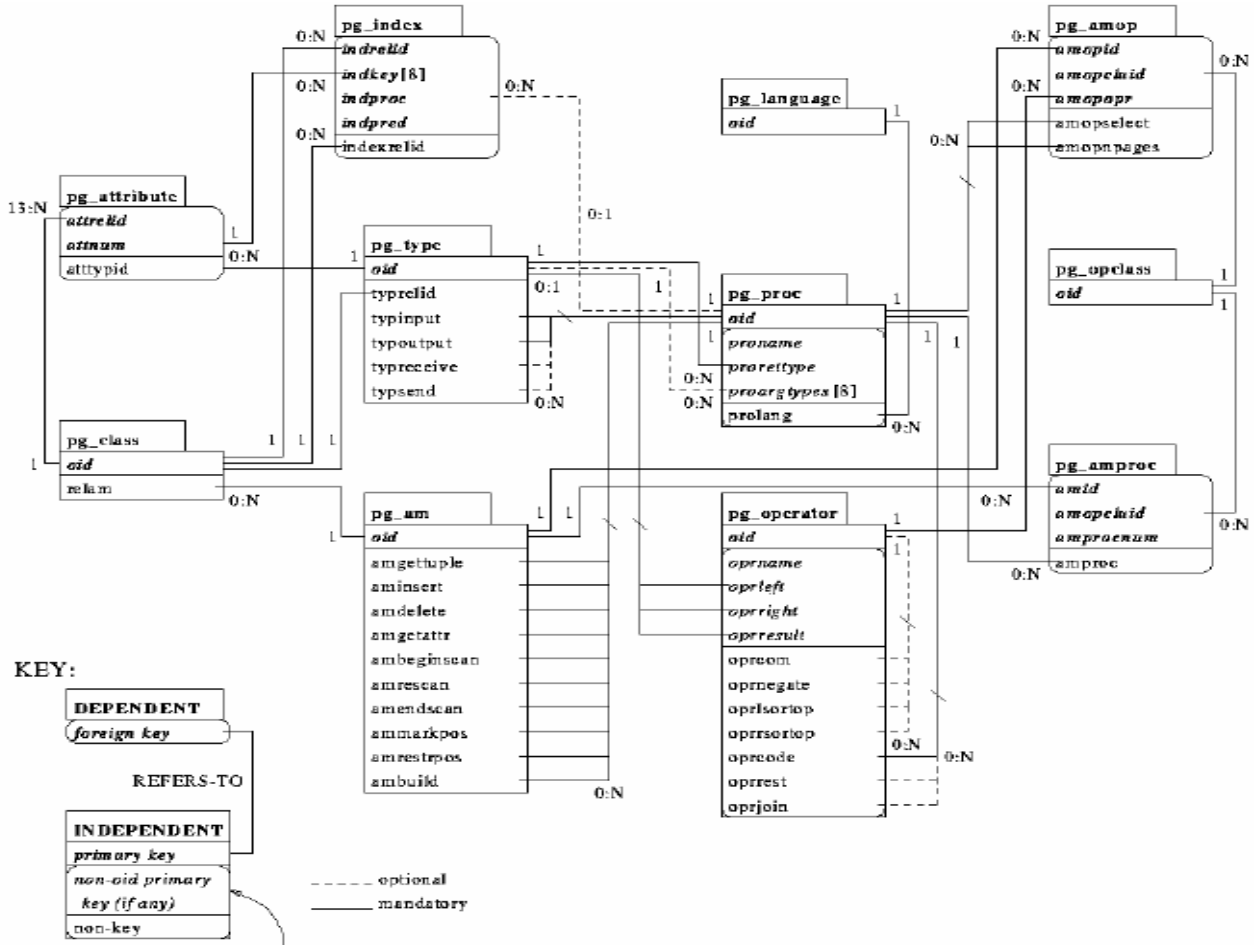

Figura 3.1 – Estrutura do Catálogo de Sistema do *Postgresql*.

3.3 Visões do Sistema

Assim como temos as tabelas, temos as visões (*views*) internas do sistema. A partir destas visões, o *Postgresql* fornece um acesso conveniente às tabelas (do catalogo) do sistema mais freqüentemente utilizadas. Estas *views* estão listadas, e brevemente descritas, abaixo:

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 3.2 - Visões de sistema do Postgresql

3.4 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, o 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.

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 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 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 linha, respectivamente.

A fim de habilitar a coleta de estatísticas do *Postgresql* é necessário configurar o arquivo *postgres.conf*, na parte de *RUNTIME STATISTICS*, da seguinte maneira:

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

# - Statistics Monitoring -

#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
```

Quadro 3.1 – Configuração do Postgres.conf

3.5 Visões de Estatísticas

Estão disponíveis 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.

Ao se utilizar as estatísticas para monitorar a atividade corrente, é importante ter em mente que as informações não são atualizadas instantaneamente. Cada processo servidor individual transmite os novos contadores de acesso a bloco e a linha para o coletor logo antes de ficar ocioso; 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). Portanto, 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, primeiro este busca 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, porque permite realizar várias consultas às estatísticas e correlacionar os resultados sem se preocupar com números variando por baixo.

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 superusuário, 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_sys_tables	O mesmo que pg_stat_all_tables, exceto que somente são mostradas as tabelas do sistema.
pg_stat_user_tables	O mesmo que pg_stat_all_tables, exceto que somente são mostradas as tabelas de usuário.

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_stat_sys_indexes	O mesmo que pg_stat_all_indexes, exceto que somente são mostrados os índices das tabelas do sistema.
pg_stat_user_indexes	O mesmo que pg_stat_all_indexes, exceto que somente são mostrados os índices das tabelas de usuário.
`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_sys_tables`	O mesmo que `pg_statio_all_tables`, exceto que somente são mostradas as tabelas do sistema.
`pg_statio_user_tables`	O mesmo que `pg_statio_all_tables`, exceto que somente são mostradas as tabelas de usuário.
`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.
`pg_statio_sys_indexes`	O mesmo que `pg_statio_all_indexes`, exceto que somente são mostrados os índices das tabelas do sistema.
`pg_statio_user_indexes`	O mesmo que `pg_statio_all_indexes`, exceto que somente são mostrados os índices das tabelas de usuário.
`pg_statio_all_sequences`	Para cada objeto de seqüência do banco de dados corrente, o número de blocos de disco lidos e de acertos no `buffer` para a seqüência.
`pg_statio_sys_sequences`	O mesmo que `pg_statio_all_sequences`, exceto que somente são mostradas as seqüências do sistema (Atualmente não está definida nenhuma seqüência do sistema e, portanto, esta visão está sempre vazia).
`pg_statio_user_sequences`	O mesmo que `pg_statio_all_sequences`, exceto que somente são mostradas as seqüências de usuário.

Tabela 3.3 - Visões de Estatísticas do Postgresql

3.6 Funções de Acesso às Estatísticas

Podem ser criadas outras formas de ver as estatísticas, escrevendo consultas que utilizam as mesmas funções subjacentes de acesso às estatísticas utilizadas pelas visões padrão. Esta funções estão listadas na tabela 3.4. As funções de acesso por banco de dados, recebem como argumento o OID do banco de dados que identifica para qual banco de dados é o relatório. As funções por tabela e por índice recebem o OID da tabela ou do índice, respectivamente (Deve ser observado que somente podem ser vistos 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 de um ao número de processos servidor ativos no momento.

Função	Tipo Retornado	Descrição
`pg_stat_get_db_numbackends(oid)`	`integer`	Número de processos servidor ativos conectados ao banco de dados
`pg_stat_get_db_xact_commit(oid)`	`bigint`	Transações efetivadas no banco de dados
`pg_stat_get_db_xact_rollback(oid)`	`bigint`	Transações canceladas no banco de dados
`pg_stat_get_db_blocks_fetched(oid)`	`bigint`	Número de solicitações de busca de blocos de disco para o banco de dados
`pg_stat_get_db_blocks_hit(oid)`	`bigint`	Número de solicitações de busca de blocos de disco para o banco de dados encontradas no `cache`
`pg_stat_get_numscans(oid)`	`bigint`	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)`	`bigint`	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_fetched(oid)`	`bigint`	Número de linhas válidas (não expiradas)

		da tabela buscadas por varreduras seqüenciais quando o argumento é uma tabela, ou buscadas por varreduras de índice, utilizando este índice, quando o argumento é um índice
`pg_stat_get_tuples_inserted(oid)`	`bigint`	Número de linhas inseridas na tabela
`pg_stat_get_tuples_updated(oid)`	`bigint`	Número de linhas atualizadas na tabela
`pg_stat_get_tuples_deleted(oid)`	`bigint`	Número de linhas excluídas da tabela
`pg_stat_get_blocks_fetched(oid)`	`bigint`	Número de solicitações de busca de bloco de disco para a tabela ou índice
`pg_stat_get_blocks_hit(oid)`	`bigint`	Número de solicitações de busca de bloco de disco encontradas no `cache` para a tabela ou o índice
`pg_stat_get_backend_idset()`	`conjunto de integer`	Conjunto de IDs de processos servidor ativos no momento (de 1 ao número de processos servidor ativos). Veja o exemplo de utilização no texto.
`pg_backend_pid()`	`integer`	ID de processo do processo servidor conectado à sessão corrente
`pg_stat_get_backend_pid(integer)`	`integer`	ID de processo do processo servidor especificado
`pg_stat_get_backend_dbid(integer)`	`oid`	ID de banco de dados do processo servidor especificado
`pg_stat_get_backend_userid(integer)`	`oid`	ID de usuário do processo servidor especificado
`pg_stat_get_backend_activity(integer)`	`text`	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)`	`timestamp with time zone`	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)
`pg_stat_reset()`	`boolean`	Reinicia todas as estatísticas atualmente coletadas

Tabela 3.4 – Funções de Acesso às Estatísticas do Postgresql

3.7 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 as visões de estatísticas e as funções de acesso às estatisticas que foram utilizadas.

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 superusuário, 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_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)

Tabela 3.5 – Visões de Estatísticas Utilizadas

Nome da Função	Descrição
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_idset()	Conjunto de IDs de processos servidor ativos no momento (de 1 ao número de processos servidor ativos).

Tabela 3.6 – Funções de Acesso às Estatísticas Utilizadas

Após uma análise criteriosa dos campos das visões e das funções mostradas nas tabelas 3.5 e 3.6, percebemos que as três funções poderiam relacionar-se para fornecer as informações necessárias. Geramos assim uma consulta, baseada na função *pg_stat_get_backend_idset(),* a qual fornece uma maneira conveniente de gerar uma linha para cada processo servidor ativo, mostrando o PID (identificação do processo) e o seu comando corrente. O Quadro 3.2 apresenta a cláusula SQL utilizada para obter o comando SQL que está em execução.

```
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;
```

Quadro 3.2. – Recuperando o Comando Corrente.

Para obter o plano de execução de uma determinada clásula SQL usamos o comando "*EXPLAIN* + instruções SQL capturada".

EXPLAIN + INSTRUÇÃO SQL EXECUTADA DURANTE A CONEXÃO DO SGBD

Quadro 3.3 – Obtendo o Plano de Execução

3.8 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 catologo de sistemas, uma tabela e uma visão são suficientes para retornar os dados desejados. Sâo eles: a tabela de sistema *PG_CLASS* e a visão é *PG_INDEXES*. A 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 3.7.

Coluna	Tipo de dado	Referencia	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 planejador. Atualizado pelos comandos VACUUM, ANALYZE e uns poucos comandos de DLL como o CREATE INDEX.
reltuples	flot4		Número de linhas na tabela. Esta é apenas uma estimativa utilizada pelo planejador. 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 relhasindex quando descobre que a tabela não possui índices.
relhaspkey	bool		Verdade se a tabela possui (ou já possuiu) uma chave primária.

Tabela 3.7. Principais campos da visão PG_CLASS

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 indice.

Coluna	Tipo	Referencia	Descrição
schemaname	name	pg_namespace.nspname	Nome do esquema que contém a tabela e o índice
tablename	name	pg_class.relname	Nome da tabela para a qual o índice se destina
indexname	name	pg_class.relname	Nome do índice
tablespace	name	pg_tablespace.spcname	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 3.8. Principais campos da visão PG_INDEXES

Para se obter informações sobre uma determina tabela ou sobre uma determinada visão (*view*) do sistema, foram criadas várias cláusulas SQL, que utilizam a *tabela* de sistema *PG_CLASS* e a *view* PG_INDEXES (as quais se relacionam através dos campos da tabela *PG_TABLE)*, para retornar os dados desejados: nome da tabela, número de linhas, número de blocos, metodo de acesso do indice e os índices existentes em uma tabela. As clausulas SQL criadas estão descrita nos quadros abaixo:

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

Quadro 3.4 - Cláusula que obtém quantidade de tuplas de uma 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 + "'
);
```

Quadro 3.5 - Cláusula que obtém quantidade de blocos "paginas" de uma tabela .

```
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 + "'
                           );
```

Quadro 3.6 - Cláusula que obtém o metodo de acesso do indice.

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

Quadro 3.7- Cláusula que obtém os indices de uma tabela.

4 Oracle 10g

4.1 Conceitos e Características

O banco de dados *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, e ainda proporciona facilidades para o desenvolvimento de aplicações *web* através do *Oracle Application Express*.

A versão 10g está voltada para o conceito de computação em *grid*. Neste sentido, busca facilitar a implementação de soluções de alta escalabilidade, fornecendo suporte para *clusters* de dados, bancos de dados distribuídos e bancos de dados em *grids*. Alem disso, fornece uma coleção de novos recursos, tais como:

- Suporte a expressões regulares (padrão IEEE/POSIX) que oferecem muito mais poder e flexibilidade na comparação de texto do que o velho operador *LIKE*;
- Novos utilitários de exportação e importação de dados com a tecnologia "*Data Pump*", para movimentação de dados em massa com acesso direto a alta velocidade;
- Melhor adequação a sistemas de *Bussiness Inteligence* (BI), com novos recursos para mineração de dados (*data mining*) e bioinformática;

- Maior suporte a Java e XML, com JVM interna agora compatível com J2SE 1.4 e mais recursos de JDBC 3.0 e *Web Services*;
- A facilidade de *Automatic Storage Management* (ASM), que simplifica o gerenciamento e organização flexível de armazenamento com um gerenciador de volumes (*volume manager*) integrado ao banco de dados.

4.2 Dicionário de Dados

O *Oracle 10g* possui um dicionário de dados que é um conjunto de tabelas especiais, criadas no 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 *views* 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. A seguir uma breve descrição de cada grupo:

USER_: exibe todos os objetos criados no próprio *schema* do usuário.

ALL_: lista todos os objetos, acessíveis ao usuário, associados a um ou vários *schemas*.

DBA_: mostra todos os objetos do banco de dados.

Além destes três grupos, o dicionário de dados do Oracle 10g é constituído também pelas visões dinâmicas V$ (visões que começam com o prefixo V$).

Na Figura 4.1 podemos visualizar, de forma simplificada, a estrutura do dicionário de dados do *Oracle*.

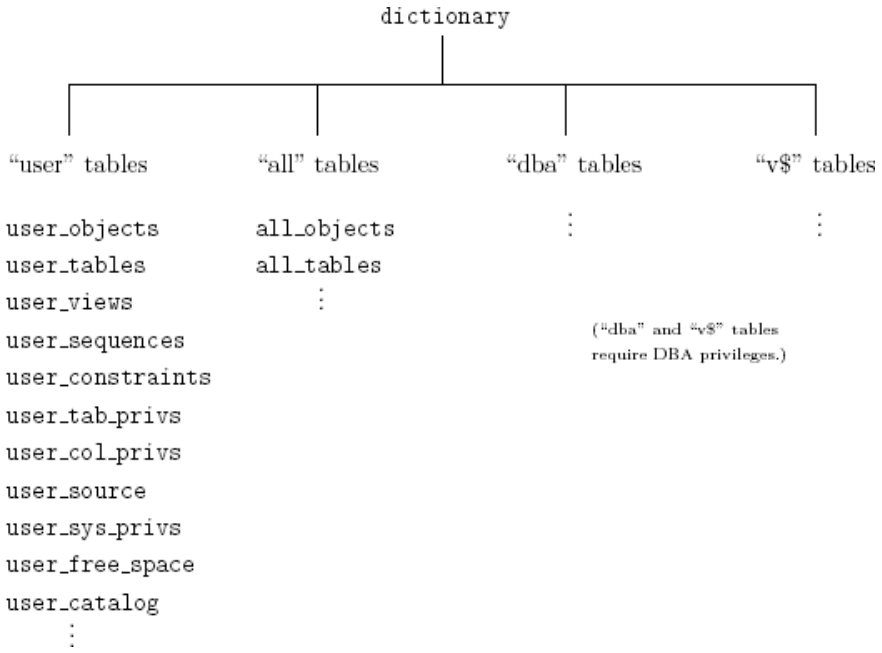

Figura 4.1 - Estrutura do Dicionário de Dados do Oracle

A seguir descrevemos as principais visões pertencentes ao grupo DBA_. Através das visões deste grupo podemos obter os metadados dos objetos armazenados no banco de dados.

Objects: mostra todos os objetos do banco de dados, como tabelas, índices, partições, lobs, views.

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, views 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, nome, tabela, localização, tamanho, entre 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*.

Sequences: possui a listagem completa de todos os objetos *sequence* criados no banco de dados.

Synonyms: mostra todos os *synonyms* criados na banco de dados.

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 *views* do banco de dados.

Updatable_Columns: permite saber se as colunas de uma tabela podem ser alteradas, inseridas ou excluídas.

Types: lista todos os tipos criados no banco de dados.

Type_Attrs: lista todos os atributos criados para os tipos do banco de dados.

Type_Methods: descreve os métodos criados para os tipos do banco de dados.

Nested_Tables: mostra a relação entre tabelas "nested" (aninhada) e a tabela a que está relacionada.

Object_Tables: fornece todas as informações sobre os objetos da tabela.

Lobs: permite saber se a LOB (Large Objects) está com índice, em qual tabela ela está, nome do segmento, entre outras informações.

Method_Params: lista todos os parâmetros associados a cada método dos tipos do banco de dados.

Method_Results: descrição dos resultados dos tipos do banco de dados.

Outro grupo de visões de extrema importância que compõe o dicionário de dados do Oracle 10g é grupo das visões dinâmicas V$. Estas visões exercem um papel importantíssimo no monitoramento das atividades do SGBD. Elas constituem as fontes para se obter a estimativa real da carga de trabalho submetida ao banco de dados. O Oracle 10g acrescentou melhorias ao dicionário de dados, com novas visões dinâmicas de desempenho. Dentre estas está a visão *V$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. A organização, em alto nível, desses grupos está descrita abaixo:

- High Availability and Recoverability
    - Backups/Archives/Recovery
    - Logminer
    - Standby Databases and Dataguard
    - Flashback Databases
- **Specific Technologies**
    - Oracle Streams
    - Olap
    - Replication/Materialized Views
    - Direct Loader
- **Database and Instance Performance**
    - Caches/Queues
    - Overall System
    - Sessions/Resource Allocation
    - Cursors/Sql Statements
    - Latches/Locks
    - Metrics
    - Multithreaded/Shared Servers
    - Rollback Segments/System Managed Undo Segments
    - Parallel Query
    - I/O

- **Database and Instance Configuration**

    - Databases/Instances
    - Redo Logs
    - Security / Privileges
    - Parameters
    - Control Files

- **Static Database And Instance Information**

    - Fixed Views

Uma característica importantíssima 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.

4.3 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. A seguir, descreveremos os grupos de *views* que foram analisados e utilizadas para extrair a carga de trabalho submetida ao *Oracle*.

Como o dicionário de dados do *Oracle* e composto por diversos grupos, os quais podem conter vários subgrupos (os quais abrangem um determinado número de *views*), o processo de pesquisa selecionou algumas visões, de grupos distintos, as quais, em conjunto, fornecem os dados desejados. A estrutura dos grupos de visões analisadas está descrita na Figura 4.2.

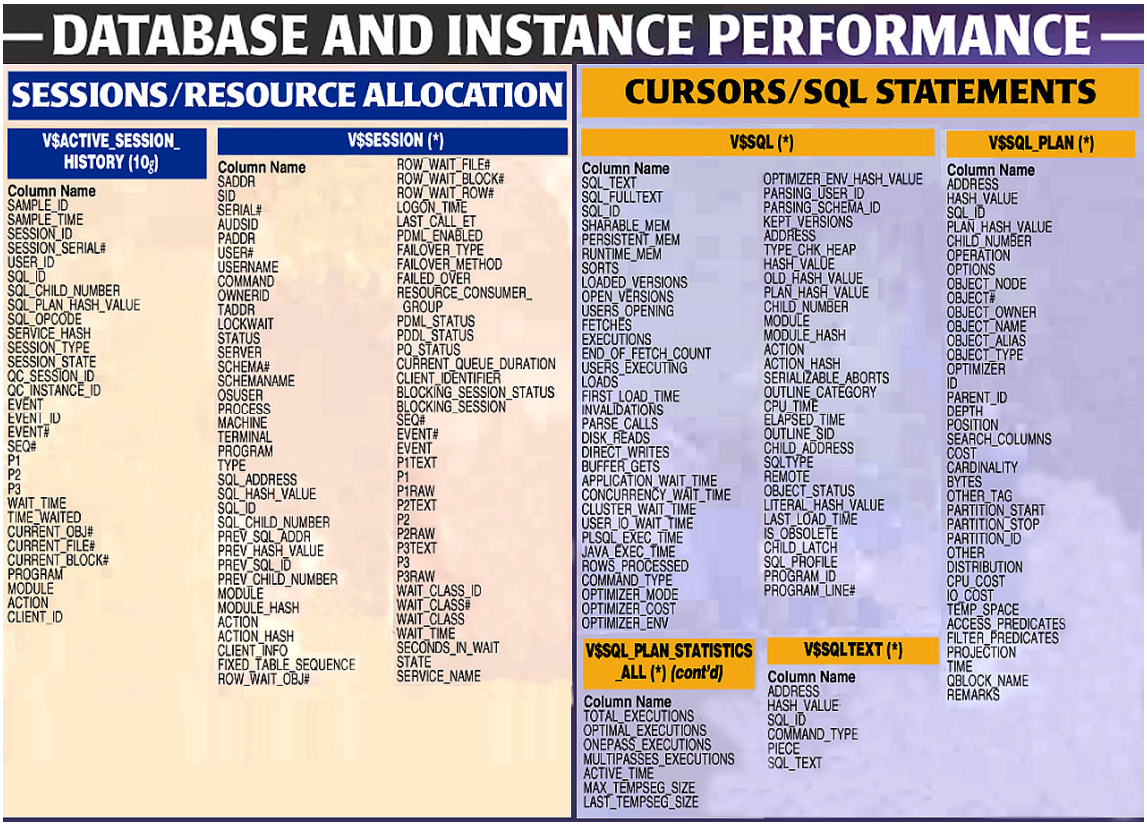

Figura 4.2 – Estrutura das *views* V$ analisadas

Após uma análise criteriosa dos campos das visões mostradas na Figura 4.2, percebeu-se que três delas poderiam relacionar-se para fornecer as informações necessárias, são elas: *V$ACTIVE_SESSION_HISTORY*, *V$SQL* e *V$SQL_PLAN*. A seguir, descreveremos essas visões e seus atributos mais importantes.

V$ACTIVE_SESSION_HISTORY:

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 4.1.

Coluna	Tio 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 4.1. Principais campos da visão V$ACTIVE_SESSION_HISTORY

V$SQL:

Esta visão lista as informações estatísticas e o texto completo de cada cláusula SQL executada. A Tabela 4.2 mostra, em detalhes, os principais campos desta visão.

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 o mesmo ou não (ao invés de comprar 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 do banco.

Tabela 4.2. Principais campos da visão V$SQL

V$SQL_PLAN:

A Visão *V$SQL_PLAN* contém a informação do plano de execução de cada instrução SQL submetida ao banco. Esta é considerada a principal *view*, pois é nela que se encontra o plano de execução, custo de I/O, número de linhas e outros detalhes das instruções SQL. A Tabela 4.3 mostra, em detalhes, os campos desta visão que foram utilizados. Já a Figura 4.3 mostra o relacionamento entre as três visões utilizadas (*V$ACTIVE_SESSION_HISTORY*, *V$SQL* e *V$SQL_PLAN*).

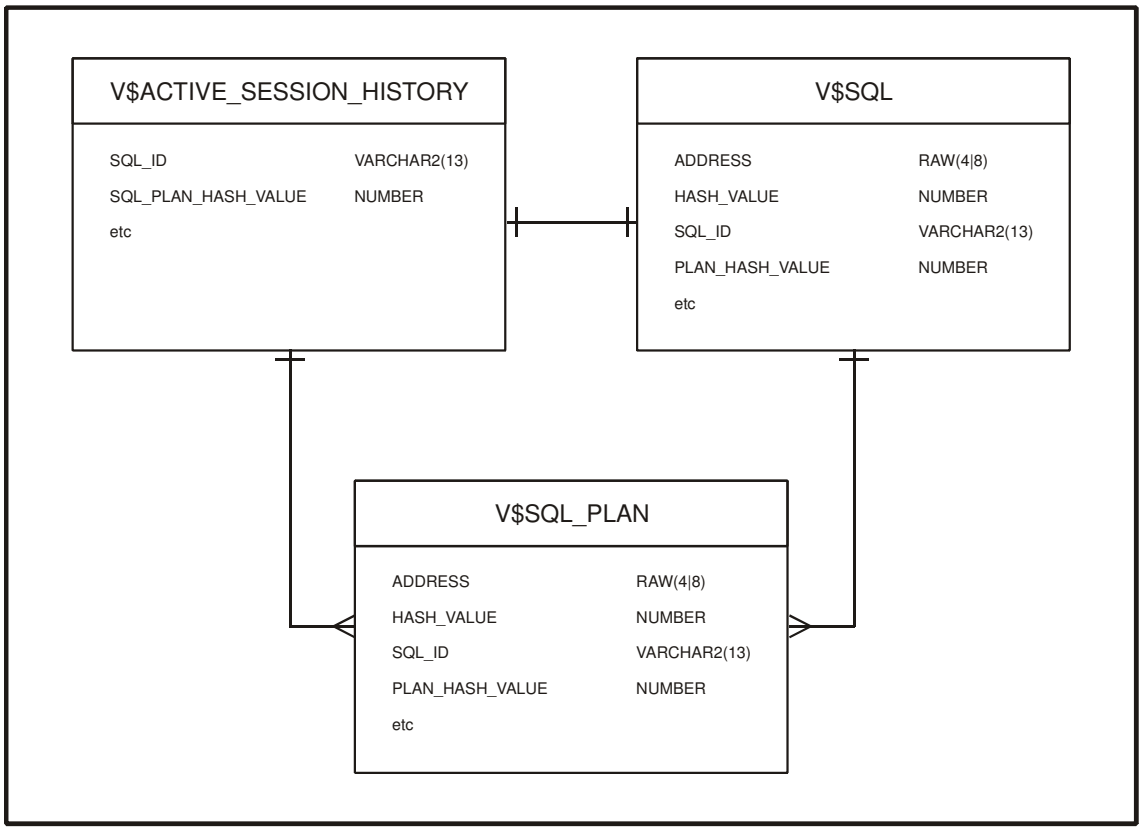

Figura 4.3 - Relacionamento entre as views v$ utilizadas

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 pode ser usadas para adicionar informação específica na view V$SQLAREA.
SQL_ID	VARCHAR2(13)	Identificador SQL do cursos 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 I/O da operação como estimado pela abordagem baseada no custo. Para instruções que usam a abordagem baseada em regras, a coluna é NULL.

Tabela 4.3. Principais campos da visão V$SQL_PLAN

O Quadro 4.1 mostra uma cláusula que recupera as informações sobre as últimas consultas executadas no Oracle 10g. Vale ressaltar que somente as consultas presentes na *view 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%'
```

Quadro 4.1 – Cláusula que obtém as últimas consultas executadas no servidor

O Quadro 4.2 mostra uma cláusula que recupera do 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
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%'
```

Quadro 4.2 - Cláusula que obtém os planos de execução das últimas consultas executadas

4.4 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 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 dicionário de dados, anteriormente descrito. Neste sentido, exploramos as *views* do grupo *DBA_*. Dentre as *views* 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.

DBA_TABLES:

Esta *view* 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 6.

Coluna	Tipo de dado	NULL	Descrição
OWNER	VARCHAR2(30)	NOT NULL	Proprietário da tabela
TABLE_NAME	VARCHAR2(30)	NOT NULL	Nome da tabela
NUM_ROWS*	NUMBER		Número de linhas da tabela
BLOCKS*	NUMBER		Número de blocos de dados usados

Tabela 4.4. Principais campos da visão DBA_TABLES

DBA_INDEXES:

Essa *view* 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 3.5 mostra, em detalhes, os campos que foram utilizados. Já a Figura 3.6 mostra o relacionamento entre as visões *DBA_TABLES* e *DBA_INDEXES*.

Coluna	Tipo de dados	NULL	Descrição
OWNER	VARCHAR2(30)	NOT NULL	Proprietário do índice
INDEX_NAME	VARCHAR2(30)	NOT NULL	Nome do índice
TABLE_OWNER	VARCHAR2(30)	NOT NULL	Proprietário da tabela indexada
TABLE_NAME	VARCHAR2(30)	NOT NULL	Nome da tabela indexada
BLEVEL*	NUMBER		Nível da árvore B+ : profundidade da raiz aos nós folha. Uma profundidade 0 indica que a raiz e a folha são o mesmo

Tabela 4.5. Principais campos da visão DBA_INDEXES

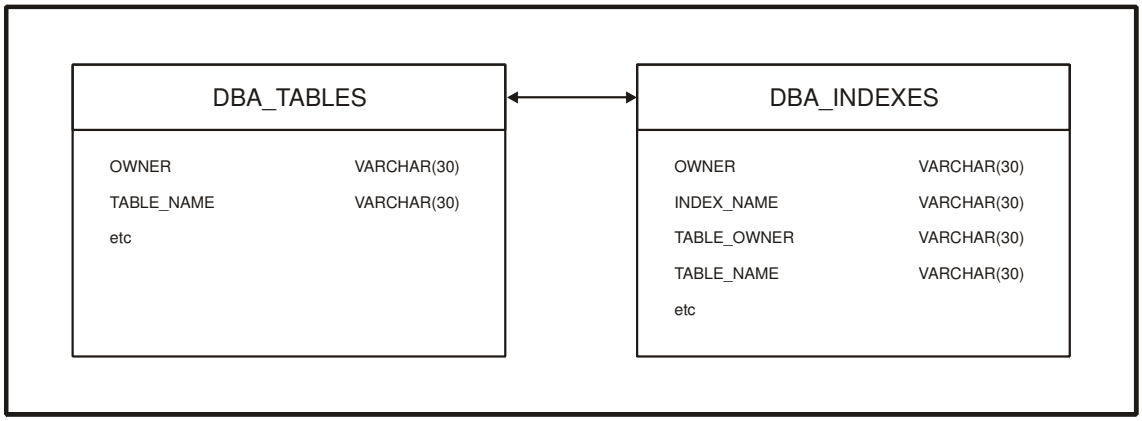

Figura 4.4 - Relacionamento entre DBA_TABLES e DBA_INDEXES

Para se obter as informações estatísticas sobre uma determina tabela, foi criada uma cláusula SQL que utiliza as *views* 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 no Quadro 4.3.

```
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(+)
```

Quadro 4.3 - Cláusula que obtém dados estatísticos de uma tabela.

5 SQL Server 2005

5.1 Conceitos e Características

A Microsoft reformulou a família de produtos SQL Server 2005 para melhor satisfazer as necessidades de segmentos específicos de clientes através do lançamento de quatro novas edições: *Express, Workgroup, Standard e Enterprise*.

Estas novas edições oferecem uma série de funcionalidades – desde elevada disponibilidade e escalabilidade até ferramentas avançadas de *Business Intelligence* – projetadas para ampliar as capacidades dos utilizadores de toda a organização. Enquanto base de dados central de baixo custo, o SQL Server 2005 oferece um valor e funcionalidade sem precedentes em comparação com outras soluções concorrentes. Para sistemas empresariais exigentes, o SQL Server 2005 representa um gigantesco passo a frente, reduzindo o período de inatividade da aplicação, elevando a escalabilidade e o desempenho e reforçando os controles de segurança e auditoria [20].

Na medida em que o SQL Server faz parte do *Windows Server System*, os clientes colhem também benefícios como um menor custo total de propriedade e um tempo de desenvolvimento mais rápido. Isso se deve à maior facilidade de gestão e integração resultantes da estratégia de engenharia comum implementada nos produtos da família *Windows Server Systems*.

As principais características do SQL Server 2005 são:

- Elevada disponibilidade
- Novas ferramentas de gestão
- Melhorias na segurança
- Escalabilidade
- Aumento da produtividade do programador
- Funcionalidades de Business Intelligence

5.2 Catálogo do Sistema

Quando criamos uma tabela no SQL Server, com suas colunas, índices, tipos de dados, etc., estas informações (metadados) são armazenadas no *Database Catalog*, um conjunto de tabelas e visões que armazenam informações sobre os objetos que o usuário criou no banco de dados, como definição de tabela, código fonte de uma *Stored Procedure* ou definição de uma *view*.

Pois bem, 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 *stored procedures* do SQL Server
4. OLE DB *schema rowsets*.
5. ODBC *catalog functions*.

5.2.1 Através de 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. A *Microsoft* não 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.

Abaixo seguem algumas tabelas de sistema e um breve comentário sobre cada uma:

sysobjects	Armazena informações sobre todos os objetos do B.D.
sysindexes	Armazena informações específicas sobre índices do B.D.
syscolumns	Armazena todas as informações sobre todas as tabelas do B.D.
syscomments	Armazena o código fonte de Stored Procedures e funções do B.D.
syslocks*	Armazena informações sobre os locks (travas) dos objetos do B.D.
sysdatabases*	Armazena informações sobre os Bancos de dados do Servidor SQL Server

Tabela 5.1. Principais tabelas de sistema do SQL Server

Obs: As tabelas de sistema marcadas com * somente existem no B.D. (banco de dados) MASTER.

5.2.2 Através das 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*).

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, enquanto no SQL Server 2005 haviam aproximadamente 50 tabelas de sistema. A figura a seguir mostra como o SQL Server 2005 exibe as visões de sistema.

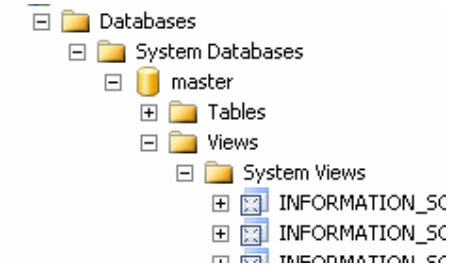

Figura 5.1. Parte da estrutura das visões de sistema do SQL Server

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).

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.

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 figura a seguir mostra as visões que compõem o esquema (grupo) *Information schema views*:

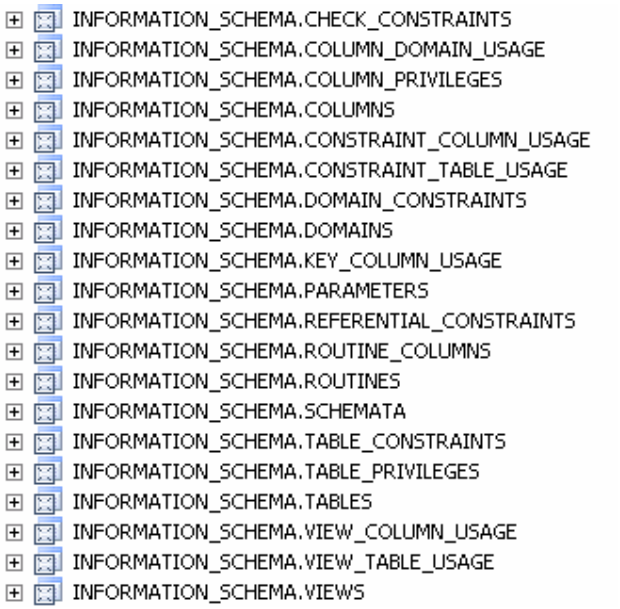

Figura 5.2. Parte da estrutura das visões do grupo Information Schema Views

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
```

Compatibility 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 (*features*) adicionadas pelo *SQL Server 2005*.

Catalog Views

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

A Microsoft recomenda o uso das visões de catálogo para acessar os metadados pelos seguintes motivos:

- Todos os metadados estão disponíveis (podem ser acessados) através das visões de catálogo.
- As visões de catálogo apresentam os metadados em um formato independente das tabelas de sistema. Além disso, estas visões não serão afetadas por futures mudanças na estrutura das tabelas de sistema.
- As visões de catálogo constituem a forma mais eficiente de acessar os metadados que compõem o núcleo do servidor.
- Os nomes das visões de catálogo, bem como de suas colunas, são autodescritivos..

5.2.3 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 os dados. Estas funções só foram implementadas a partir do SQL Server 7.0.

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.

5.2.4 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.

5.2.5 ODBC Catalog Functions

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

5.2.6 Mapeando as Tabelas de Sistema nas Visões de Sistema

A seguir iremos descrever como mapear as tabelas de sistema e as funções existentes no banco de dados *MASTER* do SQL Server 2000 nas visões de sistema e funções adicionadas no SQL Server 2005.

System table	System views or functions	Type of view or function
sysaltfiles	sys.master_files	Catalog view
syscacheobjects	sys.dm_exec_cached_plans sys.dm_exec_plan_attributes sys.dm_exec_sql_text sys.dm_exec_cached_plan_dependent_objects	Dynamic management view Dynamic management view Dynamic management view Dynamic management view
syscharsets	sys.syscharsets	Compatibility view
sysconfigures	sys.configurations	Catalog view
syscurconfigs	sys.configurations	Catalog view
sysdatabases	sys.databases	Catalog view
sysdevices	sys.backup_devices	Catalog view
syslanguages	sys.syslanguages	Compatibility view
syslockinfo	sys.dm_tran_locks	Dynamic management view
syslocks	sys.dm_tran_locks	Dynamic management view
syslogins	sys.server_principals sys.sql_logins	Catalog view
sysmessages	sys.messages	Catalog view
sysoledbusers	sys.linked_logins	Catalog view
sysopentapes	sys.dm_io_backup_tapes	Dynamic management view
sysperfinfo	sys.dm_os_performance_counters	Dynamic management view
sysprocesses	sys.dm_exec_connections sys.dm_exec_sessions sys.dm_exec_requests	Dynamic management view Dynamic management view

		Dynamic management view
sysremotelogins	sys.remote_logins	Catalog view
sysservers	sys.servers	Catalog view

Tabela 5.2. Mapeamento das tabelas de sistema e funções do banco de dados master do SQL Server 2000 para o SQL Server 2005

A seguir iremos descrever como mapear as tabelas de sistema e as funções existentes em um banco de dados qualquer (criado pelo usuário) do SQL Server 2000 nas visões de sistema e funções adicionadas no SQL Server 2005.

System table or function	System view or function	Type of view or function
fn_virtualfilestats	sys.dm_io_virtual_file_stats	Dynamic management view
syscolumns	sys.columns	Catalog view
syscomments	sys.sql_modules	Catalog view
sysconstraints	sys.check_constraints	Catalog view
	sys.default_constraints	Catalog view
	sys.key_constraints	Catalog view
	sys.foreign_keys	Catalog view
sysdepends	sys.sql_dependencies	Catalog view
sysfilegroups	sys.filegroups	Catalog view
sysfiles	sys.database_files	Catalog view
sysforeignkeys	sys.foreign_key_columns	Catalog view
sysindexes	sys.indexes	Catalog view
	sys.partitions	Catalog view
	sys.allocation_units	Catalog view
	sys.dm_db_partition_stats	Dynamic management view
sysindexkeys	sys.index_columns	Catalog view
sysmembers	sys.database_role_members	Catalog view

sysobjects	sys.objects	Catalog view
syspermissions	sys.database_permissions sys.server_permissions	Catalog view Catalog view
sysprotects	sys.database_permissions sys.server_permissions	Catalog view Catalog view
sysreferences	sys.foreign_keys	Catalog view
systypes	sys.types	Catalog view
sysusers	sys.database_principals	Catalog view
sysfulltextcatalogs	sys.fulltext_catalogs	Catalog view

Tabela 5.3. Mapeamento das tabelas de sistema e funções do banco de dados de usuário do SQL Server 2000 para o SQL Server 2005

5.3 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 sistemas, 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* (Figura 5.4), 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 no Quadro 5.1 , a obtém os últimos comandos SQL executados pelo SQL Server, juntamente com a quantidade de vezes que o comando foi executado.

Coluna	Tipo de Dados	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 5.4. – Principais campos da tabela de sistema Syscacheobjects.

```
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)
```

Quadro 5.1. – Recuperando as últimas cláusulas SQL submetidas ao servidor.

Para obter o plano de execução de uma determinada cláusula SQL capturada usamos o comando "*set showplan_all on"* seguido da cláusula *SQL* capturada". A seguir descreveremos o funcionamento deste comando.

O SQL Server provê vários comandos distintos para se obter o plano de execução de uma determina cláusula SQL. A tabela 5.5 descreve, de forma sucinta, estes comandos:

	Comando	Executa a Consulta?	Exibe o plano em modo texto?	Exibe a quantidade estimada de linhas (tuplas)?	Exibe o número de linhas corrente ?
Texto	set showplan_text on	Não	Sim	Não	Não
	set showplan_all on	Não	Sim	Sim	Não
	set statistics profile on	Sim	Sim	Sim	Sim
XML	set showplan_xml on	Não	Sim	Sim	Não
	set statistics xml on	Sim	Sim	Sim	Sim

Tabela 5.5 – Recuperando os planos de execução das últimas cláusulas SQL submetidas ao servidor.

O Quadro 5.2 mostra como podemos 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
```

Quadro 5.2 – Obtendo o Plano de Execução

Para se obter a carga de trabalho submetida ao SQL Server em formato XML podemos utilizar a visão *sys.dm_exec_requests* em conjunto com as funções *sys.dm_exec_sql_text* e *sys.dm_exec_query_plan*. A tabela 5.6 mostra os principais campos da visão *sys.dm_exec_requests*.

Nome da Coluna	Tipo de Dados	Descrição
session_id	**smallint**	ID da seção associada à requisição.
request_id	**int**	ID da requisição. Valor único no contexto da seção.
start_time	**datetime**	*Timestamp* indicando o momento em que a requisição chegou ao servidor.
status	**nvarchar(60)**	Status da requisição. Pode ser um dos seguintes valores: Background Running Runnable Sleeping Suspended
command	**nvarchar(32)**	Identifica o tipo do comando que está sendo processado. Os tipos mais comuns incluem: SELECT INSERT UPDATE DELETE BACKUP LOG BACKUP DB DBCC WAITFOR LOCK MONITOR CHECKPOINT LAZY WRITER
sql_handle	**varbinary(64)**	*Hash map* do texto da clausula SQL da requisição.
plan_handle	**varbinary(64)**	*Hash map* do plano de execução
user_id	**int**	ID do usuário que submeteu a requisição.
transaction_id	**bigint**	ID da transação à qual a requisição pertence.

cpu_time	**int**	Tempo de CPU em milissegundos.
reads	**bigint**	Número de leituras executadas pela requisição.
writes	**bigint**	Número de escritas executadas pela requisição.
transaction_isolation_level	**smallint**	Nível de isolamento utilizado pela transação à qual a requisição pertence.
lock_timeout	**int**	Período de *Lock time-out* em milissegundos definido para a requisição.
row_count	**bigint**	Nuúmero de linhas retornadas pela requisição.

Tabela 5.6. – Comandos utilizados para recuperar os planos de execução das consultas anteriormente executadas.

O quadro 5.3 mostra como podemos obter a consulta e o plano de execução referentes à uma determinada requisição, em formato XML, no SQL Server 2005.

```
DECLARE @sessionid INT
SET @sessionid = 52 -- Change to the session ID script 2 was executed in

DECLARE @sqlhandle VARBINARY(64)
DECLARE @planhandle VARBINARY(64)

SELECT @sqlhandle = sql_handle, @planhandle = plan_handle
FROM sys.dm_exec_requests
WHERE session_id = @sessionid


SELECT *
FROM sys.dm_exec_sql_text(@sqlhandle)

SELECT *
FROM sys.dm_exec_query_plan(@planhandle)
```

Quadro 5.3 – Obtendo (em XML) as cláusulas SQL e seus respectivos planos de execução.

5.4 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 descrição dessas tabelas e de seus atributos mais importantes são mostrados a seguir:

A tabela *sysobjects* cataloga as informações sobre os objetos do SGBD (tabelas, visões, procedimentos armazenado, etc). A descrição de seus principais campos, pode ser vista na Tabela 5.7.

A tabela de sistema *sysindexes* (Tabela 5.8) 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.

Para se obter informações sobre uma determina tabela, como por exemplo, nome da tabela, número de linhas, número de blocos, método de acesso do índice e os nomes dos índices existentes em uma tabela definimos várias cláusulas SQL. Estas cláusulas estão descrita nos quadros 5.4, 5.5 e 5.6.

Nome da Coluna	Tipo de Dados	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 5.7. – Parte da estrutura da tabela de sistema Sysobjects.

Nome da Coluna	Tipo de Dados	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 re-criar 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 5.8. Principais campos da tabela Sysindexes

```
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
```

Quadro 5.4 - Cláusula que obtém quantidade de *tuplas* de uma tabela .

```
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
```

Quadro 5.5 - Cláusula que obtém quantidade de blocos "paginas" de uma tabela .

```
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
```

Quadro 5.6- Cláusula que obtém os índices de uma tabela.

6 Conclusões

O desempenho dos servidores de bancos de dados é fator chave para o sucesso das aplicações de missão-crítica. Para estas aplicações, uma baixa performance significa perdas de receita e de oportunidades de negócio. A fim de assegurar uma performance 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 a performance 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 performance 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 (clausulas SQL, planos de execução, custos, etc.), bem como informações estatísticas (número estimado de linhas e de páginas de uma tabela, índices existentes, etc.), torna-se de fundamental importância.

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

Neste trabalho realizamos um estudo exaustivo dos metadados dos principais SGBDs comerciais: Postgres, 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 performance.

6 Bibliografia

[1] MONTEIRO, J. M.: *An Architecture for Automated Index Tuning*. V Workshop de Teses e Dissertações em Banco de Dados (WTBD), realizado em conjunto com o XXI Simpósio Brasileiro d Banco de Dados (SBBD). 2006.

[2] CHAUDHURI, S., DATAR, M., AND NARASAYYA, V.: *Index Selection for Databases: A Hardness Study and a Principled Heuristic Solution*. IEEE Transactions on Knowledge and Data Engineering, 16 (11):1313–1323, 2004.

[3] COSTA, R. L. C., LIFSCHITZ, S., NORONHA, M., SALLES, M. V.: *Implementation of an Agent Architecture for Automated Index Tuning*. ICDE Workshops 2005.

[4] SALLES, M. V.: Criação Autônoma de Índices em Bancos de Dados. Dissertação de Mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUCRio), 2004.

[5] MORELLI, E. M. T.: Recriação Automática de Índices em um SGBD Relacional. Dissertação de Mestrado, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUCRio), 2006.

[6] LIFSCHITZ, S.; MILANÉS, A. Y. ; SALLES, M. A. V.. Estado da arte em auto-sintonia de sistemas de bancos de dados relacionais. Relatório técnico, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), 2004.

[7] WEIKUM, G.; M¨ONKEBERG, A.; HASSE, C. ; ZABBACK, P.. Self-tuning database technology and information services: from wishful thinking to viable engineering. In: Proceedings of the International Conference on Very Large Databases (VLDB), p. 20–31, 2002.

[8] VIEIRA, M.; DURÃES, J.; MADEIRA, H. Especificação e Validação de Benchmarks de Confiabilidade para Sistemas Transacionais. IEEE Latin America Transactions, Jun. 2005.

[9] KENDALL, E.A. KRISHNA, P.V.M. PATHAK, C.V. AND SURESH, C.B., *An Application Framework for Intelligent and Mobile Agent Systems*, capítulo em Implementing Applications Frameworks: Object Oriented Frameworks, Work, ed. M. Fayad, D. C. Schmidt, R. Johnson, Wiley & Sons, 1999.

[10] SALLES, M. V., AND LIFSCHITZ, S.: Autonomic Index Management. In Proceedings of the International Conference on Autonomic Computing (ICAC), 2005.

[11] CHAUDHURI, S., AND NARASAYYA, V.: Autoadmin “what-if” Index Analysis Utility. In Proceedings of the ACM SIGMOD International Conference on Management of Data, pages 367–377, 1998.

[12] CHAUDHURI, S., AND WEIKUM, G.: Rethinking Database System Architecture: Towards a Self-tuning Risc-Style Database System. In Proceedings of the International Conference on Very Large Databases (VLDB), pages 1–10, 2000.

[13] COSTA, R. L. C., LIFSCHITZ, S. AND SALLES, M. V.: Index Self-Tuning with Agent-Based Databases. CLEI Electronic Journal, 6(1):22 PAGES, 2003.

[14] TPC: “TPC Benchmark – Revision 2.1.0”. Disponível em: < http://www.tpc.org>. Acesso em 05 de junho de 2007.

[15] MOMJIAM, B.: Postgresql – Introduction and Concepts. Addison-Wesley. New York, 2001.

[16] STINSON, B.: Postgresql Essential Reference. New Riders Publishing, Primeira Edição, 2001.

[17] GESCHWINDE, E. AND JUNGERSCHONING, H.: Postgresql Developer’s Handbook. Sams Publishing, Segunda Edição, 2002.

[18] O Coletor de Estatísticas do Postgresql. Disponível em: <http://www.javalinux.com.br/javalinux/pg74/monitoring-stats.html>. Visitado em 01/07/2007.

[19] ORACLE V$ View List. Disponível em: <http://www.dba-oracle.com/menu_v$_views_list.htm>. Acesso em 05 de junho de 2007.

[20] **SQL Server 2005 Books On-Line**. Disponível em: <http://www.microsoft.com/>. Acesso em: 11 jun. 2007.

[21] NAVATH E, S.; ELMASRI, R. **Sistemas de banco de dados**, 4. ed. São Paulo: Addison Wesley, 2005.

[22] SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistema de banco de dados.** 5. ed. São Paulo: Campus, 2006.