FUNCIONAMENTO DO BANCO DE DADOS RELACIONAL *SQL SERVER 2000 ENTERPRISE*

Alex Lima dos Santos, Anderson David Xavier, Humberto Quintino Salomão e José Ricardo Carvalho Moraes

Universidade Municipal de São Caetano do Sul
Rua Santo Antônio, 50 - São Caetano do Sul - SP - Brasil - CEP 09550-001

lima_santos@hotmail.com
andier3@gmail.com
humbertoqs@gmail.com
ricardomoraes1977@gmail.com

RESUMO

Este artigo descreve as principais características do banco de dados *SQL Server 2000*, e tem por objetivo explicar, de forma clara, o seu funcionamento, sem realizar nenhum tipo de comparação com outros SGBDs (Sistema Gerenciamento de Banco de Dados). As descrições foram realizadas com base nos principais pontos que afetam o funcionamento de um SGBD.

Palavras-chave: *SQL Server 2000,* SGBD.

ABSTRACT

This article describes main characteristics of SQL Server 2000 data base and it has as objective explaining clearly its functioning, without achieving any kind of comparison with others DBMS (Data Base Management System). Descriptions were done basing on main points that affect functioning of a DBMS.

Keywords: SQL SERVER 2000, DBMS

1. INTRODUÇÃO

Diante das diversas versões do *SQL Server*, foi escolhida a versão *2000 Enterprise* para plataforma *Windows 2000 Server*, pois ela é a versão completa deste SGBD, com o objetivo de descrever o seu funcionamento.

O *SQL Server* é um gerenciador de banco de dados relacional da *Microsoft*, e possui diversas versões, cada qual para um sistema operacional *Windows*, da própria *Microsoft*, ao contrário de outras empresas, que dispõem seus SGBDs para diversas plataformas e sistemas operacionais (MORELLI, 2001).

A Tabela 1 exibe todas as versões e seus respectivos sistemas operacionais compatíveis (RIBEIRO, s/d).

Suas ferramentas de gerenciamento são distribuídas junto com o sistema de SGBD, não necessitando ser adquiridas separadamente. Abaixo, encontram-se todas as ferramentas que são instaladas com o *SQL Server*.

- *Client Network Utility* – realiza configurações do *client SQL* para conexão em outros servidores; define qual protocolo de conexão será utilizado para acessar um servidor externo, definição, aliás, para outros servidores;
- *Configure SQL XML Support in IIS* – utilizado para configurar um diretório virtual apropriado, a fim de que seja possível acesso ao servidor de banco de dados através da Internet, por meio de http;
- *Enterprise Manager* – console central que tem todas as ferramentas necessárias para gerenciar o banco de dados;
- *Import and Export Data* – utilizado para importar e exportar dados entre bancos de dados, ou de arquivos para o banco de dados e vice-versa;
- *Profiler* – utilizado para monitorar o servidor, consegue realizar filtros e capturar as atividades no servidor que está conectado, monitorando todas as ações realizadas, e é baseado em filtros específicos;
- *Query Analyzer* – utilizado para executar *scripts SQL*, monitorar o servidor por meio de *scripts*, funções definidas pelo fabricante ou, até mesmo, pelo usuário; usado para analisar e gerenciar várias consultas em diferentes janelas ao mesmo tempo;
- *Server Network Utility* – realiza configurações do *Server SQL*, definindo qual a forma de conexão que os usuários deverão realizar para acessar o servidor; define qual protocolo de conexão será utilizado para acessar o servidor;
- *Service Manager* – utilitário gráfico utilizado para iniciar, pausar e parar os serviços do *SQL Server*.

Antes de ter sua instalação realizada, existem requisitos que devem ser atendidos, para sua realização. O *SQL Server 2000 Enterprise*, para que seu funcionamento seja eficiente, tem como primeiro ponto a se analisar o *hardware* em que será instalado o SGBD. Para a versão analisada em questão, os requisitos que lhe atendem são o sistema operacional *Windows Server*, com processador *Intel* ou compatível, o processador com *clock* de 166MHz ou superior, com, no mínimo,

Tabela 1: Versões do *SQL Server* e sistemas compatíveis

SQL Server 2000 Edition	Windows 2000 & Windows NT Server	Windows 2000 Professional & Windows NT Workstation	Windows 98 & Windows Millennium	Windows CE
Enterprise	Yes	No	No	No
Enterprise Evaluation	Yes	No	No	No
Standard	Yes	No	No	No
Developer	Yes	Yes	No	No
Personal	Yes	Yes	Yes	No
Windows CE	No	No	No	Yes

64MB de memória RAM (dependendo do *Windows Server* escolhido – no caso, o *Windows 2000 Server* necessita de, no mínimo, 256MB de memória) e 250MB de espaço em disco para a instalação típica (MICROSOFT, 2000).

Após realizarem-se todos esses passos, tem-se um sistema gerenciador de banco de dados, utilizado para gerenciar bancos de dados OLTP (*On Line Transaction Processing*) e Olap (*On Line Analytical Processing*), totalmente integrado com o sistema operacional *Windows*, onde é possível distribuir e escalonar o *SQL Server* por vários servidores, utilizando-se de serviços do *Windows 2000 Server*.

Pode-se ter, no mesmo computador, múltiplas instalações do *SQL Server*, cada qual funcionando como um servidor separado, possuindo individualmente seu próprio conjunto de sistemas e bancos de dados de usuários, que não possuem integração entre as outras ocorrências instaladas. Cada ocorrência possuirá tipos de banco de dados padrão e de usuários, sendo que cada instalação cria, no mínimo, dois arquivos de banco de dados: um deles para dados e outro para controle, que é chamado de arquivo de log. A instalação padrão disponibiliza alguns bancos de dados, sendo que quatro deles são bancos de sistema e dois são bancos de exemplos, podendo estes ser excluídos sem causar problemas maiores ao sistema (RAMALHO, 2001).

Abaixo, segue uma lista com todos os bancos instalados por padrão (ROMÃO, 2006):

- *master* – mantém todas as informações do servidor como um todo, como contas de usuários, objetos, mensagens de erro etc.;
- *model* – modelo de banco, todos os bancos que são criados têm como modelo o banco de sistema ***model***, sendo que qualquer alteração realizada nele afetará os próximos bancos de dados de usuários;
- tempdb – fornece uma área que é utilizada para todo tipo de armazenamento temporário, tal como criação de tabelas temporárias;
- msdb – é o local que tem armazenadas todas as informações de agendamento de serviços e histórico;
- *pubs* e *Northwind* – bancos de exemplo, utilizados para aprendizagem.

Existe um banco de sistema que não é instalado por padrão, chamado ***distribution,*** que só aparece quando o *SQL Server* é configurado para realizar duplicação de dados. Neste banco, ficam os dados de histórico e transações que foram usadas na duplicação.

Dentro de cada banco de dados, há tabelas que contêm uma definição de todos os objetos daquele respectivo banco de dados e permissões. A este conjunto de tabelas dá-se o nome de **catálogo de banco de dados**. Já no banco *master* tem-se um outro conjunto de tabelas de sistema, que possui informações do sistema como um todo; a esse outro conjunto de tabelas, especificamente, é dado o nome de **catálogo de sistema**. A recuperação de dados dessas tabelas pode ser realizada por *stored procedures* de sistema, já definida pelo fabricante, que possibilita recuperar informações de um determinado objeto ou do sistema como um todo. Pode-se, ainda, utilizar *views* de informações ou, até mesmo, realizar uma consulta direta às tabelas, desde que existam permissões para isso.

A segurança implementada no servidor é realizada em dois níveis, sendo o primeiro deles autenticação de *login*, e o segundo, permissão de usuário e cargo de banco de dados. A autenticação de *login* pode ser efetuada de dois modos, sendo ela *Windows authentication* ou *SQL authentication*, com a principal diferença evidenciando-se em que o modo *Windows authentication* só permite conexão por meio do *Windows 2000*. Sendo assim, o usuário não pode especificar uma conta de *login* para se conectar ao *SQL Server*.

Depois de conectado ao servidor, o usuário deve possuir uma conta de banco de dados, conta esta que controla a propriedade sobre objetos e permissões para executar determinadas instruções. Tais contas podem ser atribuídas a cargos fixos de servidor, cargos fixos de banco de dados ou cargos definidos pelo próprio usuário.

O monitoramento de tentativas de acesso pode ser realizado por alertas configurados no servidor, de modo que a administração seja facilitada. Além disso, é possível realizar uma série de alertas de desempenho e

agendamentos que permitem que determinadas instruções sejam executadas automaticamente. Todas essas informações, como já foi dito anteriormente, ficam armazenadas no banco de sistema **msdb**.

2. PROCESSANDO CONSULTAS

Toda atividade que permite serem extraídos dados do banco de dados é, na verdade, um processamento de consulta, e toda essa atividade inclui traduções de consultas em expressões que possam ser implementadas em nível físico do sistema de arquivos, sendo que o custo para todo esse processo de consulta é determinado pelo acesso ao disco (SILBERSCHATZ, 1999).

Como o banco analisa e interpreta uma instrução *SQL*

Dentro do *SQL Server*, existe um componente de servidor chamado *Relational Engine*, que realiza o *parse* da consulta, a otimização, os planos de execução e o processamento dos comandos DDL; sendo assim, ele reforça a segurança (MICROSOFT, 2001).

Toda a análise, quando uma consulta é solicitada ao banco de dados, ocorre da maneira a seguir explicitada.

O *Relational Engine* compila a consulta em um plano de execução otimizado. Depois de compilado, esse plano de execução vai ao encontro do conjunto de dados solicitados na consulta. Estes dados são transferidos para ele por meio de outro componente, chamado *Storage Engine*, que tem por função, nesse caso, transferir os dados do *buffer* para o *relational engine* (PASTORELO, 2005).

O *Relational Engine* tem como principais responsabilidades:

- analisar as principais solicitações *SQL* e quebrá-las para unidades lógicas menores, aperfeiçoar os planos de execução, estimando o custo BDE de cada série solicitada primeiramente em termos de I/O, assim escolhendo as etapas de menor custo e, então, combinando para que seja possível um plano otimizado;
- executar toda a série de operações lógicas definidas pelo plano de execução na seqüência recebida;
- processar DDLs e outros comandos;
- formatar os resultados e retorná-los para o **cliente** solicitante. Esse formato pode ser tradicional, tabular ou um documento XML.

3. CONTROLES DE TRANSAÇÕES E CONCORRÊNCIA

3.1. Transação

Uma transação é um conjunto de instruções *SQL* que são validadas juntas em um instante de tempo. Se alguma instrução possuir algum erro na sua execução, todas as outras instruções que foram feitas devem ser desfeitas também.

O *SQL Server* utiliza dois tipos de transação: a explícita (utiliza os comandos *begin transaction*, indicando o início de uma transação, e o *commit/roolback transaction*, onde o primeiro indica que as instruções da transação foram salvas sem nenhum erro, e o segundo indica que as instruções da transação foram desfeitas por algum erro) e a implícita, que não necessita dos comandos *begin*, *commit* e *roolback*.

Alguns comandos, como *select into, insert, select, update, delete, truncate table, begin transaction* ou *begin tran, commit transaction* ou *commit, rollback transaction* ou *roolback* e *save transaction*, representam uma transação.

No *SQL Server*, dentro de cada banco de dados criado, existe um arquivo chamado *Transaction Log* (TL), que contém todas as transações executadas até o último momento, podendo ser utilizado para recuperar transações quando houver perda de dados.

Duas coisas devem ser evitadas em transações aninhadas muito longas: uma é a complexidade de execução da lógica de programação, pois, algumas vezes, não fica claro tudo o que pode acontecer (por exemplo, algum erro, alguma mudança não prevista nos dados); outra é que transações muito longas fazem com que as tabelas que estão sendo utilizadas fiquem presas com *locks*, impedindo que outras transações utilizem essas tabelas por, talvez, **um longo período de tempo.**

3.2. *Lock*

Lock é um mecanismo que faz com que as linhas de uma tabela utilizadas para execução de uma transação fiquem bloqueadas para acesso de outras transações, até que a transação que as utiliza libere seu acesso, porque foi executada com sucesso ou não, assegurando, assim, que duas ou mais transações não alterem a mesma linha de uma tabela no mesmo instante que é executada. Isto permite que, quando uma transação finalizar, a outra possa modificar os dados feitos ou alterados pela anterior.

Quando um comando *COMMIT* é executado numa transação, este sinaliza que as instruções foram executadas com sucesso, registra no *Transaction Log* a transação que foi feita e desfaz o *lock* efetuado nas tabelas utilizadas. No *ROLLBACK*, todas as instruções da transação são desfeitas e se desfaz o *lock* feito nas tabelas utilizadas.

O *SQL Server* utiliza alguns tipos de *lock*s nas tabelas, mas normalmente usa o *share lock*, quando executa uma operação de leitura em uma tabela, e, também, o *exclusive lock*, utilizado para comandos de escrita (*INSERT*, *DELETE*, *UPDATE*). Na Tabela 2 abaixo, são listados os tipos de item em que o *SQL Server* pode efetuar um *lock*.

Se o desejado é poder utilizar uma tabela, mesmo que ela esteja presa por um *lock*, o *SQL Server* permite utilizar *loching hints*, que são mecanismos utilizados com essa finalidade. Vejam-se alguns na Tabela 3.

3.3. Uso de protocolo de bloqueio no *SQL Server*

O *SQL Server* utiliza um protocolo de controle de bloqueio, que permite definir o grau de isolamento de uma transação em relação à outra, definindo como o bloqueio dos dados utilizados será feito. Tudo isso o próprio usuário pode definir.

Sintaxe:

```
SET TRANSACTION ISOLATION LEVEL { SERIALIZABLE
    | REPEATABLE READ
    | READ COMMITTED
    | READ UNCOMMITTED}
```

SERIALIZABLE – bloqueia os dados até o final da transação, impedindo que novas linhas sejam inseridas nas tabelas em uso.

REPEATABLE READ – bloqueia os dados utilizados até o final da transação, mas permite que novas linhas sejam inseridas nas tabelas em uso.

READ COMMITTED – durante a transação, se, num momento, o dado que a transação utiliza não estiver sendo utilizado, este dado poderá ser modificado.

READ UNCOMMITTED – não usa bloqueio, podendo gerar durante a transação dados inconsistentes.

Tabela 2: Tabela com todos os itens que o *SQL Server* pode efetuar um *lock*

Item	Observações
Row	Uma coluna de uma tabela identificada por um RID (*row identifier*).
Page	Página com 8.060 *bytes*; uma linha não ocupa duas páginas.
Extent	Grupos de oito páginas de dados ou índices.
Table	Todas as páginas de dados e índices de uma tabela.
Database	*Lock* em todo o banco.

Tabela 3: Tabela com alguns tipos de *Locking Hint* que o *SQL Server* utiliza

Locking Hint	Observações
Nolock	Burla os *lock*s de outras transações.
Holdlock	Permite a leitura, mas impede a atualização até o final da transação.
Tablock	Impede leitura ou atualização até o final de um comando.
Tablockx	Impede leitura ou atualização até o final da transação.

O *SQL Server* permite ao usuário definir um tempo de espera para que um dado que esteja bloqueado por uma transação fique livre. O próprio usuário pode definir este tempo através do seguinte comando: SET *LOCK*_TIMEOUT 1000 (neste comando, qualquer transação que bloquear um dado por mais de um segundo será abortada). No exemplo fornecido, foi utilizado um segundo, mas o tempo de espera é de escolha do usuário. O valor atual do tempo de espera pode ser visto com o comando SELECT@@*LOCK*_TIMEOUT.

3.3.1. Detecção e recuperação de deadlocks

Mesmo utilizando o protocolo de bloqueio, o *SQL Server* não impede que *deadlocks* aconteçam. Para acabar com um *deadlock*, caso este aconteça, o *SQL Server* permite definir um grau de prioridade para as transações que fazem parte deste *deadlock*, podendo abortar a transação que possua a menor prioridade, mas esse critério sobre a prioridade também pode ser modificado pelo usuário.

4. CATÁLOGOS DO SISTEMA OU DICIONÁRIO DE DADOS

O catálogo do sistema é um conjunto de tabelas utilizadas para operar e gerenciar o sistema gerenciador de banco de dados, neste caso o *SQL Server*. Estas tabelas são, conforme Ferreira (2004), armazenadas no banco de dados *Master* (controla os bancos de dados do usuário e a operação do *SQL Server*. Tem como tamanho inicial 16MB. É importante manter um *back-up* atualizado desse banco de dados, pois ele contém informações sobre contas de *login*, processos em execução, mensagens de erro, bancos de dados criados no servidor, espaço alocado para cada banco de dados, travas [*lock*s] de linhas ativas, espaço alocado para cada banco de dados e procedimentos armazenados do sistema), que é um banco gerado automaticamente pelo *SQL Server*, quando instalado em um computador.

4.1. Tabelas do catálogo do sistema

Algumas tabelas do catálogo do sistema e sua finalidade encontram-se listadas a seguir:

- *syscharsets* – códigos de página que estabelecem quais caracteres estão disponíveis e sua ordem de classificação;
- *sysconfigures* ou *syscurconfigs* – parâmetros de configuração do *SQL Server*;
- *sysdatabases* – informações sobre os bancos de dados existentes;
- *sysdevices* – informações sobre os dispositivos utilizados pelo banco de dados, como o dispositivo de fita;
- *syslanguages* – idiomas suportados pelo servidor;
- *syslocks* – informações sobre travas ou *lock*s existentes no momento em que as transações que estão sendo executadas;
- *syslogins* – exibição das contas de usuários para acesso ao banco de dados;
- *sysmessages* – exibição das mensagens de erro mostradas pelo sistema, caso haja;
- *sysprocesses* – exibição dos processos em execução no momento;
- *sysremotelogins* – exibição das contas de acesso remoto para o banco de dados;
- *sysservers* – exibição dos servidores remotos do banco de dados;
- *sysusages* – exibição do espaço em disco disponibilizado para cada banco de dados.

5. SEGURANÇA

Em termos de segurança do *SQL Server 2000*, há uma série de fatores que influenciam sua utilização, dependendo de qual sistema operacional é utilizado. Por exemplo, usando um sistema operacional inferior, como o *Windows 98*, o mesmo não possui segurança em nível de arquivo, assim comprometendo os arquivos que compõem o banco de dados do *SQL*, que permanecem desprotegidos no sistema. Já nos sistemas a partir do *Windows 2000*, há uma confiança na segurança em relação à proteção dos dados (SHAPIRO, 2002).

A maneira de obter acesso ao *SQL Server* é um *login*, seja o mesmo interno do *SQL Server* ou um *login* que pode ser mapeado através de um usuário ou grupo do *Windows*. Após o *login*, é efetuado um mapeamento para um usuário do banco de dados e para cada banco

de dados que este determinado *login* estará habilitado a possuir permissão.

O processo todo de verificação de cargo de banco de dados funciona da seguinte maneira: depois de conectado ao servidor, o usuário envia ao banco de dados uma instrução, como uma solicitação de visualização de um determinado objeto do banco de dados em especifico ou, até mesmo, uma instrução *SQL*. Quando o *SQL Server* recebe essa instrução, ele primeiro verifica as permissões desse usuário para tal solicitação; em seguida, pode ocorrer uma resposta positiva, e o *SQL Server* executará ou indicará um erro, caso o usuário não tenha as permissões necessárias.

O *SQL Server* permite dividir seu plano de segurança e suas permissões em dois segmentos: o primeiro, com a segurança do SGBD, e a outra, no banco de dados e seus objetos. Este plano facilita o gerenciamento do banco de dados, levando em consideração as prioridades ao efetuar a divisão do banco (SILBERSCHATZ, 1999).

Em relação à segurança dos objetos e dados, o *SQL Server 2000* oferece um modelo de segurança bastante flexível, onde pode ser administrado um único acesso por *login* a um ou mais banco de dados, e se podem restringir acessos a tabelas, visualização e outros objetos dentro dos tais bancos de dados, tornando seu controle simples (SHAPIRO, 2002).

No *SQL Server*, é denominado como *role* um grupo de usuários de banco de dados, pelo qual existem dois tipos de grupos (PICHILIANI, 2006), a seguir explicitados.

Fixed Server Roles: neste *role*, são agrupados lajens, que, por sua vez, trabalham com permissões que dizem respeito ao gerenciamento do servidor. Existe uma limitação pela qual não é possível criar *Server Roles*, sendo possível apenas incluir lajens nos feche *Server Roles* predeterminados.

Fixed Database Roles: neste tipo de *role*, colocam-se os usuários de bancos de dados agrupados para conceder-lhes os privilégios referentes aos objetos do banco de dados. Desta forma, podem-se colocar os usuários de bancos de dados dentro de um feche *database role*, sendo possível também criar os próprios *roles* e aos tais atribuir permissões, ao invés de trabalhar concedendo ou negando permissões para usuários de banco de dados individualmente.

De acordo com Shapiro (2002), podem-se classificar os dados com os seguintes níveis: confidencial, restrito, secreto e altamente secreto, considerando como **confidenciais todos** os dados que estão disponíveis a todos os usuários em um servidor de uma determinada empresa. Acesso **restrito** é adotado para arquivos que serão disponibilizados a um certo grupo de pessoas – muitas vezes são dados temporários que irão ser ou não efetivados no futuro. As informações com o nível **secreto** são dados acessados apenas por pessoas que têm realmente a necessidade do conhecimento ou de quem tenha sido provada sua confiança para o acesso aos dados. Por fim, os dados **altamente secretos** são considerados como de máxima segurança de uma empresa, pois a divulgação de suas informações poderia ser considerada desastrosa a membros interligados à mesma (por exemplo, seriam cartões de crédito da empresa, contas de banco, assinaturas digitais, planos de patentes etc.) (SILBERSCHATZ, 1999).

Com o sistema *client/server*, o *SQL* tem um recurso que suporta a interação de vários usuários interligados em uma rede, porém existem riscos ao efetuar a comunicação desses dados na rede, pois, caso uma pessoa venha a ter acesso à visualização dos dados que trafegam na rede, por intermédio de um analisador de pacote ou uma espécie de rastreador, será possível visualizar os dados sem nenhuma dificuldade. Neste caso, a melhor solução é utilizar um meio de codificação de dados. Por exemplo, no *Windows 2000* podem-se citar algumas opções de codificação, como VPNS (*Virtual Private Networks*) e IPSec (*Internet Protocol Security*) (SHAPIRO, 2002).

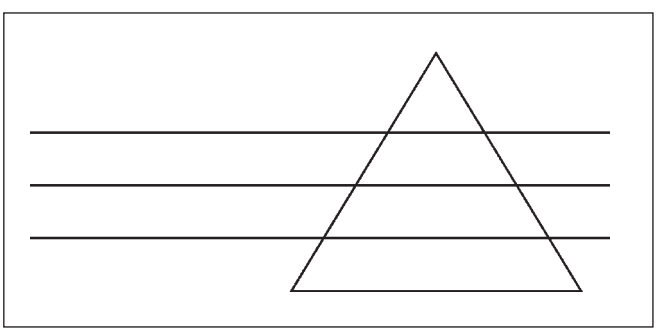

Figura 1: Representação dos níveis classificação

6. OTIMIZAÇÃO

A otimização apresenta um aspecto muito importante que se dá ao desempenho quanto a um ambiente de grande porte, onde é extremamente necessária a otimização.

Uma das técnicas mais conhecidas para melhorar o desempenho de uma consulta é a utilização de índices, com o intuito de melhorar a velocidade da recuperação de dados em tabelas complexas, pois, na maioria dos casos, é muito mais fácil efetuar uma busca direta a um determinado dado ou informação, ao invés de ter que efetuar uma varredura na tabela, ou seja, uma verificação item por item de uma lista, porém é o otimizador que decide o que fazer, baseado em uma estimativa sobre a utilização ou não de um índice. Essa estimativa é baseada em informações estatísticas sobre o índice, que dizem qual a distribuição dos dados.

Caso uma tabela seja pequena, a utilização de índices é descartada pelo otimizador, assim utilizando o método de varredura. Se na cláusula *WHERE* da consulta estão presentes colunas indexadas, é bem provável que o otimizador resolva utilizar o índice.

Além das alternativas de índices e varredura das tabelas, é possível analisar quais consultas possuem um baixo desempenho e, assim, serem elas revisadas pelos desenvolvedores *SQL* para um melhoramento das mesmas.

A otimização nada mais é do que uma função fundamental do servidor do banco de dados, que, por sua vez, tem a funcionalidade de agilizar os meios de acesso aos dados, sempre obtendo um desempenho aceitável e a possibilidade de evoluir, pois, conforme as instruções *SQL* são amplamente melhoradas, os servidores *SQL* tendem a obter um desempenho ainda melhor em um nível superior.

7. ARQUITETURA DE ARMAZENAMENTO FÍSICO

Os bancos de dados no *SQL Server 2000* são armazenados em grupos de arquivos que contêm dados individuais e de log. Cada banco de dados é composto de arquivos de dados principais e, opcionalmente, de secundários, assim como arquivos de log (SILBERSCHATZ, 1999).

Os arquivos de dados são facilmente identificados pelas suas extensões: principal (*.mdf) e secundários (*.ndf), além dos arquivos de log, (*.ldf). Estes arquivos de dados e logs podem ser armazenados em qualquer unidade não-comprimida FAT e NTFS. Com exceção dos arquivos de logs, os dados nos arquivos do *SQL Server* são gravados em páginas.

Cada página armazena 8K de dados, permitindo um máximo de 8.060 *bytes* em cada página, sendo que 96 *bytes* são usados para cada cabeçalho de página. Devido ao tamanho da página, os registros também têm tamanho máximo de 8.060 *bytes* para todos os registros, exceto para texto e dados de imagem, que são gravados em uma coleção de páginas separadas, as quais permitem até 2GB de informações por item de dados, e a localização destas páginas de dados é gravada com o registro-fonte.

Estas páginas são combinadas em grupos de oito e alocadas em tabelas e índices no *SQL Server*, recebendo o nome de extensão. Caso não seja necessário utilizar oito tabelas de 8K de armazenamento, o *SQL Server* direciona o problema a duas diferentes classes de extensão: extensões informes, que são completamente alocadas em um determinado objeto, e extensões mistas, que permitem que o compartilhamento de uma única extensão de oito páginas seja dividida entre oito objetos no total.

7.1. Grupos de arquivos

Cada banco criado contém um grupo de arquivo principal que, por padrão, contém todos os arquivos de dados. Estes grupos de arquivos são um meio de categorizar ou agrupar arquivos de dados em unidades, que podem ser distribuídas pelos vários discos físicos e administrados de forma individual, permitindo que tabelas e índices específicos, assim como o texto, *ntext* e dados de imagem, sejam acessados para um grupo de arquivos específicos (SILBERSCHATZ, 1999).

O banco *SQL Server 2000* é limitado a 256 grupos de arquivos. Cada grupo de arquivo, entretanto, pode conter até 32.767 arquivos de dados; logo, a habilidade de segmentar o banco de dados por vários discos é enorme.

7.2. Arquivos de dados

Há dois tipos de dados no *SQL Server*: principal e secundário. Ao criar-se um novo banco de dados, um arquivo de dados principal simples é criado para armazenar todos os seus dados, e fica logicamente localizado no grupo de arquivo PRINCIPAL, que é criado também automaticamente e definido como grupo de arquivo padrão para o banco de dados. Os novos objetos criados são alocados para um grupo de arquivo PRINCIPAL, por padrão, se não for especificado previamente por alguém. Os arquivos de dados podem pertencer somente a um grupo de arquivo (SILBERSCHATZ, 1999).

Os arquivos de dados principais são gravados, por padrão, com a extensão *.mdf; já os secundários são gravados com a extensão *.ndf.

7.3. Logs de transação

Todos os bancos de dados contêm, ao menos, um arquivo de dados e um de log. No arquivo de log, é encontrado o arquivo log de transação que é usado para registrar as informações que descrevem as alterações aplicadas aos dados do banco de dados. Os registros de log contêm o início de cada transação, o estado de retorno ou execução da transação, os dados que foram alterados e as informações necessárias para desfazer a transação, como valores anteriores de coluna. Eventos adicionais, como alocação de página, e eventos de índice também são registrados no log de transação, permitindo que se retorne o banco de dados a um ponto específico e garanta integridade aos dados, encaminhando cada transação na sua seqüência original.

Os logs de transação não contêm páginas como arquivos de dados; eles gravam informações como registro de log, sendo armazenados fisicamente com a extensão *.ldf.

8. TÉCNICAS DE RECUPERAÇÃO

O *SQL Sever* possui alguns mecanismos para recuperação de dados, explicitados nos itens seguintes.

8.1. *Back-up*

O *back-up* é uma cópia de segurança que se pode gravar em uma fita regravável, *hard disk*, CD, rede (através de *Named Pipe* "mecanismo de comunicação interprocesso") e outros dispositivos físicos de armazenamento.

Antes de se pensar em *back-up*, tem-se que programar uma estratégia adequada para *back-up* e restauração (PATTON, 2002).

O *SQL Server* disponibiliza diversas ferramentas para recuperação de dados perdidos, facilitando a implementação da estratégia de recuperação. Segue, abaixo, a descrição dessas ferramentas.

- *Create Database Back-up Wizard* – possibilita que se construam tarefas de banco de dados separadas, que podem ser programadas e executadas manualmente;
- *Database Maintenance Plan Wizard* – essa ferramenta proporciona a vantagem de poder agrupar uma ou mais tarefas de manutenção em uma função, como também executar este mesmo em mais de um banco de dados;
- *Transact-SQL* (T-SQL) – é uma extensão do *SQL* padrão que possibilita criarmos *scripts* que interagem com o *SQL Server*. Dessa forma, conseguiríamos criar nossas próprias funções de *back-up*.

8.1.2. Tipos de back-up

- *Back-up full* – faz o *back-up* de todos os arquivos de dados e de log, faz o *back-up* de todas as páginas de dados e registra a localização dos arquivos.
- *Back-up differential* – faz o *back-up* das alterações que ocorreram desde o último *back-up full*; o *back-up differential* também faz o *back-up* de log.
- *Back-up* log – faz o *back-up* de todas as alterações que ocorrem no banco de dados, faz o *back-up* de todas as instruções gravadas desde o último *back-up* do log, e depende do modelo de recuperação em que o banco de dados esteja configurado.

8.1.3. Restaurando o banco de dados

A restauração do banco de dados *SQL Server* pode ser feita utilizando recursos já incorporados ao sistema. E ainda mais: o banco de dados lhe apontará sugestões sobre *back-up*s completos, diferenciais e de logs de

transação que se utilizariam para reconstruir o banco de dados, ou se poderá fazê-lo de maneira automatizada, gerando *scripts* com *Transact-SQL* (PATTON, 2002).

9. ÍNDICES

Índices são usados para recuperação rápida de informações dos dados das tabelas. O *SQL Server* permite que os índices sejam criados em tabelas e visualizações.

Um índice consiste em um conjunto de páginas conhecidas como árvore B+ (SIQUEIRA, 2006).

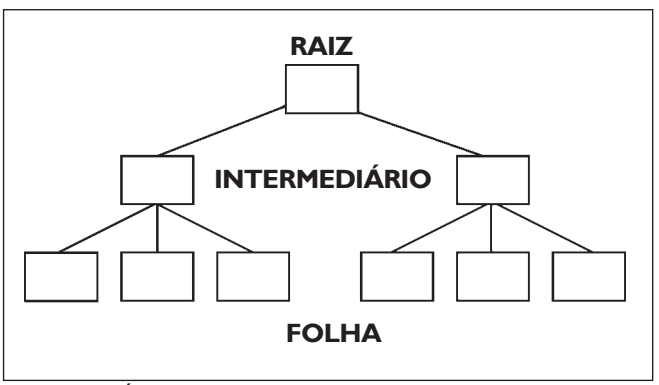

Figura 2: Árvore B+

Utiliza-se a árvore B+ para localizar a linha e, depois, dirige-se a linha de dados individual. A página raiz contém entrada de índices, assim como o ponteiro para cada página abaixo da raiz. Cada índice pode ter um ou mais índices intermediários.

A criação de índices envolve a seleção de 1 a 16 colunas, que são usadas pela lógica do *T-SQL* (*Transact-SQL*), a fim de diminuir a quantidade de busca necessária para localizar dados existentes em uma tabela. Sem a utilização de índices, seria necessário varrer toda a tabela em busca dos dados, verificando registro a registro. Porém, os índices ocupam espaço em disco, e, quando se atualiza um índice, todos os outros índices são também atualizados; portanto, ao invés de otimizar a consulta, estar-se-ia atrapalhando o desempenho do banco de dados sugerido.

Há dois tipos de índices no *SQL Server*: cada um possui requisitos e efeitos de gravação exclusivamente diferentes e efeitos de dados de tabela. Estes dois tipos são índices com *cluster* e sem *cluster*.

9.1. Opções de índices

O *SQL Server* oferece várias opções de índices. Deve-se especificar qual usar antes de criar seus índices.

9.1.2. Índices de cluster (clustered)

Um índice *cluster* é aquele onde a ordem física das páginas de dados é a mesma ordem do índice. Cada tabela está limitada a um índice com *cluster*, pois estes índices reclassificam fisicamente os dados, e os dados da tabela são localizados nas páginas que mantêm o índice com *cluster* (PASTORELO, 2006).

Os índices **agrupados** são utilizados da seguinte forma:

- colunas em que os dados são acessados freqüentemente;
- colunas usadas com *ORDER BY* e *GROUP BY*;
- colunas que são alteradas freqüentemente;
- em chaves primárias, contanto que não haja outras colunas melhores;
- em chaves estrangeiras, porque geralmente elas não são únicas.

9.1.3. Índices sem cluster (non-clustered)

Um índice sem *cluster* possui uma ordem física diferente da ordem dos dados. Cada tabela ou visualização pode ter até 249 índices sem *cluster*. De forma contrária aos índices com *cluster*, os índices sem *cluster* não classificam fisicamente os dados sublinhados; eles fornecem um ponteiro de dados no ponto de índice correspondente. Se a tabela não tiver um índice com *cluster* (chamado de um *heap*), o ponteiro é composto por identificador de arquivo, número de página e linha de dados na página. Se a tabela principal não tiver um índice com *cluster*, o ponteiro é a chave do índice com *cluster*.

Os índices **não-agrupados** são utilizados da seguinte forma:

- colunas que são usadas nas cláusulas *ORDER BY* e *GROUP BY*;
- colunas que são freqüentemente utilizadas na cláusula *WHERE*.

9.1.4. Índices únicos/não-únicos

O índice único determina se um índice permite valores duplicados. O *SQL Server* utiliza índices não-únicos por padrão, ou seja, valores duplicados são permitidos.

Esse tipo de índice agiliza muito o processo de busca do *SQL Server*, pois, uma vez que o valor procurado é achado, não são necessárias mais pesquisas.

Os índices clusterizados são sempre criados pelo *SQL Server* como únicos (SIQUEIRA, 2006).

9.1.5. Índices únicos/não-únicos

O índice composto é formado por duas ou mais colunas. Esse tipo de índice é utilizado quando duas ou mais colunas são pesquisadas em conjunto, com isso reduzindo o número de índices utilizados pelo *SQL Server* e ajustando-o para obter um melhor desempenho.

Os índices compostos podem ser *cluster* ou sem *cluster*, conter entre 2 e 16 colunas, e ter até 900 *bytes* (SIQUEIRA, 2006).

9.1.6. Índices ascendentes/descendentes

O *SQL Server* por padrão utiliza índices ascendentes, mas também podem ser implementados índices descendentes. Para se ter uma noção de como funcionam esses tipos de índices, podem-se visualizar, no alfabeto, ascendentes seriais A, B, C..., e descendentes Z, Y, X... Com esse analogia, fica fácil entender o conceito de índices ascendentes/descendentes.

REFERÊNCIAS BIBLIOGRÁFICAS

FERREIRA, Adriana. "Introdução ao *SQL Server*". Disponível em: <www.SQLmagazine.com.br/Artigos/SQLserver/06_intro_SQLserver.asp>. Acesso em: 14/10/2006.

MICROSOFT. *Administrando um banco de dados do* Microsoft SQL Server. São Paulo: Ed. Microsoft Lerning, 2000.

_____. *Programando um banco de dados do* Microsoft SQL Server. São Paulo: Ed. Microsoft Lerning, 2001.

MORELLI, Eduardo Terra. SQL Server 2000 *Fundamental*. São Paulo: Ed. Érica, 2001.

PASTORELLO, Thiago. "*SQL Server 2000* – Funcionalidades e *Features*". Disponível em: <www.linhadecodigo.com.br/artigos.asp?id_ac=583>. Acesso em: 02/09/2006.

_____. "*SQL Server* – Índices". Disponível em: <www.linhadecodigo.com.br/artigos_impressao.asp?id_ac=619>. Acesso em: 10/08/2006.

PATTON, Robert & OGLE, Jennifer. *Projetando e administrando banco de dados SQL Server 2000 como servidor* enterprise .net. Rio de Janeiro: Ed. Alta Books, 2002.

PICHILIANI, Mauro. "*Roles* e permissões no *SQL Server*". Disponível em: <www.imasters.com.br/artigo/1039/SQL_server/roles_e_permissoes_no_SQL_server>. Acesso em: 09/09/2006.

RAMALHO, José Antônio**.** *Learn Microsoft SQL Server 2000*. Texas: Ed. Wordware, 2001.

RIBEIRO, Paulo. "Versões existentes do SQL Server 2000". Disponível em: <www.devmedia.com.br/articles/viewcomp.asp?comp=2489>. Acesso em: 26/08/2006.

ROMÃO, Bruno. "O *Microsoft SQL Server*". Disponível em: <www.portaldaprogramacao.com/artigos2.asp?n=850>. Acesso em: 15/10/2006.

SHAPIRO, Jeffrey. *SQL Server 2000 – The complete reference*. Califórnia: Ed. Osborn/McGraw-Hill, 2002.

SILBERSCHATZ, A.; KORTH, H.F. & SUDARSHAN, S. *Sistemas de banco de dados*. 3. ed. São Paulo: Makron Books, 1999.

SIQUEIRA, Frank. "Controle de concorrência". Disponível em: <www.inf.ufsc.br/~frank/BDD/concorrencia.htm>. Acesso em: 14/10/2006.