

SQL magazine

Ano 10 :: Edição 117 :: R\$ 14,90



Banco de Dados
Avaliando recursos de paralelismo
apoiado por bancos de dados

Oracle

Bancos conectáveis com o Oracle 12c

SQL Server

Boas práticas de monitoramento

Oracle

Configurando alta disponibilidade

PROJETANDO BANCOS NOSQL

Conheça algumas técnicas
de modelagem utilizadas
em bancos NoSQL



SQL Server
Realizando tuning
com o SQL Server 2012

SQL Server
Analisando eventos de
espera com foco em performance

ISSN 1677918-5



Estas soluções são monitoradas pela tecnologia FairCom todos os dias!



Abastecido pela tecnologia de Banco de Dados FairCom



Confiabilidade no espaço exterior



Segurança quando você mais precisa



Mais do que sorte seu bilhete para o sucesso



Segurança Financeira em cada transação



Seu mundo conectado



O Poder por trás da logística

A FairCom é uma empresa de tecnologia de banco de dados de alta performance, que oferece aos desenvolvedores mais liberdade no gerenciamento dos dados. Por mais de 30 anos a FairCom tem sido a escolha de implementações de missão crítica em todo o mundo devido à integridade e confiabilidade de seus produtos, garantindo pouco ou nenhum tempo de parada.

- Com baixos custos de licenciamento, requisitos de hardware mínimos e administração simplificada, a FairCom oferece um dos banco de dados de menor custo da indústria em seu segmento.
- Milhares de ISVs e 43% das empresas da revista Fortune 100 nos EUA utilizam a ferramenta de banco de dados multiplataforma da FairCom para potencializar seus aplicativos.
- A FairCom, comprometida em oferecer o que há de melhor e mais moderno em tecnologia de banco de dados, trabalha lado a lado com os ISVs e outros desenvolvedores para inovar constantemente, sempre baseada nas necessidades de seus clientes e em um firme entendimento do futuro da tecnologia de banco de dados.
- A arquitetura única da FairCom permite a seus clientes recuperar o controle do desenvolvimento de seu produto em seu próprio ritmo, com confiança na flexibilidade e integridade da sua ferramenta de banco de dados.
- Garanta o melhor ROI para seu produto com a integração do gerenciamento dos dados, simplificando a distribuição e suporte.

Sumário

Conteúdo sobre Boas Práticas

05 – Modelagem SQL x NoSQL

[Mauro Pichiliani]

Conteúdo sobre Boas Práticas

10 – Realizando tuning com o SQL Server 2012

[Fernando Weschenfelder]

Conteúdo sobre Boas Práticas

16 – Eventos de espera no SQL Server

[Rafael Guidastri]

24 – Monitorando uma instância SQL Server

[Stefano Takamatsu Gioia]

Conteúdo sobre Novidades

31 – Por dentro do Oracle Database 12c

[Gilson Martins]

Conteúdo sobre Boas Práticas, Conteúdo sobre Novidades

40 – Bancos conectáveis com o Oracle 12c

[Ricardo Ferro]

Conteúdo sobre Boas Práticas

48 – Alta disponibilidade no banco de dados Oracle – Parte 3

[Ricardo Rezende]

Conteúdo sobre Boas Práticas

60 – Avaliando recursos de paralelismo apoiado por bancos de dados

[Marcelo Josué Telles]



Dê seu feedback sobre esta edição!

A SQL Magazine tem que ser feita ao seu gosto. Para isso, precisamos saber o que você, leitor, acha da revista!

Dê seu voto sobre esta edição, artigo por artigo, através do link:

www.devmedia.com.br/sqlmagazine/feedback

EXPEDIENTE

Editor

Rodrigo Oliveira Spínola (rodrigo.devmedia@gmail.com)

Subeditor

Eduardo Oliveira Spínola

Consultora Técnica

Daniella Costa (daniella.devmedia@gmail.com)

Jornalista Responsável

Kaline Dolabella - JP24185

Capa

Romulo Araujo

Diagramação

Janete Feitosa

Distribuição

FC Comercial e Distribuidora S.A
Rua Teodoro da Silva, 907
Grajaú - RJ - 206563-900

Atendimento ao leitor

A DevMedia possui uma Central de Atendimento on-line, onde você pode tirar suas dúvidas sobre serviços, enviar críticas e sugestões e falar com um de nossos atendentes. Através da nossa central também é possível alterar dados cadastrais, consultar o status de assinaturas e conferir a data de envio de suas revistas. Acesse www.devmedia.com.br/central, ou se preferir entre em contato conosco através do telefone 21 3382-5038.

Publicidade

publicidade@devmedia.com.br – 21 3382-5038

Anúncios – Anunciando nas publicações e nos sites do Grupo DevMedia, você divulga sua marca ou produto para mais de 100 mil desenvolvedores de todo o Brasil, em mais de 200 cidades. Solicite nossos Media Kits, com detalhes sobre preços e formatos de anúncios.

Fale com o Editor!

É muito importante para a equipe saber o que você está achando da revista: que tipo de artigo você gostaria de ler, que artigo você mais gostou e qual artigo você menos gostou. Fique à vontade para entrar em contato com os editores e dar a sua sugestão!

Se você estiver interessado em publicar um artigo na revista ou no site SQL Magazine, entre em contato com os editores, informando o título e mini-resumo do tema que você gostaria de publicar:

Rodrigo Oliveira Spínola - Editor da Revista
rodrigo.devmedia@gmail.com



RODRIGO OLIVEIRA SPÍNOLA

Editor Chefe da SQL Magazine, Mobile e Engenharia de Software Magazine. Professor da Faculdade Ruy Barbosa, uma instituição parte do Grupo DeVry. Doutor e Mestre em Engenharia de Software pela COPPE/UFRJ.

CURSOS ONLINE

A Revista SQL Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.

Feita para Desenvolvedores de Software e DBAs



CONHEÇA OS CURSOS MAIS RECENTES:

- **Cursos: Curso de noSQL (Redis) com Java**
- **Desenvolvimento para SQL Server com .NET**
- **Curso PostgreSQL - Treinamento de banco de dados (Curso Básico)**

Para mais informações :

[\(21\) 3382-5038](http://www.devmedia.com.br/cursos/banco-de-dados)



DEVMEDIA

Modelagem SQL x NoSQL

Veja algumas técnicas de modelagem utilizadas em bancos NoSQL

A modelagem é uma das atividades iniciais envolvidas no processo de criação de bancos de dados tradicionais que conta com diversas técnicas consolidadas e utilizadas nas tarefas do dia a dia que envolvem programação e banco de dados. Estas técnicas são diretamente ligadas aos bancos de dados relacionais que suportam a linguagem SQL. Contudo, novas abordagens para o armazenamento, manipulação e tratamento de dados abrem discussão sobre novas formas de modelagem de dados.

Apesar de existir muito material, recursos, exemplos e aplicações voltadas para os bancos de dados NoSQL, pouco se fala sobre as características, aspectos e técnicas de modelagem que podem ser utilizados com tecnologias NoSQL. De fato, esta falta de discussão acaba se tornando uma barreira de aprendizado para muitas pessoas que, muitas vezes, simplesmente desistem de experimentar tecnologias NoSQL por não encontrar uma comparação com o que já existe no mundo SQL. Desta maneira, este artigo apresenta uma discussão sobre a criação de modelos para bancos NoSQL tendo como guia as técnicas e recursos empregados na concepção e criação de bancos que suportam a linguagem SQL.

A discussão apresentada neste artigo não se propõe a ser muito extensa ou se concentrar em um banco de dados relacional ou NoSQL específico, pois o foco é nas possibilidades que a modelagem proporciona quando se está trabalhando com bancos de dados relacionais ou NoSQL.

Aspectos gerais de modelagem

De um ponto de vista mais geral, a modelagem representa uma abstração da realidade. Basicamente o que estamos fazendo quando modelamos um banco de dados é montar uma representação da realidade de acordo com o nosso objetivo, que no caso é o armazenamento de dados. Além da abstração, também realizamos outras tarefas, tais como filtrar o que nos interessa, interpretar cenários, enquadrar os dados em um formato que seja interessante e saber o que é relevante ou não para o modelo. Estas atividades são realizadas de forma iterativa, ou seja, o modelo provavelmente vai passar por várias versões antes de chegar à versão final que será

Resumo DevMan

Porque este artigo é útil:

O conteúdo deste artigo é útil para aqueles que estão começando a aprender tecnologias NoSQL e possuem dificuldade para se adaptar e utilizar o conhecimento já adquirido na modelagem relacional. A partir da discussão apresentada é possível compreender melhor como a modelagem funciona nos bancos de dados NoSQL do ponto de vista da modelagem de entidades, relacionamentos e atributos. Neste contexto são discutidos exemplos de como as integridades de entidade, de domínio e referencial afetam a modelagem tradicional e como estes aspectos são tratados pelas tecnologias agrupadas pela sigla NoSQL.

implementada em um sistema gerenciador de banco de dados relacional (SGBDR).

A modelagem relacional tradicional possui algumas características que a tornaram adequada para muitos tipos de ocorrências de dados que, de certa forma, sempre envolvem um usuário final. Um dos aspectos que pouca gente nota quando se fala em modelagem tradicional é que os modelos geralmente são criados pensando-se em geração de relatórios com algum nível de agregação de dados. Esta modelagem também é, até certo ponto, influenciada pela própria linguagem SQL, que possui muitos recursos para trabalhar com conjuntos de dados ao invés de dados individuais. Também existe certa influência do uso de transações nas características de modelagem, assim como a consistência e integridade de dados.

Quem está aprendendo a modelar bancos de dados tradicionais logo nota que há uma preocupação grande em garantir que o modelo de dados seja rígido, ou seja, não aceite grandes variações fora do que foi modelado. Há também uma preocupação constante em garantir que anomalias de dados não sejam inseridas no modelo por meio da aplicação das famosas regras de normalização ou formas normais.

Durante a fase de modelagem tradicional é comum que os modelos recebam muitas modificações para que eles estejam de acordo com as recomendações da terceira forma normal. Isso quer dizer que o modelo deve receber modificações tais como a criação de novas entidades, atributos e relacionamentos, para que certas anomalias não ocorram nos dados. Esta é uma das principais características de modelos relacionais e que acaba

guiando a modelagem para um lado mais rígido, adequando-se bem ao uso de transações, concorrência e integridade do esquema do banco de dados.

Contudo, recentemente os bancos de dados agrupados sob a sigla NoSQL propuseram abordagens diferentes para o armazenamento, manipulação de dados e modelagem de dados. O que era considerado tradicional e desejável no modelo relacional começou a ser desafiado pelos bancos NoSQL, pois o uso de transações, rigidez no esquema, integridade e também aspectos de escalabilidade, alta disponibilidade e concorrência de acesso nem sempre são desejáveis como nos bancos SQL. Certamente esta nova forma de pensar e ver o banco de dados gera um impacto na modelagem que precisa ser conhecido para que se possa escolher, dependendo do contexto, qual é o melhor tipo de modelagem a ser adotado em cada situação.

Quem está iniciando os estudos em tecnologias NoSQL não deve ter em mente que tudo o que foi aprendido deve ser deixado de lado. Muito do que foi aprendido com modelagem tradicional pode e deve ser utilizado na modelagem NoSQL. Algumas técnicas, recomendações e recursos são praticamente iguais, enquanto outras exigem um esforço de aprendizado. No geral, é preciso sempre focar na solução e como a modelagem pode ser utilizada para atender as necessidades do negócio independente de como a modelagem for implementada.

A modelagem tradicional empregada para atender as necessidades de armazenamento já existe há muitos anos e diversos modelos de dados foram criados desta forma. Há, inclusive, repositórios com diversos modelos de dados prontos para serem utilizados. Infelizmente, a modelagem para bancos NoSQL não conta com tantos exemplos, recursos educacionais e discussões que possam auxiliar quem está começando a trabalhar com as tecnologias NoSQL. Contudo, a falta de recursos não deve ser um fator que impeça a avaliação e o uso de modelos NoSQL no dia a dia das soluções, independente de qual paradigma de armazenamento seja mais adequado.

Apesar de existirem diferentes formas de armazenar e organizar os dados nos produtos agrupados pela sigla NoSQL, a próxima seção vai discutir alguns dos principais aspectos relacionados à modelagem de um banco de dados relacional ou não levando em consideração os SBGDR tradicionais e algumas abordagens NoSQL mais populares.

Diferenças de modelagem entre NoSQL e SQL

Um dos principais pontos que diferenciam a modelagem de bancos de dados NoSQL da modelagem tradicional é a rigidez do modelo. Nas técnicas tradicionais de criação deste artefato é necessário fazer um levantamento de requisitos e especificar com detalhes cada uma das entidades, relacionamentos e atributos que, no momento da criação do banco de dados físico, vão se tornar tabelas, relacionamentos (implementados por constraints) e colunas, respectivamente. Uma vez que este levantamento seja feito, é necessário especificar quais são os atributos das entidades que futuramente se tornarão as colunas das tabelas.

Os atributos da modelagem tradicional são rígidos, ou seja, todos eles são necessários para cada instância da entidade. Isso quer dizer que cada linha da tabela deverá ser composta pelos valores das colunas que foram criadas. Por exemplo, se uma entidade possui 10 atributos, espera-se que cada um deles receba um valor para cada instância da entidade. Se houver uma tentativa de inserir menos ou mais do que 10 valores para uma nova instância (linha) desta entidade (tabela), um erro deverá ser retornado indicando uma violação de integridade de entidade.

Em bancos de dados NoSQL orientados a documento esta rigidez não existe. É possível criar uma coleção (análogo a uma entidade ou tabela), que recebe os documentos (análogo a instâncias da entidade ou linhas) e que possui atributos (análogo a colunas da tabela). Contudo, a definição de quais atributos o documento irá possuir é criada de acordo com os dados, tornando o modelo flexível. Isso quer dizer que é possível ter em uma coleção um documento com três atributos e outro documento dentro da mesma coleção com 10 atributos. Esta flexibilidade é característica de muitos bancos de dados NoSQL e gera novas possibilidades de armazenamento.

Vejamos um exemplo onde a flexibilidade pode ser útil. Em um cadastro de produtos de uma loja é provável que cada produto tenha características únicas que precisam ser analisados. Um tênis possui cor, tamanho e marca. Já uma bebida possui sabor, data de validade e conteúdo calórico. Geralmente a modelagem tradicional resolve esta situação criando uma tabela de produtos com um atributo de texto livre para armazenamento de uma descrição ou observação. Já a modelagem NoSQL pode separar melhor estas características em atributos que não foram criados durante a modelagem. Desta forma é possível armazenar dados de acordo com as características dos produtos sem a necessidade de uma modelagem rígida.

Até aqui vimos algo que vai contra a integridade de entidade de uma tabela, pois tal integridade diz que o formato dos dados deve ser o mesmo para cada instância da entidade. Entretanto, muitos cenários não exigem a integridade de entidade devido a certas características dos dados. Nestes casos é recomendado trabalhar com um esquema de banco de dados que seja flexível e permita a inserção de valores para atributos que não foram previamente modelados.

A flexibilidade de esquema também pode ser utilizada para armazenar diferentes valores para um mesmo atributo, uma vez que nos bancos NoSQL orientados a documentos cada documento dentro de uma coleção é tratado de forma individual. Isso quer dizer que, por exemplo, no atributo CODIGO podemos ter um valor numérico para um documento X e um valor alfanumérico para o atributo CODIGO em um documento Y, sendo que ambos os documentos estão dentro da mesma coleção C. Isso pode acontecer em cenários onde cada fabricante tem um código interno do seu produto.

Geralmente a modelagem tradicional resolve esta situação de integridade de domínio escolhendo um tipo de dados que armazena

qualquer tipo de valor ou criando diferentes atributos dentro da mesma entidade. Nos bancos NoSQL orientados a documentos não é preciso nenhum trabalho adicional, pois, como já foi dito, cada documento é tratado de forma individual.

O terceiro tipo de integridade que se encontra em um modelo relacional tradicional é a integridade referencial. Este tipo de integridade garante a correspondência entre instâncias de entidades e é utilizado para estabelecer os relacionamentos. Na fase de modelagem física são utilizadas as constraints chave estrangeira e chave primária para garantir a integridade referencial destes relacionamentos.

Uma das principais dúvidas de quem está começando a aprender bancos de dados NoSQL é sobre os relacionamentos e os joins. Os iniciantes em NoSQL sentem dificuldade em compreender que não é necessário fazer joins e relacionar as entidades ou coleções no mundo NoSQL, pois o tipo de tratamento para obter os dados é outro.

Por exemplo, a Figura 1 mostra um modelo simples de usuários (*users*) e suas habilidades (*skill*). Neste cenário, a modelagem relacional tradicional cria três tabelas representadas pelas colunas da Figura 1: uma tabela para armazenar os usuários (*users*), uma tabela para as habilidades (*skill*) e uma tabela de ligação entre usuários e habilidades (*user_skill*), configurando um relacionamento N:M que foi implementado como dois relacionamentos 1:N.

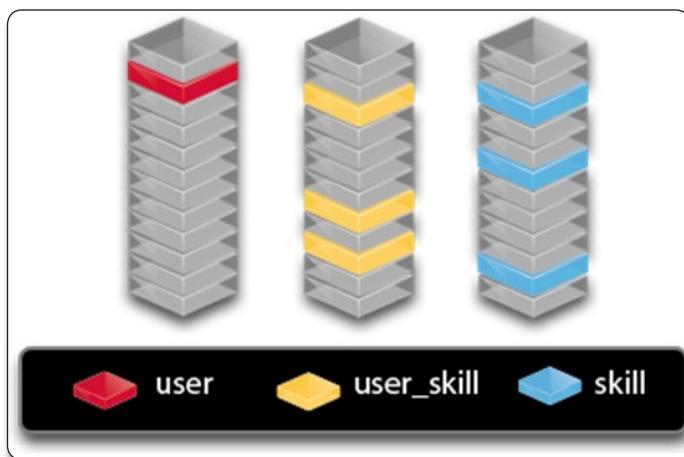


Figura 1. Relacionamento clássico entre usuários e habilidades

De acordo com a Figura 1, um usuário é representado pela cor vermelha, três habilidades que ele possui são representadas pela cor azul clara e a indicação que este usuário possui estas habilidades é indicada pela cor amarela. Esta é a maneira tradicional e clássica de representar este relacionamento.

O relacionamento entre usuários e habilidades pode ser representado de diversas maneiras no mundo NoSQL, dependendo de qual paradigma se utiliza para armazenar os dados. Contudo, é fato que as tecnologias NoSQL possuem uma tendência a representar entidades e relacionamentos de forma diferente. No caso da modelagem tradicional, o relacionamento assumiu a forma de uma entidade adicional (*user_skill*) e o modelo final acabou

com três entidades. Supondo que desejamos modelar este mesmo cenário em um banco de dados NoSQL orientado a grafos, uma possível representação é mostrada na Figura 2, onde nota-se que a modelagem do cenário é feita de acordo com nós e arestas de um grafo.

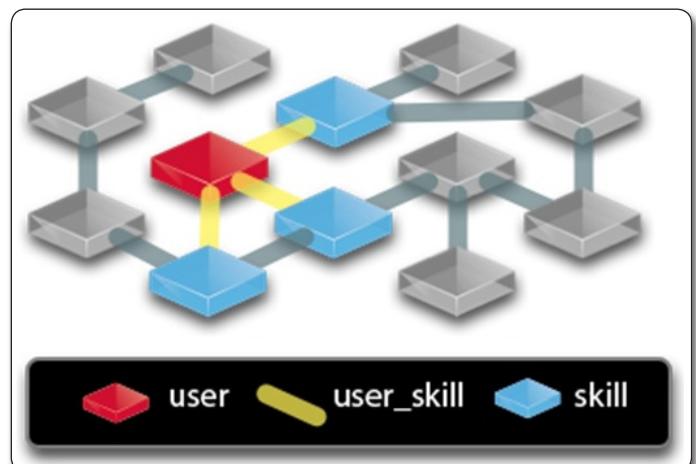


Figura 2. Representação do relacionamento entre usuários e habilidades em um grafo

Além da representação na forma de grafo, é muito comum encontrar os relacionamentos do tipo N:M ou 1:N embutidos dentro de uma mesma entidade nos bancos NoSQL orientados a documentos. Tal representação é realizada por meio de um atributo que contém outra coleção que por sua vez contém diferentes documentos. Este tipo de abordagem embutida permite encadear dentro de uma mesma entidade diferentes tipos de relacionamentos sem a necessidade de se criar novas entidades no modelo. É uma maneira diferente de enxergar os relacionamentos sem precisar separá-los em entidades individuais e depois relacioná-las de alguma forma.

A Figura 3 exemplifica este tipo de relacionamento embutido colocando lado a lado como seria a modelagem tradicional das entidades *Book*, *Album*, *Track* e *Jeans* no formato relacional (lado esquerdo da figura) e no formato embutido (lado direito da figura) como um documento JSON, que é uma forma de representação de dados utilizada por bancos de dados NoSQL orientados a documentos.

É possível notar na Figura 3 que certos atributos das entidades da direita representam outros documentos com outras características, porém eles estão dentro dos documentos da coleção. Este é o caso do atributo *Details* da coleção *Product*, pois ele possui os atributos *Model*, *Length* e *Width* que não existem em *Product*.

A modelagem tradicional possui uma tendência de se concentrar mais nas entidades do que nos relacionamentos. Entretanto, na modelagem voltada para bancos de dados NoSQL os relacionamentos são traduzidos de forma diferenciada devido à possibilidade de representá-los de diferentes formas (arestas em grafos, embutidos em documentos) e também à flexibilidade de criação de atributos para as entidades. Isto faz com que cenários onde seja

Modelagem SQL x NoSQL

necessário representar hierarquias, múltiplos relacionamentos de acordo com o tempo, árvores e grafos, sejam mais atrativos para a modelagem NoSQL do que a modelagem relacional. Devido a esta característica não é a toa que cada vez mais profissionais veem optando por tecnologias NoSQL para modelar dados provenientes de redes sociais e relacionamentos entre pessoas, objetos, situações, eventos, preferências e comportamentos.

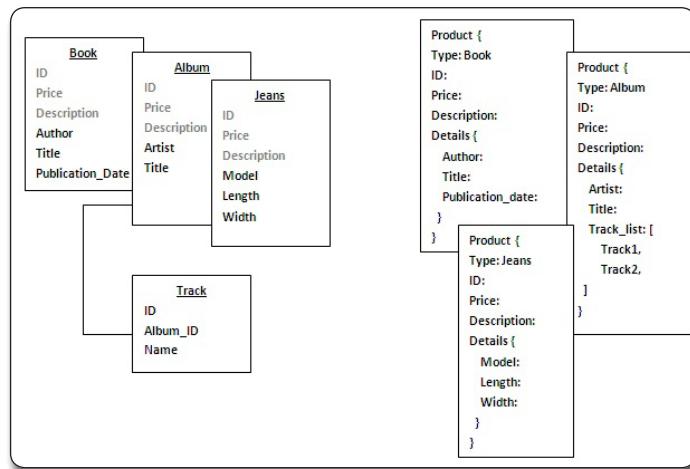


Figura 3. Exemplo de relacionamento embutido no formato JSON

Outro grande motivador para o uso de tecnologias NoSQL que remete à modelagem é o armazenamento de objetos utilizados pela aplicação diretamente no banco de dados. Porém este armazenamento nem sempre é direto, ou seja, a aplicação deve fazer uso de algum framework ou ORM (Object-Relational Mapping) para realizar a persistência dos objetos. Aqui, tanto as tecnologias NoSQL como SQL permitem este tipo de armazenamento, no entanto bancos de dados de documentos possuem uma pequena vantagem por utilizar internamente formatos como JSON, que já são apropriados para a serialização e persistência de objetos.

Os profissionais que estão acostumados a enxergar os relacionamentos entre entidades da maneira tradicional podem ficar um pouco confusos com relacionamentos embutidos ou a representação dos relacionamentos na forma de um grafo. Para explicar melhor, vale a pena recorrer a aquela famosa frase: “Quando tudo que se tem é um martelo, a tendência é imaginar todos os problemas como um prego e sair martelando tudo”.

A modelagem tradicional trata os relacionamentos através da colocação de atributos em entidades diferentes. Uma vez que o relacionamento esteja criado no modelo, é comum utilizar um JOIN em uma instrução que compara o valor de atributos em diferentes tabelas.

CURSOS ONLINE

A Revista Clube Delphi oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.



CONHEÇA ALGUNS DOS CURSOS:

- **Curso de Multicamadas com Delphi e DataSnap**
- **Delphi para Iniciantes**
- **Criando componente Boleto em Delphi**
- **Loja Virtual em Delphi Prism**

Para mais informações :

www.devmedia.com.br/cursos/delphi
(21) 3382-5038

A maneira de se obter dados através dos relacionamentos é um pouco diferente com as tecnologias NoSQL. Nestes cenários utiliza-se algum tipo de algoritmo para percorrer o grafo ou rotina recursiva para chegar até um nível específico de dado dentro de vários documentos. Este tipo de mudança de pensamento deve ser encarado não como algo mais complexo e difícil, mas sim como algo que é adequado e faz muito sentido em determinados cenários. O mesmo vale para outras tarefas comuns no mundo SQL, como filtros (cláusula WHERE), agregações (cláusulas GROUP BY e HAVING) e ordenações (cláusula ORDER BY), apenas para citar alguns exemplos.

É importante destacar que tanto a modelagem tradicional como a modelagem NoSQL permitem fazer o armazenamento de praticamente qualquer dado em qualquer cenário. A diferença está nas possibilidades, quantidade de esforço de implementação, recursos e outros elementos práticos que podem tornar mais simples e rápido o desenvolvimento da solução e do modelo de acordo com o contexto e características dos dados. Quem já possui alguma experiência com modelagem sabe que sempre vai haver múltiplas soluções para os problemas dependendo do contexto, porém cada uma possui certas vantagens e desvantagens que devem ser analisadas além das recomendações gerais, fórmulas prontas, diretrizes e comentários do tipo ‘via de regra’.

É claro que existem várias implicações, tanto para o banco de dados como para as aplicações, quando se utiliza as técnicas de NoSQL de modelagem, que vão contra as integridades de entidade, domínio e referencial. Contudo, aqui é preciso estar com a mente aberta para compreender novas formas de se enxergar como os dados são modelados e vão se comportar quando eles estiverem armazenados em um banco de dados NoSQL. Em outras palavras, é preciso ter mente aberta para compreender outras abordagens que vão contra o que já está estabelecido pela modelagem relacional tradicional.

Conclusão

Este artigo apresentou uma discussão sobre a modelagem de bancos de dados relacional e bancos de dados NoSQL. As principais técnicas de modelagem tradicionais foram apresentadas e comparadas com o que é possível realizar com as tecnologias NoSQL.

Saber modelar um banco de dados exige muito mais do que listar entidades, criar relacionamentos e detalhar atributos. Modelar é uma tarefa que envolve compreensão, representação, abstração, organização e outros aspectos que se distanciam das atividades práticas de implementação de um modelo de dados, seja ele um modelo relacional ou não.

As tecnologias NoSQL permitem novas maneiras de se pensar em como a modelagem pode ser realizada, especialmente quando há necessidade de flexibilização de armazenamento e representação de relacionamentos complexos e não há uma necessidade grande de manter as integridades de domínio, entidade e relacional.

Autor



Mauro Pichiliani

mauro@pichiliani.com.br / @pichiliani

É bacharel em Ciência da Computação, mestre e doutorando pelo ITA (Instituto Tecnológico de Aeronáutica) e MCP, MCDBA e MCTS. Trabalha há mais de 10 anos utilizando diversos bancos de dados SQL e NoSQL. É colunista de SQL Server do web site iMasters (<http://www.imasters.com.br>) e mantenedor do DatabaseCast (@databascast), o podcast brasileiro sobre banco de dados.



Links:

Artigo “NoSQL Data Modeling Techniques”, de Ilya Katsov

<http://highlyscalable.wordpress.com/2012/03/01/nosql-data-modeling-techniques/>

Repositório de modelos relacionais Database Answer

http://www.databaseanswers.org/data_models/

Você gostou deste artigo?

Dê seu voto em www.devmedia.com.br/sqlmagazine/feedback

Ajude-nos a manter a qualidade da revista!



Realizando tuning com o SQL Server 2012

Conheça recursos importantes para a execução de tuning de banco de dados

Com a crescente demanda por tecnologia da informação dentro das organizações, diversos aspectos são levados em consideração pela área de TI visando a entrega de serviços e sistemas para alavancar e sustentar os negócios, como a disponibilidade e o desempenho, itens que geralmente andam correlacionados e que exigem altos investimentos.

Para garantir que um sistema de informação tenha uma performance adequada aos processos de negócio e não gere discontentamentos e reclamações por parte dos usuários, um banco de dados deve garantir que todas as consultas e transações SQL sejam retornadas o mais rápido possível, de forma consistente e sem pontos de falha. Para isso, é preciso que o administrador de banco de dados saiba como monitorar seu banco de forma prática e eficiente, aplicando conceitos para realização de *tuning* com o objetivo de garantir rápida resposta aos processos que são submetidos à instância. Além disso, é importante que o DBA conheça as características e a criticidade dos principais sistemas utilizados nas empresas em que atua, para que a análise seja completa e mais precisa possível.

Neste contexto, dando ênfase especificamente ao desempenho em um ambiente de produção, este artigo irá abordar conceitos e dicas para análise de sessões no banco de dados, bem como a identificação de processos consumidores de recursos através de ferramentas do próprio SQL Server, como SQL Server Management Studio e SQL Profiler. Além disso, serão destacadas algumas recomendações sobre a utilização de alternativas para controlar o consumo de CPU e memória em um servidor SQL Server, como também o uso de novos objetos para administração e monitoramento de um banco SQL.

Desempenho em um banco de dados

Diariamente, em uma equipe de desenvolvimento de sistemas, diversas funcionalidades são desenvolvidas tendo como um dos seus objetivos a entrega cada vez

Resumo DevMan

Porque este artigo é útil:

Neste artigo serão apresentados recursos fundamentais para monitoramento de banco de dados com relação a desempenho. Neste contexto, serão exemplificadas algumas alternativas para se obter diagnósticos de problemas de alto consumo de recursos em um servidor SQL Server, visando a realização de tuning por parte do DBA. O objetivo principal do tuning em um banco de dados é a resolução de problemas relacionados a desempenho que tornam insatisfatória a realização de consultas e transações SQL. Os sintomas de lentidão podem ser frequentemente relatados por usuários de sistemas ou de banco de dados; então cabe ao DBA a tarefa de investigar, diagnosticar e resolver o problema através da criação de índices, geração e atualização de estatísticas ou até mesmo alterar a lógica de alguma consulta SQL, tendo como meta a melhoria contínua da performance.

mais frequente para atender a solicitações para correções de bugs e/ou melhorias. No entanto, após o setor de desenvolvimento realizar as liberações das rotinas criadas ou alteradas para a equipe de testes, geralmente o que ocorre são validações sobre a lógica e operacionalidade do sistema. Deste modo, um aspecto fundamental para o usuário final acaba sendo desprezado: o desempenho.

Com relação a este importante requisito, permanecem abertas algumas questões sobre as dificuldades que as áreas de desenvolvimento de sistemas enfrentam para manter uma estrutura adequada para a execução dos testes, tais como custos operacionais, de infraestrutura e de pessoal. Quanto à rotina de testes, a mesma deveria ser executada tendo como base o número de sessões existentes no banco de dados de produção, ou ao menos, baseando-se em sua proporcionalidade. Porém, na prática, os testes pré-produção quase nunca refletem a realidade do contexto organizacional, justamente por seguirem um fluxo onde poucas sessões de banco de dados são abertas para submeter o processo de execução dos testes. Como resultado disso, os reflexos dos problemas de performance são observados

quase sempre em produção, onde diversos sistemas são executados de forma paralela, gerando concorrência entre sessões ativas e sobrecarga no servidor de banco.

Partindo desta problemática, um item de fundamental importância para um DBA, dentre tantos outros, é avaliar o desempenho do banco de dados já pensando na possibilidade de realizar o *tuning*. No SQL Server, diversas são as possibilidades de se alcançar este objetivo, seja através da execução de scripts de monitoração, ferramentas gráficas que geram *dashboards* com indicadores e métricas sobre o tempo de execução de rotinas, ou mesmo com o resumo dos recursos consumidos em uma instância SQL. Assim, cada administrador de banco fará sua análise de acordo com sua necessidade, até porque os ambientes de produção variam muito de empresa para empresa.

Aliado a estes aspectos, podem ser observados alguns benefícios com relação ao monitoramento de bancos de dados SQL Server, tais como:

- Ao monitorar os tempos de resposta de consultas e transações SQL, é possível determinar se o desempenho é satisfatório ou se há necessidade de criação de novos índices, bem como a eliminação de outros;
- Ao monitorar as consultas SQL que são disparadas na instância, pode-se averiguar se as mesmas retornam os resultados desejados ou se elas podem sofrer ajustes em sua lógica, visando otimizar o tempo de resposta e garantir que somente os dados necessários sejam retornados ao usuário, ou a outro procedimento existente no banco;
- Observar os planos de execução das consultas SQL para identificação de problemas com leituras completas em uma tabela, também chamadas de *Full Table Scan*, evitando que taxas excessivas de leituras a disco sejam recorrentes no banco.

Estes são apenas alguns dos benefícios quando falamos em *tuning* de banco de dados, os quais certamente poderão garantir a satisfação dos usuários dos sistemas existentes nas organizações, provendo a máxima otimização deste poderoso gerenciador de banco chamado SQL Server.

Recursos para monitorar um banco

Certamente um aspecto de extrema importância para um administrador de banco de dados é conhecer as diferentes maneiras de se realizar *tuning* em uma instância SQL Server. No entanto, para que isto seja possível, é necessário obter conhecimento sobre as ferramentas que podem ser empregadas para identificar e diagnosticar problemas de performance em um servidor de banco de dados. Uma destas ferramentas, que existe desde a versão 2000 do SQL Server, é o Activity Monitor, aprimorado na versão 2012 para dar maior visibilidade gráfica para o monitoramento das sessões conectadas à instância. O Activity Monitor pode ser acionado através do SQL Server Management Studio, utilizando um ícone existente na barra de tarefas, ou após clicar com o botão direito sobre o servidor de banco e selecionar a opção Activity Monitor.

Uma vez acionada a ferramenta, pode-se definir o intervalo de tempo de refresh em que o SQL Server usará para trazer as informações para os gráficos, bem como o conjunto de sessões conectadas, podendo-se filtrar pelo status das transações, como *runnable*, *running* ou *suspended*, para melhor identificação das sessões ativas. Além destes recursos e outras possibilidades de filtragem, a ferramenta ainda apresenta

uma lista dos diferentes tipos de recursos em espera (como PAGEIOLATCH_EX, ASYNC_NETWORK_IO e CXPACKET), gráficos de consumo de I/O por *datafiles* e principais transações SQL executadas na instância, podendo-se ordenar o resultado pelo consumo de CPU, leituras lógicas e físicas em disco ou consultas com maior consumo de memória e tempo de resposta, conforme podemos observar na **Figura 1**.

O Activity Monitor ainda permite capturar as transações SQL que estão sendo executadas no banco, bem como monitorar *locks* e eliminar sessões através da opção *kill*.

Além desta excelente alternativa para capturar sessões em um banco de dados SQL Server, existem outras opções para auxiliar o monitoramento e diagnosticar problemas relacionados à performance, como as *stored procedures* de sistema. Através delas é possível, por exemplo, realizar o rastreamento de sessões SQL utilizando o seguinte conjunto de *procedures*: *sp_trace_create*, *sp_trace_generateevent*, *sp_trace_setevent*, *sp_trace_setfilter* e *sp_trace_setstatus*. Estas são usadas para monitorar sessões SQL conectadas na instância e identificar as transações executadas.

Ainda analisando as *procedures* do SQL Server como alternativas para o monitoramento, podemos observar

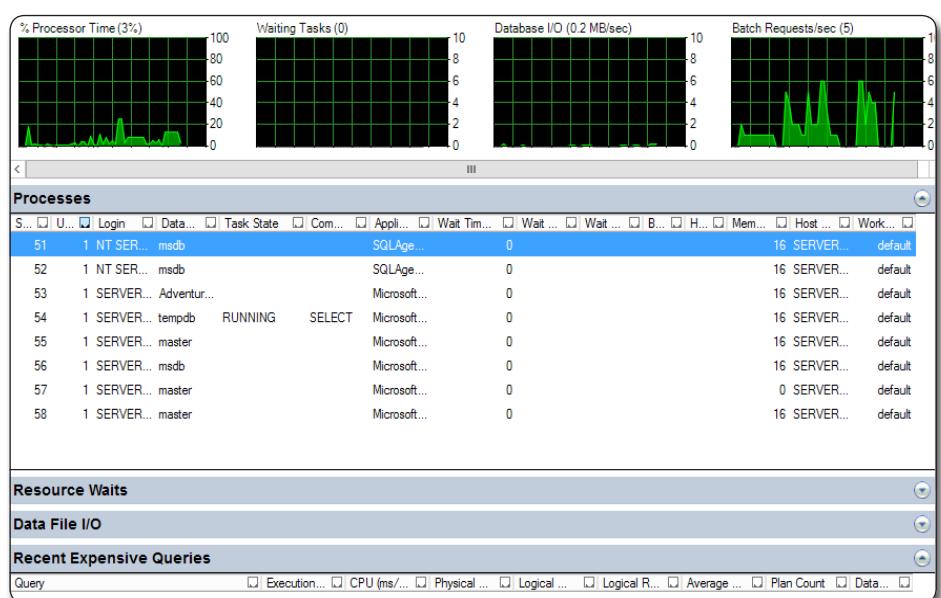


Figura 1. Visão geral dos gráficos existentes na ferramenta Activity Monitor

outras que são muito utilizadas para monitorar sessões de usuário. São elas: *sp_who*, que fornece informações de usuários e processos ativos, mostrando inclusive a instrução executada no banco; *sp_lock*, que apresenta informações sobre bloqueios, como ID do objeto da base de dados, ID do índice, tipo de bloqueio ou recurso ao qual se aplica o bloqueio; *sp_spaceused*, a qual mostra uma estimativa da quantidade atual de espaço em disco utilizada por uma tabela ou pelo banco de dados inteiro; e *sp_monitor*, que exibe estatísticas, uso de CPU, uso de I/O e a quantidade de tempo ocioso de uma sessão desde a última execução desta *stored procedure*.

Já com relação à identificação de processos ativos no SQL Server, com o objetivo de capturar eventos de sessões da base, como o início de uma transação até sua conclusão, monitoramento de erros, *deadlocks* e consumo de recursos físicos no servidor de banco de dados, pode ser utilizada a ferramenta SQL Server Profiler. Através dela, os dados coletados para uma ou mais sessões da base podem ser salvos em uma tabela ou exportados para um arquivo, para ser posteriormente analisado pelo DBA no diagnóstico de algum problema existente.

Para monitorar as transações e sessões de um banco de dados SQL Server existem ainda alternativas como o System Monitor, que contribui para rastrear o uso de recursos do servidor, tal como o número de solicitações de páginas do gerenciador de *buffers* em uso, permitindo analisar o desempenho e a atividade do servidor por meio de objetos ou contadores definidos pelo usuário para monitorar eventos. O System Monitor deve ser acionado a partir do Windows Server. Com ele, contadores de desempenho para uma ou mais instâncias do SQL Server podem ser criados de forma muito simples.

De maneira sucinta e para rápida consulta, um banco de dados SQL Server contempla ainda a existência de funções internas que exibem estatísticas que mostram a atividade do SQL Server desde o momento em que o servidor foi

iniciado. Essas estatísticas são armazenadas em contadores pré-definidos do SQL Server. Como exemplos dessas funções, podem-se citar: *@@CPU_BUSY*, que contém indicações sobre o tempo de execução do SQL Server pela CPU; *@@CONNECTIONS*, que exibe o número de conexões ou tentativas de conexão do SQL Server; e *@@PACKET_ERRORS*, que mostra o número de pacotes de rede existentes nas conexões do SQL Server.

Por fim, o SQL Server ainda possibilita examinar as estatísticas de desempenho e a consistência lógica e física de um banco de dados, usando instruções DBCC (*Database Console Command*), bem como realizar, através do recurso *Display the Estimated Execution Plan*, uma análise sobre o desempenho das instruções SQL executadas na base que se deseja ajustar. O otimizador de banco de dados, que utiliza as estatísticas do SQL Server, dá recomendações sobre adição, remoção ou modificação de índices.

Tendo como base estas alternativas para análise de performance de um banco, é possível monitorar uma base de dados SQL Server visando identificar processos e rotinas SQL que apresentam lentidão na instância. Para isto, uma base deve ser verificada com frequência e os resultados devem servir para gerar um diagnóstico com o intuito de identificar a causa de um possível problema. Com base nestas opções indicadas pelo artigo, administradores de bancos de dados podem criar suas próprias consultas SQL para verificar e capturar sessões no banco.

Alternativas para capturar sessões no banco

Embora o SQL Server disponibilize algumas *procedures*, instruções DBCC (BOX 1), funções e ferramentas para monitorar processos no banco de dados, conforme vimos, existem outras maneiras de descobrir sessões causadoras de problemas de performance, tais como retenção de tempo de CPU, consumo elevado de memória ou concorrência de disco (I/O). Uma delas é conhecida como *Dynamic Management Views* (DMVs), ou visões de gerenciamento dinâmico, que servem para retornar informações sobre o

estado do servidor, identificar problemas e proporcionar o ajuste do desempenho em uma instância SQL Server. Estas *views* iniciam sempre com o prefixo “dm” em seus nomes, para diferenciar dos demais objetos de sistema. Através do site da Microsoft podem-se obter informações de forma completa sobre estas *views*, como suas características e principais funcionalidades.

BOX 1. Instruções DBCC

Instruções DBCC (Database Console Command ou Comandos de Console de Banco de Dados) servem para realizar tarefas em uma instância SQL Server com o objetivo de validar a integridade do banco de dados, bem como habilitar o rastreamento de sessões e retornar informações diversas sobre objetos do banco de dados.

Para exemplificar o uso de tabelas de sistema e das DMVs, duas consultas muito utilizadas para monitorar sessões ativas numa instância SQL Server são apresentadas na **Listagem 1**.

Ambas apresentam resultados semelhantes, pois são filtradas somente as sessões ativas por meio da coluna *status* da tabela *sysprocesses* (existente na base Master) e da coluna *status* da view *dm_exec_requests*. A principal diferença é que a segunda consulta retorna uma coluna chamada *statement_executing*, que é responsável por retornar a instrução SQL executada naquele momento no banco.

Estas consultas provavelmente serão alternativas quase que imediatas para uma análise de sessões ativas em um momento de lentidão no banco de dados. Assim, uma vez identificadas as sessões que demoram muito tempo para sair do *status* de ativa, tais como *suspended*, *running* ou *runnable*, é prudente capturar a transação SQL que está rodando e iniciar o processo de *tuning*, o qual será detalhado mais adiante.

Mantendo o foco sobre a análise das sessões existentes no banco de dados, é possível também verificar quais objetos estão consumindo mais recursos em uma instância, como é mostrado, por exemplo, na **Listagem 2**.

Neste exemplo, a consulta retorna uma lista com as 25 *stored procedures* executadas

na instância que apresentaram as taxas mais altas de leituras lógicas, em ordem decrescente. Este indicador poderá sugerir quais objetos necessitam de revisão em sua lógica, ou até mesmo ajustes para readequação de índices nas respectivas colunas utilizadas. As *procedures* retornadas fazem parte das estatísticas do SQL Server, coletadas desde o último reinício da instância do banco.

Listagem 1. Consultas para identificar sessões ativas.

```
-- Consulta 1 - VERIFICA PROCESSOS ATIVOS
Select Processo = spid
,Computador = hostname
,Usuario = loginame
,[Status] = [status]
,BloqueadoPor = blocked
,TipoComando = cmd
,Aplicativo = program_name
from master..sysprocesses
where status in ('runnable','suspended','running')
order by blocked desc, status, spid

-- Consulta 2 - VERIFICA SESSÕES ATIVAS COM SQL STATEMENT
SELECT r.session_id ,
r.[status],
r.wait_type ,
r.scheduler_id ,
SUBSTRING(qt.[text], r.statement_start_offset / 2,
(CASE WHEN r.statement_end_offset = -1
      THEN LEN(CONVERT(NVARCHAR(MAX), qt.[text])) * 2
      ELSE r.statement_end_offset
      END - r.statement_start_offset ) / 2) AS [statement_executing] ,
DB_NAME(qt.[dbid]) AS [DatabaseName] ,
OBJECT_NAME(qt.objectid) AS [ObjectName] ,
r.cpu_time ,
r.total_elapsed_time ,
r.reads ,
r.writes ,
r.logical_reads ,
r.plan_handle
FROM sys.dm_exec_requests AS r
CROSS APPLY sys.dm_exec_sql_text(sql_handle) AS qt
WHERE status IN ('suspended','running','runnable')
ORDER BY r.scheduler_id ,
r.[status] ,
r.session_id ;
```

Listagem 2. Consulta que retorna stored procedures com maiores taxas de leituras lógicas.

```
SELECTTOP ( 25 )
p.name AS [SP Name] ,
qs.total_logical_reads AS [TotalLogicalReads] ,
qs.total_logical_reads / qs.execution_count AS [AvgLogicalReads] ,
qs.execution_count ,
ISNULL(qs.execution_count /
DATEDIFF(Second, qs.cached_time, GETDATE()),0) AS [Calls/Second] ,
qs.total_elapsed_time ,
qs.total_elapsed_time / qs.execution_count AS [avg_elapsed_time] ,
qs.cached_time
FROM sys.procedures AS p
INNER JOIN sys.dm_exec_procedure_stats AS qs
ON p.[object_id] = qs.[object_id]
WHERE qs.database_id = DB_ID()
ORDER BY qs.total_logical_reads DESC ;
```

Ainda com relação aos diferentes objetos de um banco de dados, existe uma alternativa interessante para se monitorar e observar quais são estes objetos dentro de uma base específica que estão consumindo mais memória RAM, como podemos verificar na **Listagem 3**.

Listagem 3. Consulta que retorna os objetos com maior consumo de memória na base.

```
SELECT OBJECT_NAME(p.[object_id]) AS [ObjectName] ,
p.index_id ,
COUNT(*) / 128 AS [Buffer size(MB)] ,
COUNT(*) AS [Buffer_count]
FROM sys.allocation_units AS a
INNER JOIN sys.dm_os_buffer_descriptors
      AS b ON a.allocation_unit_id = b.allocation_unit_id
INNER JOIN sys.partitions AS p ON a.container_id = p.hobt_id
WHERE b.database_id = DB_ID()
AND p.[object_id] > 100
GROUP BY p.[object_id] ,
p.index_id
ORDER BY buffer_count DESC ;
```

**Não perca tempo
reinventando a roda!**

COBREBEMX

**Componente completo para sua
Cobrança por Boleto Bancário
e Débito em Conta Corrente**

**Mais de 40 exemplos
em diversas linguagens
de programação**

**Geração e leitura de arquivos
(remessa e retorno) nos padrões
FEBRABAN e CNAB**

**Testes e Downloads
gratuitos em nosso site**

**ACESSE E CONHEÇA O COMPONENTE EM:
WWW.COBBREM.COM**

Realizando tuning com o SQL Server 2012

Neste caso, uma informação muito importante para diagnóstico é o resultado do campo *Buffer size(MB)*, que apresenta o valor em MB que o respectivo objeto está consumindo na instância do banco de dados. Para este exemplo, foi utilizada a base de experimentos já consagrada no SQL Server, AdventureWorks2012, disponível para download de forma gratuita. Na **Figura 2** podemos verificar o resultado da consulta apresentada na **Listagem 3**, mesmo que devido à pouca utilização dessa base não tenhamos valores na coluna *Buffer size(MB)*.

The screenshot shows the SQL Server Management Studio interface with two tabs: 'Results' and 'Messages'. The 'Results' tab is selected and displays a table of data. The table has four columns: 'ObjectName', 'index_id', 'Buffer size(MB)', and 'Buffer_count'. The data consists of 16 rows, each representing a different object in the database. Most objects have a 'Buffer size(MB)' value of 0, except for 'DatabaseLog' which has a value of 24.

	ObjectName	index_id	Buffer size(MB)	Buffer_count
1	DatabaseLog	0	0	24
2	JobCandidate	1	0	17
3	Product	1	0	15
4	Person	1	0	7
5	ProductCostHistory	1	0	5
6	ProductReview	2	0	5
7	PurchaseOrderDetail	2	0	5
8	vStateProvinceCountryRegion	1	0	4
9	Illustration	1	0	4
10	DatabaseLog	2	0	3
11	WorkOrderRouting	1	0	3
12	WorkOrder	1	0	2
13	ProductCategory	1	0	2
14	PhoneNumberType	1	0	2
15	Location	1	0	2
16	fulltext_indexeddocid_610101214	1	0	2

Figura 2. Resultado da consulta sobre buffer de objetos

Outra importante alternativa para avaliar a possibilidade de realização de um *tuning* é verificar o uso de índices no banco de dados. Quando um índice é criado sobre uma coluna de uma tabela, o otimizador do SQL Server passa a adotá-lo para consultas baseado nas estatísticas coletadas e atualizadas pelo banco. No entanto, o mesmo índice pode pertencer a uma tabela muito utilizada para instruções DML (*Data Manipulation Language*), fazendo com que este índice fique subutilizado para consultas e acabe participando de processos de inserção e atualização de dados, podendo gerar atrasos nestes processos na medida em que diversas linhas inseridas nesta coluna passam por ordenações de dados. Na **Listagem 4** verificamos um exemplo de consulta que identifica índices que são usados com mais frequência para escrita do que para leitura.

Neste caso, a coluna *Difference*, resultante da consulta da **Listagem 4**, irá mostrar em ordem decrescente os índices com maiores diferenças entre leituras e escritas, para análise da real necessidade dos mesmos. Quanto maior a diferença, menos utilizado para leitura o índice está sendo na base de dados e, desta forma, poderá ser reavaliado sobre sua real necessidade junto aos sistemas existentes no banco.

Dicas para a realização de tuning de banco de dados

Ao capturarmos as transações SQL responsáveis por deixar o servidor de banco de dados lento, devemos partir para a fase de

correção e ajustes de desempenho, também chamada de *tuning* de banco de dados. Para que isto ocorra, alguns aspectos são fundamentais para auxiliar um DBA a obter êxito em seu plano de ação, a saber:

- O uso de ferramentas para realizar *tuning* torna o trabalho do DBA mais eficiente, visto que as mesmas avaliam as estatísticas do banco e realizam sugestões para melhoria de rotinas SQL. Desta forma, A Microsoft recomenda a utilização da Database Engine Tuning Advisor (DTA), pois ela pertence ao conjunto de produtos do SQL Server e trabalha de forma contígua com o SQL Profiler. Através da ferramenta DTA é possível verificar índices faltantes empregando o recurso *Execution Plan*, também existente no Management Studio;
- Realizar constante análise sobre os índices e estatísticas das tabelas das bases de dados. Muitas vezes a adição de um índice gera ganhos em torno de 99% para o plano de execução, porém deve-se ter cuidado para não criá-los em excesso;
- Recomenda-se que, antes de realizar o *tuning* de banco de dados diretamente em ambientes de produção, as customizações sejam feitas em bancos de testes, principalmente quando utilizadas as ferramentas SQL Profiler e DTA, bem como outros sistemas para ajustes de desempenho. Estas ferramentas podem sobrecarregar um servidor de produção dependendo dos recursos de *hardware* disponíveis;
- Analisar se existem funções (*functions*) que possam gerar garrafais no desempenho das consultas, como funções na cláusula WHERE;

Listagem 4. Consulta que aponta os índices com maior taxa de escrita do que de leitura.

```
SELECT OBJECT_NAME(s.[object_id]) AS [Table Name],  
       i.name AS [Index Name],  
       i.index_id,  
       user_updates AS [Total Writes],  
       user_seeks + user_scans + user_lookups AS [Total Reads],  
       user_updates - ( user_seeks + user_scans + user_lookups )  
          AS [Difference]  
FROM   sys.dm_db_index_usage_stats AS s WITH ( NOLOCK )  
       INNER JOIN sys.indexes AS i WITH ( NOLOCK )  
              ON s.[object_id] = i.[object_id]  
                 AND i.index_id = s.index_id  
WHERE  OBJECTPROPERTY(s.[object_id], 'IsUserTable') = 1  
       AND s.database_id = DB_ID()  
       AND user_updates > ( user_seeks + user_scans + user_lookups )  
       AND i.index_id > 1  
ORDER BY [Difference] DESC ,  
        [Total Writes] DESC ,  
        [Total Reads] ASC ;
```

- Averiguar os parâmetros de instância do banco de dados, através da procedure *sp_configure* ou da tabela *sys.configurations*, e propor alterações de acordo com o ambiente e seus recursos físicos disponíveis;
- Verificar a lógica das rotinas SQL e propor melhorias dentro das boas práticas da Microsoft.

Conclusão

Como foi possível observar no decorrer do artigo, o SQL Server 2012 oferece poderosas ferramentas para monitoramento de banco de dados, tendo como objetivo principal a realização de *tuning* uma vez identificados os problemas de desempenho. De fato, cabe ao administrador de banco de dados um estudo prático para analisar e identificar possíveis gargalos de desempenho; seja esse estudo feito por ferramentas como Activity Monitor ou System Monitor, através da execução de Dynamic Management Views (DMVs), ou pelas ferramentas SQL Profiler e DTA, ou ainda um combinado entre algumas delas.

Por fim, cabe ressaltar que a realização de *tuning* requer, muitas vezes, paciência e dedicação, pois cada ambiente de banco de dados possui características próprias e devido a isso um problema de performance não se resolve a partir de uma simples “receita de bolo”. Além disso, é necessário que haja um bom entendimento prévio sobre o negócio e muito diálogo com as equipes de desenvolvimento dos sistemas envolvidos, possibilitando assim facilitar a realização de *tuning* da maneira mais assertiva possível.

Você gostou deste artigo?

Dê seu voto em www.devmedia.com.br/sqlmagazine/feedback

Ajude-nos a manter a qualidade da revista!



Autor



Fernando Weschenfelder

fernandoweschenfelder@gmail.com

Atua no ramo de Tecnologia da Informação há mais de 9 anos. É bacharel em Administração de Empresas com ênfase em Análise de Sistemas e pós-graduado em Gestão Estratégica de TI, na PUC-RS. Possui certificações de banco de dados e gestão de TI. Trabalhou também com bancos de dados Oracle e Ingres, além de ter realizado diversos cursos na área. Atua como DBA SQL Server em empresa prestadora de serviços em consultoria de banco de dados, em Porto Alegre (Advanced IT). Participou da criação e execução de grandes projetos de alta disponibilidade, ambientes de alta criticidade e complexidade, na área de banco de dados e sistemas de informações.



Links:

SQL Server 2012 – Tuning e ferramentas de monitoramento.

<http://technet.microsoft.com/pt-br/library/ms189081.aspx>

Dynamic Management Views (DMVs).

<http://technet.microsoft.com/pt-br/library/ms188754.aspx>

Download de base de dados exemplo para SQL Server.

<http://msftdbprodsamples.codeplex.com/releases/view/93587>

Diferentes Wait Types no SQL Server 2012

<http://msdn.microsoft.com/en-us/library/ms179984.aspx>

Instruções DBCC SQL Server 2012

<http://technet.microsoft.com/pt-br/library/ms188796.aspx>

Conhecimento faz diferença!



The magazine cover for issue 28 features a large black robot standing next to a hand holding a small green plant. The title 'Evolução do software' is prominently displayed. Below the title, there are several columns of text and icons related to software engineering topics like 'Definições, preocupações e custo', 'Cuidados a serem tomados na implantação', and 'Projetos'. A large red starburst graphic at the bottom right contains the text '+ de 290 vídeos para assinantes'.

Faça já sua assinatura digital! | www.devmedia.com.br/es

Faça um upgrade em sua carreira

Em um mercado cada vez mais focado em qualidade, ter conhecimentos aprofundados sobre requisitos, metodologia, análises, testes, entre outros, pode ser a diferença entre conquistar ou não uma boa posição profissional. Sabendo disso a DevMedia lançou uma publicação totalmente especializada em Engenharia de Software. Todos os meses você pode encontrar artigos sobre Metodologias Ágeis; Metodologias tradicionais (document driven); ALM (application lifecycle); SOA (aplicações orientadas a serviços); Análise de sistemas; Modelagem; Métricas; Orientação à Objetos; UML; testes e muito mais. **Assine Já!**



DEV MEDIA

Eventos de espera no SQL Server

Analisando eventos de espera com foco em performance do servidor SQL Server

O desempenho das aplicações é tema recorrente em qualquer empresa, independente de seu porte, do tipo de ambiente e do perfil dos softwares que são executados.

Para medir o desempenho da aplicação, temos uma série de ferramentas que nos auxiliam a identificar problemas, minimizar gargalos e consequentemente melhorar continuamente o comportamento de nossos sistemas. Podemos citar como exemplo os contadores do *performance monitor* (perfmon), os trases do próprio SQL Server que nos direcionam para as consultas mais lentas ou que consomem mais recursos e também os Eventos de Espera (Wait Events) do nosso servidor de banco de dados. E é sobre esse último item que falaremos nesse artigo. O que são os eventos de espera? Como coletar essas informações? O que elas nos dizem? Quais eventos devem ser tratados com mais atenção e quais podem ser descartados na minha análise? Responderemos essas perguntas ao decorrer do artigo.

O que são os eventos de espera?

Sempre que o banco de dados aguarda por algum recurso para que um determinado processamento seja concluído, essa informação é registrada. A espera pode ser por CPU, por disco, por escrita, não importa. Qualquer espera para que uma ação seja concluída é armazenada. Esses são os chamados eventos de espera. Analisando essas informações podemos determinar se temos uma grande espera por lock, por exemplo, se temos problemas com paralelismo de consultas ou com escrita ou leitura de disco. Isso nos ajuda no direcionamento dos trabalhos de performance.

Onde ficam essas informações e como acessá-las?

Os eventos de espera são registrados de forma crescente no servidor e só são zerados após uma

Resumo DevMan

Porque este artigo é útil:

Com a análise de informações sobre eventos de espera, é possível identificar gargalos de performance do seu servidor e direcionar os esforços na otimização de consultas. Também é possível traçar perfis de utilização do seu ambiente e identificar qual o melhor aproveitamento das janelas diárias para execução de rotinas batch sem prejudicar o ambiente produtivo. Para entender como isto pode ser feito, este artigo trata da análise dos eventos de espera do seu servidor SQL Server e de como podemos utilizar essas informações em análises de desempenho e performance, direcionando nossos esforços para a otimização de consultas ou até mesmo o upgrade de nossos servidores.

reinicialização do sistema. Dessa forma, uma foto dos eventos de espera do banco de dados de agora é sempre maior que uma de minutos atrás.

Para obter essas informações existem duas maneiras:

- Através do comando DBCC:

```
DBCC sqlperf (waitstats)
```

- Por meio da view DM_OS_WAIT_STATS:

```
select * from sys.dm_os_wait_stats
```

Embora ligeiramente diferentes, os comandos acima retornam as mesmas informações. Devemos, no entanto, manter nosso foco nas seguintes:

- **Wait Type:** Nome do evento de espera;
- **Wait Time:** Tempo acumulado de espera do evento.

Com essas informações temos os eventos e o tempo de espera acumulado desde a última inicialização do serviço de banco de dados.

Todos os eventos são importantes?

É difícil elencar eventos que sempre aparecerão em suas análises. Pensando em priorizar nossa atenção, é mais simples eliminarmos eventos que não têm relação com o tempo de resposta de nossa aplicação. Eventos do tipo Sleep e também Waitfor podem ser desconsiderados imediatamente por não influenciarem o tempo de resposta de nossas aplicações. Além desses, geralmente os eventos de sistema, como checkpoints do banco ou a limpeza da fila de processos são eventos que podem ser desconsiderados da análise. Na **Tabela 1** temos uma relação desses Wait Events.

Outros eventos que podem ser desconsiderados são eventos que ocorrem frequentemente, mas não têm relação com desempenho, como eventos de backup e inicialização e desligamento do servidor.

Como coletá-los?

Executando um dos comandos que listamos anteriormente, é possível ter uma ideia de como os eventos estão acumulados no exato momento da execução. Mas isso não é o bastante para que tenhamos uma visão clara do que acontece no que se refere à espera nos nossos bancos de dados. Para uma análise mais detalhada, recomenda-se uma coleta contínua dessas informações para que possamos ter uma visão linear, tal qual um monitor de desempenho. A maneira mais simples para se fazer isso é utilizando o SQL Server Agent e criando um JOB para que as informações sejam coletadas.

Criando o JOB

Para criar o job, vamos acessar nosso servidor e ir até o SQL Server Agent, que fica na barra lateral esquerda. Expandindo essa seleção, é possível ver a pasta *Jobs*. Com um clique com o botão direito, a opção *New Job...* estará disponível. Basta clicar nessa opção, conforme indica a **Figura 1**.

Feito isto, uma nova tela para a criação do job será disponibilizada (veja a **Figura 2**). Dentre as opções que temos do lado esquerdo, trabalharemos com três delas:

- **General:** onde daremos as informações gerais do job que estamos criando;
- **Steps:** onde configuraremos os passos que nosso job executará;
- **Schedules:** que é onde agendaremos a execução do job.

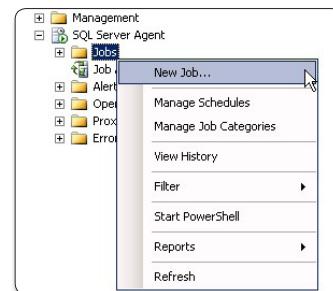


Figura 1. Criando o job

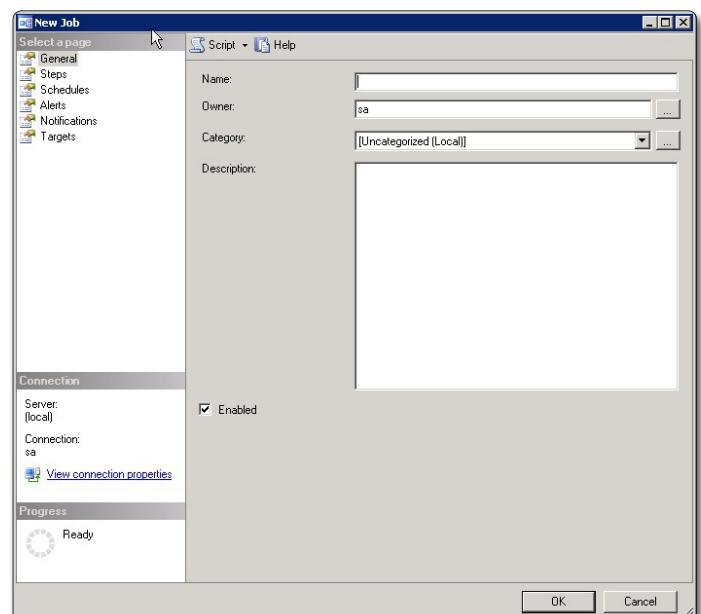


Figura 2. Wizard para criação de novo job

Eventos de espera do tipo *Idle*, que podem ser desconsiderados em uma análise de desempenho

SLEEP	BROKER_TO_FLUSH	DBMIRROR_EVENTS_QUEUE	RESOURCE_QUEUE
WAITFOR	SQLTRACE_INCREMENTAL_FLUSH_SLEEP	ONDEMAND_TASK_QUEUE	SLEEP_BPOOL_FLUSH
KSOURCE_WAKEUP	BROKER_EVENTHANDLER	REQUEST_FOR_DEADLOCK_SEARCH	SLEEP_SYSTEMTASK
SLEEP_BPOOL_FLUSH	BROKER_TRANSMITTER	LOGMGR_QUEUE	SLEEP_TASK
BROKER_TASK_STOP	CHECKPOINT_QUEUE	SLEEP_TASK	SQLTRACE_BUFFER_FLUSH
XE_TIMER_EVENT	CLR_SEMAPHORE	CLR_MANUAL_EVENT	WAITFOR
XE_DISPATCHER_WAIT	DBMIRROR_EVENTS_QUEUE	BROKER_RECEIVE_WAITFOR	PREEMPTIVE_OS_AUTHENTICATION-NOPS
FT_IFTS_SCHEDULER_IDLE_WAIT	DBMIRROR_WORKER_QUEUE	PREEMPTIVE_OS_GETPROCADDRESS	LOGMGR_QUEUE
SQLTRACE_BUFFER_FLUSH	DBMIRRORING_CMD	BAD_PAGE_PROCESS	LAZYWRITER_SLEEP
CLR_AUTO_EVENT	KSOURCE_WAKEUP	OLEDB	CHECKPOINT_QUEUE
BROKER_EVENTHANDLER	BROKER_TRANSMITTER	ONDEMAND_TASK_QUEUE	REQUEST_FOR_DEADLOCK_SEARCH

Tabela 1. Relação de Wait Events

Eventos de espera no SQL Server

A primeira opção, *General*, é onde daremos o nome ao nosso job, e onde podemos cadastrar uma breve descrição das atividades que estão relacionadas a ele (Figura 3). Na tela que é disponibilizada para que informemos as configurações, deveremos informar os seguintes campos:

- **Name:** Nome do job;
- **Owner:** Qual usuário é o dono do job. Recomenda-se que seja um administrador;
- **Description:** Breve descrição dos passos relacionados ao job.

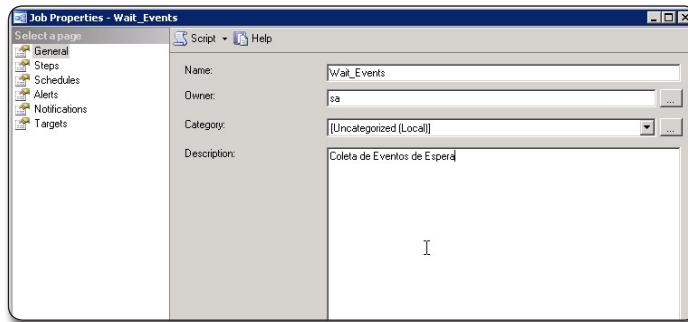


Figura 3. Informações gerais sobre o job

Após essas configurações, passaremos aos passos (*Steps*) que nosso job executará. Steps são os “passos” que nosso job executará. Por exemplo, se temos duas procedures que precisam ser executadas mandatoriamente uma após a outra, adicionamos dois steps, e escrevemos o comando *exec* de cada procedure em um step separado. Dessa maneira, garantimos que a execução do segundo procedimento só será executado após o término do primeiro. Assim também conseguimos analisar o tempo de execução de cada objeto separadamente.

No menu lateral esquerdo, selecionando a opção *Steps*, uma tela com novas opções será mostrada. Nessa nova tela, devemos selecionar a opção *New...* no canto inferior esquerdo (Figura 4). Essa opção criará um novo step.

A tela onde o novo step será criado abrirá com duas opções no menu lateral esquerdo:

- **General:** que é onde vamos inserir o comando a ser executado;
- **Advanced:** que é onde configuraremos o retorno do nosso job.

Na opção *General* da tela de configuração dos *steps*, é necessário que configuremos os seguintes parâmetros (Figura 5):

- **Step name:** Nome do passo;
- **Type:** Tipo de comando a ser executado. No nosso caso, uma instrução *Transact-SQL*;
- **Database:** Em que banco de dados o comando será executado. Como eventos de espera referem-se ao servidor como um todo, não importa em qual base o comando será executado;
- **Command:** Comando que será executado.

Para esse job, utilizaremos o comando *DBCC SQLPERF*. Essa instrução tira uma fotografia dos eventos de espera no momento da sua execução. Veja a Listagem 1.

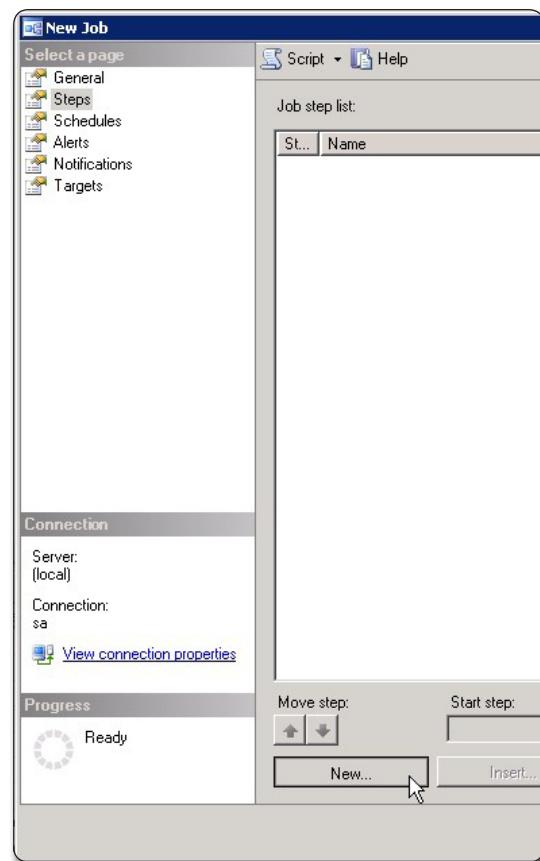


Figura 4. Criando um novo step

Listagem 1. Código do job.

```
set nocount on
declare @run_date datetime
create table #temp (
    timestamp datetime,
    wait_type sysname,
    requests bigint,
    wait_time bigint,
    signal_wait_time bigint)
set @run_date = getdate()
insert into #temp(wait_type, requests, wait_time, signal_wait_time)
exec('dbcc sqlperf (waitstats)')
update #temp set timestamp = @run_date
select * from #temp
drop table #temp
```

Observando esse código, podemos perceber que é criada uma tabela temporária onde são inseridos os dados resultantes do comando *dbcc sqlperf (waitstats)*.

Como o comando *dbcc* não retorna a data em que ele foi executado, para que possamos trabalhar com uma linha do tempo,

adicionamos o horário em que capturamos a informação. Esse tempo é obtido através do trecho `set @run_date = getdate()` que se encontra logo após a criação da tabela temporária. Após a tabela estar pronta, um select é dado para obtermos o *record set*.

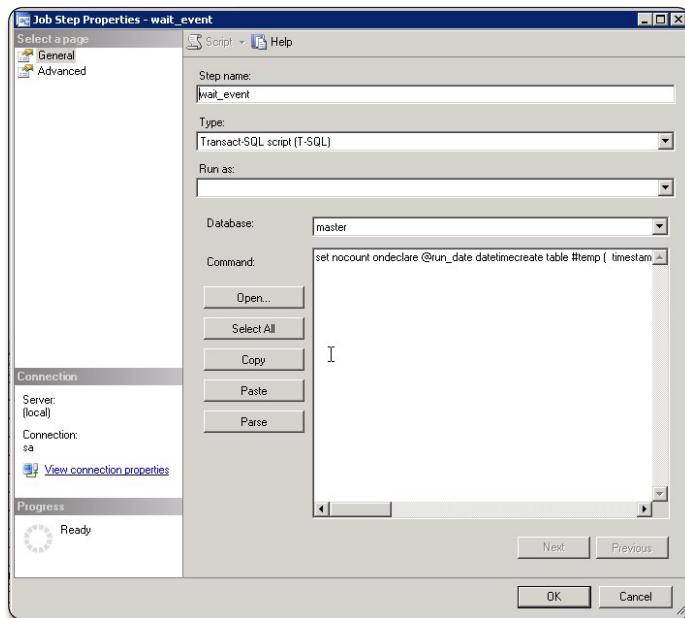


Figura 5. Configurando o step

Para que o *record set* seja gravado em um arquivo de log, utilizamos as opções contidas na página *Advanced* (**Figura 6**). É nessa etapa que direcionamos o resultado das execuções contidas no step. É possível salvar os logs de execução em arquivos e até mesmo em tabelas.

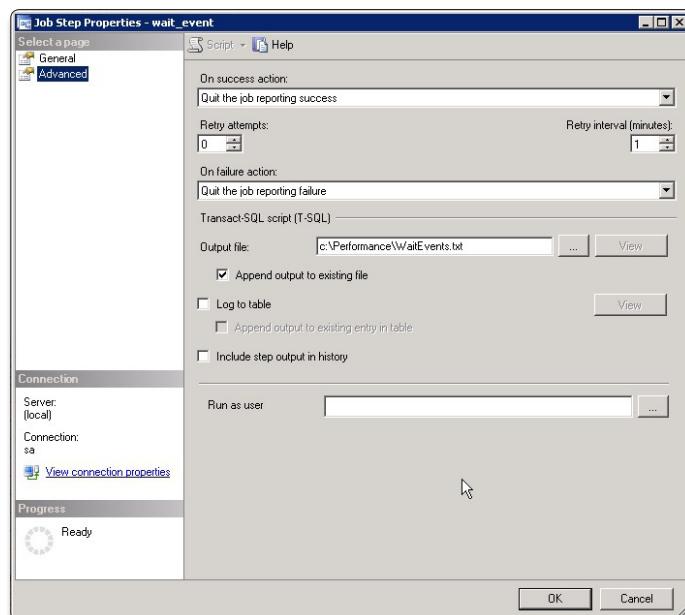


Figura 6. Configurando o arquivo de log

Nessa tela, configuraremos apenas a opção *Output file*, com o arquivo de log. É importante ressaltar que o arquivo já precisa existir, mesmo que vazio. Além disso, o checkbox *Append output to existing file* deve ser marcado. Isso garante que a cada execução, o resultado será adicionado ao final do arquivo, caso contrário será mantido apenas a última execução do *step*.

Com os *steps* configurados, é preciso criar um gatilho que forçará a execução do job. Para isso precisamos definir o intervalo e o período que desejamos que a execução dessas instruções seja realizada. O que garante a execução do job é seu *schedule*, e para configurar essas informações passaremos para a opção *Schedules*, no menu lateral esquerdo, que ao ser acessado apresenta uma nova tela, semelhante à da **Figura 7**. Nessa nova tela, no canto inferior direito, devemos selecionar a opção *New...*

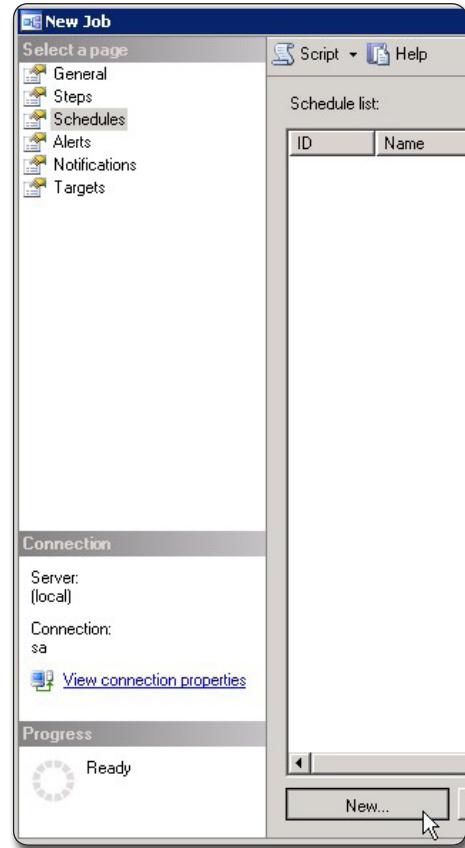


Figura 7. Criando um novo schedule

Quando criamos um novo schedule, uma nova janela com várias opções é exibida (**Figura 8**). Nessa nova janela, configuraremos as opções:

- **Name:** Nome do schedule;
- **Enabled:** A seleção desse checkbox garante a execução do job;
- **Daily Frequency:** Frequência de execução. No caso do exemplo, executaremos os comandos do nosso *step* a cada 10 minutos;
- **Duration:** Pode-se optar por um período pré-determinado de execução. No caso do exemplo, utilizamos a opção *no end date*, que significa que nosso job nunca parará de executar automaticamente.

Eventos de espera no SQL Server

Ao final da tela, em um campo não editável (o campo *Summary*), temos uma descrição por extenso de nosso agendamento para verificação e para que possamos aferir que nossas configurações geraram realmente o *schedule* que desejamos.

Com essas opções configuradas, nosso job está pronto para ser executado, e para isto, basta clicar com o botão direito sobre ele e selecionar a opção *Start Job at step...* (Figura 9). Após isso, o arquivo já pode ser visto carregado no diretório especificado, conforme demonstra a Figura 10.

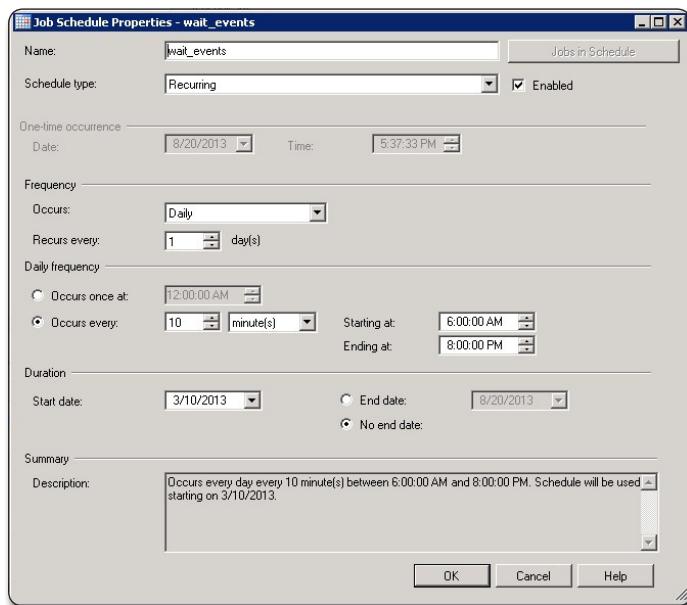


Figura 8. Configurando o agendamento

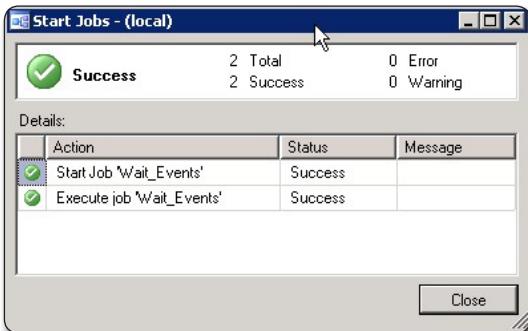


Figura 9. Sumário de execução

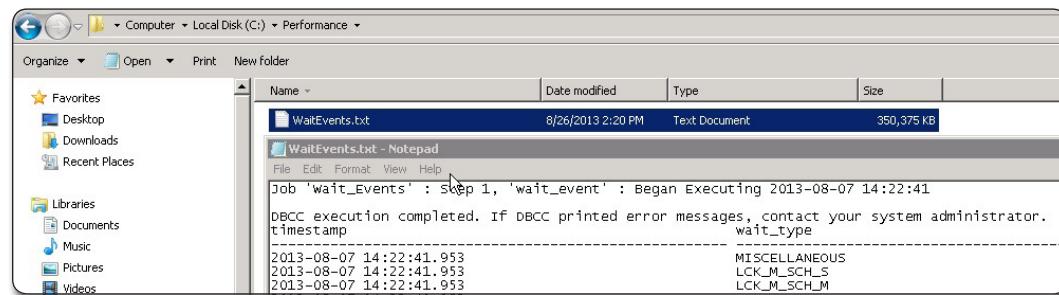


Figura 10. Arquivo gerado

Agora que temos os dados, precisamos transformar essas informações quase ilegíveis em informações que possam ajudar na tomada de decisão.

Analizando as informações

Uma vez que temos os dados em mãos, é necessário tratá-los para que possamos realizar as análises e transformar todos esses dados em informações analíticas. Para interpretar essas informações existem duas abordagens que precisam ser feitas de uma forma ou de outra. São elas, a análise acumulada e a linear.

Análise acumulada

Como comentamos anteriormente, os eventos de espera têm valores acumulados. Isto significa que o último snapshot do nosso job terá os valores maiores que o primeiro, e essa diferença será o volume de espera que tivemos no período em que nossa coleta foi realizada. Então, para identificar quais eventos de espera foram os mais expressivos durante nosso período de coleta, a análise é relativamente simples. Basta isolar em nosso arquivo de log a primeira execução e também a última, e encontrar a diferença entre elas. Dessa maneira, se mantivermos o nosso job ativo por uma semana, subtraindo a primeira foto da última, teremos os números de espera dessa semana de coleta.

Aplicando o Princípio de Pareto (veja o BOX 1), onde definimos os eventos responsáveis por 80% do nosso tempo total de espera, no nosso exemplo obtivemos o gráfico da Figura 11.

BOX 1. Princípio de Pareto

A Lei de Pareto (também conhecido como princípio 80-20), afirma que para muitos fenômenos, 80% das consequências advêm de 20% das causas.

Por exemplo, uma livraria não pode ter todos os títulos do mercado, portanto ela aplica a regra de Pareto e foca em 20% dos títulos que geram 80% da receita.

Trazendo isso para o universo do artigo, devemos focar nos eventos que representam 80% ou mais da nossa espera acumulada.

Essa análise já nos dá uma noção de onde nosso ambiente está perdendo tempo, mas não nos diz, por exemplo, quantos segundos de espera têm em cada evento. Para isso, precisamos de uma análise mais detalhada, como veremos a seguir.

Análise linear

Uma análise mais complexa, mas consequentemente mais completa que a acumulada, é aquela em que traçamos uma linha do tempo dos nossos snapshots. Essa abordagem consiste em subtrair o primeiro snapshot do seguinte, sucessivamente. Assim, se tivermos, por exemplo, 100 fotos no total

da coleta terão 100 pontos em nossa análise, com a variação de um para o outro e não do último para o primeiro. Dessa maneira é possível identificar períodos do dia em que temos picos de espera por determinado evento e relacionar com as consultas executadas naquele momento.

Após realizar o tratamento do nosso arquivo, manualmente ou através de ferramentas ou procedures, teremos as informações dispostas como mostra a Tabela 2.

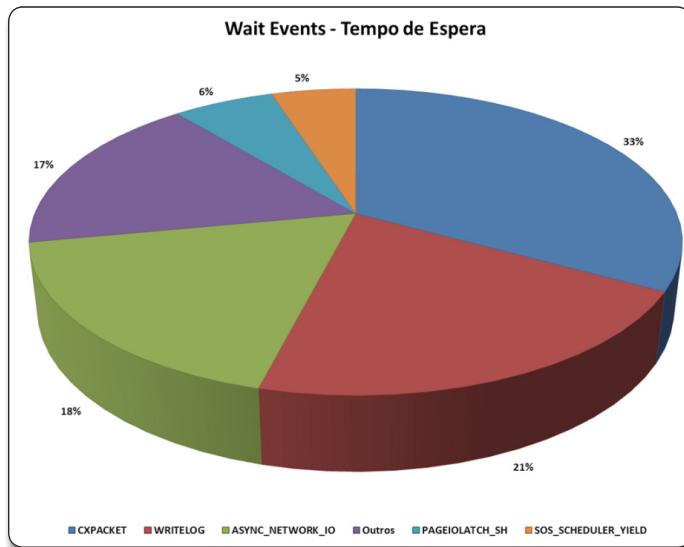


Figura 11. Eventos de espera acumulados

Evento	Momento 1	Momento 2	Momento n...
Cxpacket	10 ms	30 ms	40 ms
Writelog	15 ms	55 ms	150 ms

Tabela 2. Eventos de espera tratados

O número encontrado entre um *snapshot* e outro corresponde ao número em milissegundos de espera por aquele recurso, naquele intervalo. Usando o nosso exemplo de intervalos de 10 minutos, o número encontrado ali corresponde a quanto tempo o servidor perdeu naqueles 10 minutos. Como veremos a seguir, os números podem ser muito grandes.

O gráfico exposto na Figura 12 mostra o comportamento do nosso servidor em um período útil, considerando apenas os eventos que selecionamos no Pareto.

Nele podemos verificar que temos picos de até 3.800 segundos, ou seja, 63 minutos dentro de um intervalo de 10 minutos de coleta. Isso acontece por que o SQL soma aos *Wait events* a espera de todas as transações que estiverem rodando no servidor. Por exemplo, se temos 10 queries executando e cada uma delas tem uma espera de seis segundos por CXPACKET (veremos a descrição desse e de outros eventos posteriormente), o SQL acumulará para esse evento 60 segundos. Se estivermos realizando coletas com intervalos de 5 segundos, do momento 0 segundo para o momento 5 segundos, teremos uma espera

acumulada de 60 segundos. Isso gera esperas exorbitantes, como podemos ver no gráfico da Figura 13.

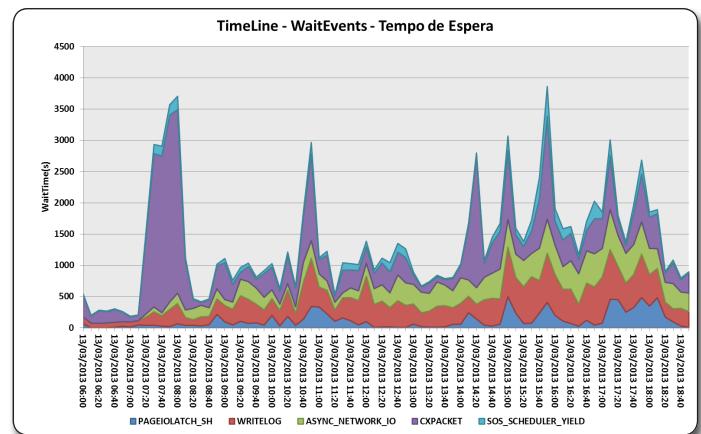


Figura 12. Eventos de espera na linha do tempo

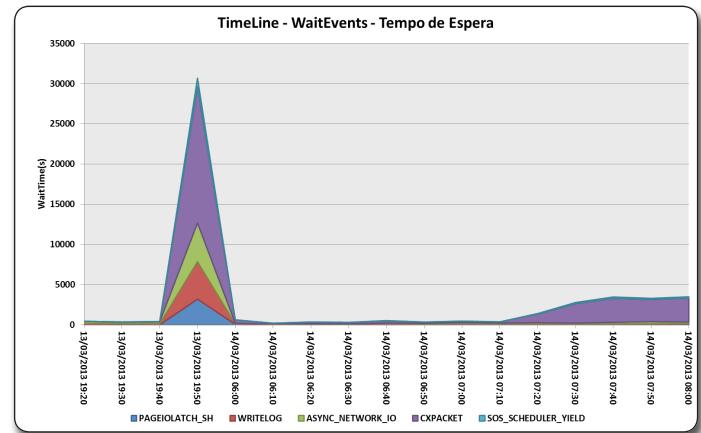


Figura 13. Espera de 500 minutos

Agora que identificamos nossos principais eventos, aqueles que mais agrideem nosso servidor, falaremos um pouco mais sobre eles, sobre o que representam e para onde nos direcionam, e principalmente sobre o que podemos fazer para tentar minimizá-los.

Principais eventos de espera

Ao longo da análise nos deparamos com os eventos de espera. Dissemos o tempo todo que eles direcionam nossa análise e nos dão pistas do que ocorre com nosso servidor e com nossas consultas. Então o que eventos como CXPACKET e WRITLOG estão querendo nos dizer? Vamos direcionar nossa análise para os eventos que encontramos em nossas análises.

WRITLOG

Esse evento nos aponta o tempo que as transações do SQL esperam por escrita no arquivo LDF. É comum encontrarmos um número elevado desse evento em servidores que possuem o arquivo de dados (MDF) no mesmo disco do arquivo de log (LDF). Isso gera uma concorrência por disco e também por tráfego de

rede que pode ser minimizado com a movimentação do arquivo LDF para um disco físico diferente do arquivo MDF.

Se os arquivos já estiverem em discos diferentes e essa espera continuar muito alta, pode-se tentar alterar o modo de recuperação do banco de dados para minimizar a escrita de logs de transação. Se essa for a opção, deve-se tomar cuidado com a frequência de backups, já que o modo de recuperação *simple*, por exemplo, que minimiza a escrita por log, também impossibilita o backup de transaction log. Ficar sem o backup de transação é perigoso e quem toma essa decisão deve estar ciente dos riscos.

CXPACKT

CXPACKT é o evento que registra o tempo que o servidor tem de esperar por consultas que têm o processamento paralelizado.

Quando o SQL calcula o plano de execução de uma determinada consulta, ele pode optar por dividir o processamento dessa instrução nos processadores que ele tem disponíveis. Quando dividida, algumas partes terminam de executar antes das outras, porém enquanto todas não processarem, o record set não é exibido. O tempo de espera entre a primeira e a última parte é registrado como CXPACKT.

Para minimizar o número desse evento, é preciso analisar as consultas que possuem os maiores números de CPU Time (ms). Quanto maior o tempo de processamento, maior a possibilidade de o SQL optar por paralelizar a execução dessa consulta. Quando dizemos que o SQL escolhe se a consulta será ou não paralelizada, estamos dizendo literalmente isso, que o servidor, baseado no plano de execução, opta por paralelizar essa instrução em mais de um processador ou não. Essa definição é feita pelo parâmetro *max degree of parallelism*. Naturalmente ele é configurado com o valor 0, e esse é o sinal de que o SQL pode decidir quando paralelizar uma consulta. O evento CXPACKT pode ser zerado se esse parâmetro for configurado com o valor 1. Isso indicará que o SQL só terá a sua disposição um CPU para a execução de suas consultas.

Embora essa ação zere o tempo de espera por paralelismo, ela pode prejudicar muito os tempos de execução. Portanto, se optar por uma configuração de valor 1, você terá de indicar em cada query da sua aplicação qual será o grau de paralelismo com o *hint OPTION (MAXDOP N)*, onde N é o número de CPUs que o servidor utilizará para paralelizar a execução.

Embora essa opção dê um maior controle sobre as consultas executadas, ela requer um conhecimento enorme da aplicação.

ASYNC_NETWORK_IO

Este evento registra o tempo que o SQL Server espera para que a aplicação trate a informação depois que ela já foi processada pelo banco de dados. Por exemplo: o SQL Server recebeu a solicitação da aplicação, executou a instrução e devolveu o record set. O tempo de espera para que a aplicação trate a informação, indique que a transação foi realizada com sucesso e finalize a transação é registrado como ASYNC_NETWORK_IO.

Como existem situações em que diminuir o record set é impossível, uma relação com o tráfego de rede deve ser feita para que se verifique contenções nesse recurso. Caso as consultas não possam ser diminuídas, um aumento na largura da banda de rede pode ser a solução para minimizar números altos de espera por esse recurso.

SOS_SCHEDULER_YIELD

A espera por SOS_SCHEDULER_YIELD representa o socorro do SQL Server para as threads que rodam no servidor. Literalmente um SOS. O servidor de banco de dados nunca deixa uma thread esperando sem iniciá-la, então imaginemos o seguinte cenário: o banco de dados recebe e começa a executar uma consulta pesada e para melhorar a performance ele paralleliza essa execução em todos os CPUs disponíveis. É nesse momento que se dá a execução de outra consulta. Como a primeira consulta está esperando por CXPACKT, o SQL opta por deixá-la de lado um pouco e iniciar a execução da segunda consulta. O tempo de espera pelo início da execução da consulta 2 e o tempo para que o SQL devolva a atenção para a consulta 1 são registrados como SOS_SCHEDULER_YIELD.

Quanto mais velozes forem suas instruções, menor será esse evento. Portanto, para que ele seja minimizado, devemos voltar nossa atenção para as consultas com maior tempo total de execução (*elapsed time*). Diminuindo o tempo de execução, diminuiremos o tempo de espera.

PAGEIOLATCH_SH

Esse evento está relacionado à espera pela transferência de dados do disco para a memória. Quando executamos uma consulta muito grande, que retorne cargas muito grandes de dados, geralmente não encontramos esses dados no cache. Desse maneira, o SQL Server tem de buscar esses dados em disco e só então devolver o resultado. A espera por essa leitura física é registrada como PAGEIOLATCH_SH. Uma maneira simples de minimizar essa espera é identificar consultas que fazem varreduras totais em tabelas. Identificamos essas situações através dos planos de execução, onde aparecem os operadores *table scans* e *index scans* que fazem a leitura integral da tabela e do índice respectivamente. Minimizando a incidência desses operadores você minimizará a espera por esse evento.

Além desses eventos, uma série de outros pode aparecer em suas análises. Eventos do tipo LCK_M indicam que temos espera por locks em tabelas. Se for esse o caso, procurar por consultas que não perderiam a credibilidade se executadas com a *hint (NOLOCK)* e alterá-las para fazer uso dessa opção pode minimizar tais esperas. Para conhecer a relação completa de eventos de espera, veja a seção **Links** ao final do artigo.

Conclusão

Identificar as consultas mais lentas e tentar atacá-las é uma forma muito eficiente de melhorar o desempenho de nossas aplicações, porém não é a única. Aliar essa análise a outros

fatores além de auxiliar na contextualização do problema pode poupar esforços desnecessários ou evitar que nos voltemos para o lado errado.

Ao longo do artigo conseguimos conhecer melhor uma ferramenta importantíssima no auxílio das análises de desempenho do dia a dia, os eventos de espera. Aprendemos como coletá-los, como transformar esses dados em informações palpáveis e, principalmente, tirar conclusões com essas informações.

Os eventos de espera abordados no artigo são apenas alguns dos inúmeros que temos no nosso banco de dados, e como já citamos, não existe um grupo que seja o mais importante ou que sempre vá aparecer em nossas análises. Cabe ao administrador de banco de dados identificar os que mais prejudicam sua aplicação e dessa maneira tentar atacar o problema, minimizando os seus efeitos. Diminuir o tempo de espera do seu servidor trará consequente ganho de desempenho, o que afinal de contas é o que devemos buscar constantemente.

Autor



Rafael Guidastri

rafa.guidastri@gmail.com

Atua no mercado de tecnologia desde o início de 2005. Analista de Sistemas formado pela UNILINS (Centro Universitário de Lins), trabalha com SQL Server desde a versão 2000, tendo passado por desenvolvimento e administração. Especializado em performance de bancos de dados é atualmente DBA da Athié|Wohnrath, empresa de Arquitetura e Engenharia de São Paulo. Atuou em projetos de migração, administração, performance e troubleshooting em ambientes de altíssima disponibilidade, criticidade e complexidade.



Links:

SQL Server – Wait Events

<http://msdn.microsoft.com/en-us/library/ms179984.aspx>

Você gostou deste artigo?

Dê seu voto em www.devmedia.com.br/sqlmagazine/feedback

Ajude-nos a manter a qualidade da revista!



FÓRUM DEVMEDIA

O lugar perfeito para você ficar por dentro de tudo o que acontece nas tecnologias do mercado atual

No fórum da DevMedia você irá encontrar uma equipe disponível e altamente qualificada com consultores e colaboradores prontos para te ajudar a qualquer hora e sobre qualquer assunto. Temos as salas de Java, .NET, Delphi, Banco de Dados, Engenharia de Software, PHP, Java Script, Web Design, Automação comercial, Ruby on Rails e muito mais!



ACESSE AGORA
www.devmedia.com.br/forum

Monitorando uma instância SQL Server

Alternativas e boas práticas ao monitorar um banco de dados SQL Server

A rotina de qualquer DBA inclui a monitoração de um banco, seja de forma proativa ou para encontrar e descobrir problemas que estejam ocorrendo no ambiente.

Com o passar dos anos, o SQL Server evoluiu neste aspecto, entregando ferramentas cada vez mais completas e intuitivas para a monitoração do banco de dados. Porém, muitas vezes, o impacto que ferramentas como o SQL Server Profiler geram acaba agravando ou criando novos problemas de recursos, caso o servidor não esteja bem dimensionado ou o banco de dados seja extremamente transacional e sensível.

É importante ressaltar que o direcionamento da Microsoft é diminuir cada vez mais o uso do SQL Server Profiler, com a introdução dos *Extended Events* na versão 2008 do SQL Server.

Portanto, serão apresentadas alternativas de monitoração do SQL Server que geram pouco impacto no banco de dados e que, inclusive, podem ser implantadas em longo prazo para a geração de métricas e possibilitar a investigação posterior de eventos.

SQL Trace

O SQL Trace surgiu com o jurássico SQL Server 6.5 como uma forma de monitorar a comunicação cliente-servidor. Com o objetivo de capturar *queries* e *stored procedures*, sua limitação era visível ao tentar monitorar eventos mais complexos, como a aquisição e liberação de *locks*.

SQL Server Profiler

Com o surgimento do SQL Server 7.0, a Microsoft implementou uma forma gráfica e poderosa de capturar diversos tipos de eventos gerados pelo SQL Trace. A **Figura 1** exemplifica a arquitetura do SQL Trace a partir do SQL Server 7.0.

Note que o SQL Server Profiler é apenas uma das diversas maneiras para capturar os eventos internos do banco de dados ou oriundos da comunicação cliente-servidor.

Resumo DevMan

Porque este artigo é útil:

Com o surgimento de novas edições do SQL Server, novas ferramentas foram adicionadas ao arsenal do DBA. Com base nisso, este artigo abordará técnicas de monitoração que surgiram até o SQL Server 2012 e boas práticas para sua implantação. Adotar boas práticas ao gerenciar um banco de dados culmina na monitoração proativa da instância. Ou seja, independente da existência de problemas no SQL Server, deve existir um acompanhamento de desempenho histórico para prever ou corrigir problemas rapidamente.

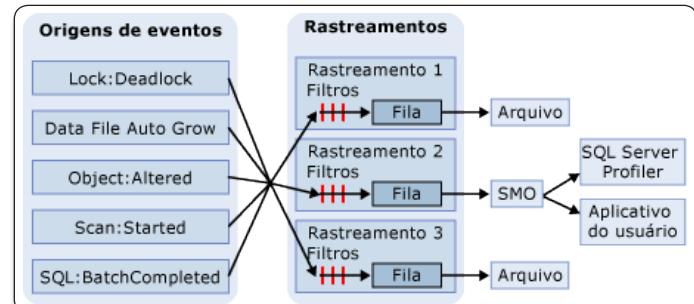


Figura 1. Arquitetura do SQL Trace a partir do SQL Server 7.0

Criando um trace via SQL Server Profiler

Para criar um *Trace* no SQL Server Profiler, devemos proceder da seguinte forma. Primeiro, inicie a ferramenta SQL Server Profiler, e clique em *File > New Trace*. Em seguida, informe o servidor e as credenciais de acesso. O usuário SQL Server deve ser administrador do banco de dados. A partir do SQL Server 2005, a permissão ALTER TRACE tornou esse privilégio mais granular.

As propriedades gerais do Trace são mostradas na **Figura 2**. São apresentados templates com eventos pré-selecionados para diversas situações (Ex: Locks, Tuning). O Trace pode ser nomeado para facilitar a visualização e localização posterior. Pode-se configurar a saída do trace para um arquivo ou tabela. Estas configurações de saída serão úteis ao demonstrarmos as

melhores práticas de um Trace. Por último, podemos definir um horário para que o trace seja interrompido. Na tela da **Figura 2** selecione o template Tuning.

Conforme a **Figura 3**, selecione a aba *Events Selection* e clique em *Column Filters*. Apenas alguns filtros pré-selecionados no Template são exibidos. Informaremos o valor 1000 (valor em milissegundos) no item *Duration / Greater than or equal*.

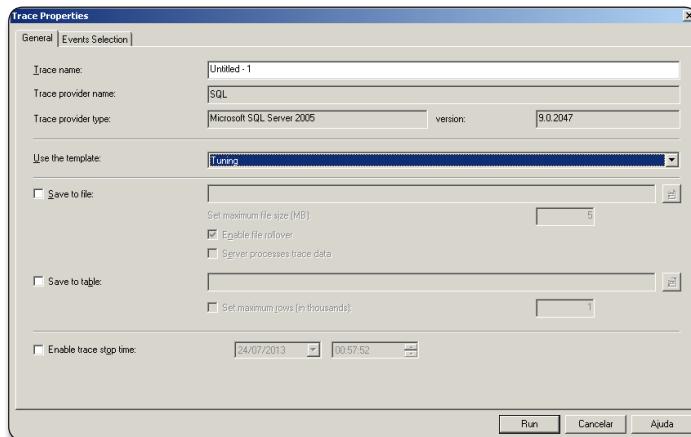


Figura 2. Propriedades gerais do Trace

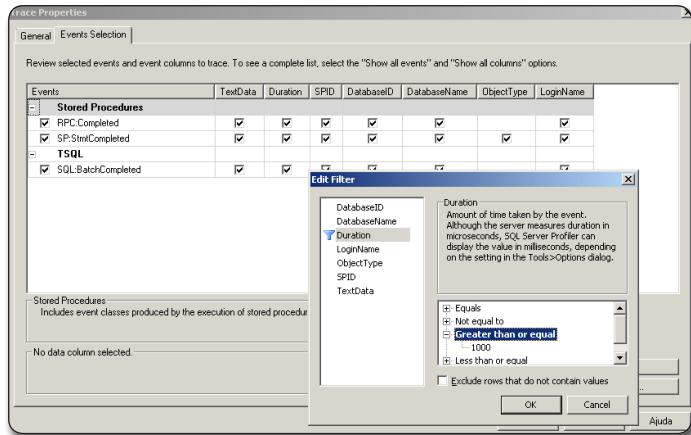


Figura 3. Configurando o Trace

Feito isso, após clicar em *OK/Run*, o trace será iniciado e coletará queries com duração igual ou superior a um segundo.

Criando um Trace via T-SQL

Para criar o mesmo *Trace* via T-SQL, execute o script da **Listagem 1** no servidor destino (ver **BOX 1**). Pode-se observar que esta captura deve ser gravada em uma tabela ou em um arquivo, pois não há uma ferramenta GUI como o SQL Server Profiler para exibir o resultado dos eventos capturados.

Após executar esse script, o último *SELECT* mostrará o ID do Trace criado. Esta variável é utilizada para fechar o arquivo e parar o Trace. No exemplo apresentado na **Listagem 2** estamos encerrando um Trace com ID de valor 2.

BOX 1. Gerando o script T-SQL de um Trace com o SQL Profiler

Um jeito fácil e rápido de gerar o script de um Trace que foi criado visualmente por meio do SQL Server Profiler é navegando em File > Export > Script Trace Definition. A ferramenta criará um arquivo com extensão .sql. Basta editar o valor InsertFileNameHere, inserindo o caminho no qual o Trace será salvo.

Otimizando a performance do SQL Trace

Encontramos diversos estudos na Internet demonstrando empiricamente que uma sessão mal configurada de Trace pode causar até 30% de sobrecarga em um servidor de banco de dados. Em cenários específicos com alta quantidade de transações, muitos usuários e poucos recursos disponíveis, pode causar lentidão e até mesmo o congelamento do sistema.

A seguir, listamos recomendações para mitigar tais riscos ao executar uma sessão de Trace ou SQL Server Profiler:

- **Restritividade ao escolher eventos:** conheça o retorno esperado de cada evento escolhido em seu Trace para não causar sobrecarga por dados em demasia. Por exemplo: em um caso específico, o evento *SP:StmtStarting* pode gerar resultados semelhantes ao *SQL:BatchStarting*, portanto não há necessidade de adicionar os dois contadores;
- **Utilize filtros:** Além de tornar o seu Trace mais legível, os filtros diminuirão a massa de dados a ser gerada, reduzindo o impacto da sessão;
- **Opte por Server Side Traces:** executar um Trace via T-SQL é comprovadamente mais leve que a ferramenta gráfica SQL Server Profiler;

• **Opte pela gravação do Trace em arquivo:** Sempre que possível, opte pela gravação de um arquivo de extensão .trc com os dados coletados. Esta opção consome menos recursos em comparação com o display de informações na ferramenta ou a gravação em tabela. Lembre-se de configurar tamanhos razoavelmente pequenos para o arquivo (até 200 MB), pois desta maneira será mais fácil transportar o arquivo ou abri-lo posteriormente. Esteja atento ao drive de destino do arquivo. É recomendável escolher um drive no qual não existam datafiles e logfiles do SQL Server, evitando a concorrência de I/O;

- Lembre-se de adicionar um lembrete, job ou configurar um tempo de parada para o seu Server Side Trace. Ao criar uma sessão via script, torna-se muito mais fácil esquecer o Trace ativo e consumindo recursos do banco de dados de forma desnecessária;

• Caso deseje utilizar o SQL Server Profiler para um Trace rápido, inicie a ferramenta no próprio servidor: executando localmente, haverá menor tráfego de rede.

Alternativas ao SQL Trace

Nos tópicos seguintes, exploraremos recursos de monitoração sofisticados como Extended Events e o SQL Data Collector. Porém, estas funções surgiram apenas na versão 2008 do SQL Server.

Monitorando uma instância SQL Server

Listagem 1. Script para criação de um SQL Trace.

```
DECLARE @TraceID int,
        @Tamanho bigint,
        @Arquivo nvarchar (256),
        @On bit,
        @Duracao bigint

SELECT @Tamanho = 20
SELECT @Arquivo = 'C:\TraceTeste'
SELECT @On = 1

EXEC sp_trace_create @TraceID output,
2,
-- Configura o trace para criar um novo arquivo ao atingir o tamanho de 20 MB.
@Arquivo, -- Caminho do arquivo sem a extensão .trc.
@Tamanho, -- Tamanho máximo do arquivo em MB.
NULL, -- Data/hora de parada do trace
5 -- Número máximo de arquivos a serem criados no Trace.

-- Seleciona os eventos que serão adicionados ao trace.
EXEC sp_trace_setevent @TraceID, 10, 1, @on
EXEC sp_trace_setevent @TraceID, 10, 3, @on
EXEC sp_trace_setevent @TraceID, 10, 11, @on
EXEC sp_trace_setevent @TraceID, 10, 35, @on
EXEC sp_trace_setevent @TraceID, 10, 12, @on
EXEC sp_trace_setevent @TraceID, 10, 13, @on
EXEC sp_trace_setevent @TraceID, 45, 1, @on
EXEC sp_trace_setevent @TraceID, 45, 3, @on
EXEC sp_trace_setevent @TraceID, 45, 11, @on
EXEC sp_trace_setevent @TraceID, 45, 35, @on
EXEC sp_trace_setevent @TraceID, 45, 12, @on
EXEC sp_trace_setevent @TraceID, 45, 28, @on
EXEC sp_trace_setevent @TraceID, 45, 13, @on
EXEC sp_trace_setevent @TraceID, 12, 1, @on
EXEC sp_trace_setevent @TraceID, 12, 3, @on
EXEC sp_trace_setevent @TraceID, 12, 11, @on
EXEC sp_trace_setevent @TraceID, 12, 35, @on
EXEC sp_trace_setevent @TraceID, 12, 12, @on
EXEC sp_trace_setevent @TraceID, 12, 13, @on

-- Configurando filtro de Duration para 1000000 (em microsegundos).
SELECT @Duracao = 1000000
EXEC sp_trace_setfilter @TraceID, 13, 0, 4, @Duracao

-- Inicia o Trace.
EXEC sp_trace_setstatus @TraceID, 1

-- Mostra o ID do Trace criado.
SELECT @TraceID 'ID do Trace'
```

```
DECLARE @TraceID int,
        @Tamanho bigint,
        @Arquivo nvarchar (256),
        @On bit,
        @Duracao bigint

SELECT @Tamanho = 20
SELECT @Arquivo = 'C:\TraceTeste'
SELECT @On = 1

EXEC sp_trace_create @TraceID output,
2,
-- Configura o trace para criar um novo arquivo ao atingir o tamanho de 20 MB.
@Arquivo, -- Caminho do arquivo sem a extensão .trc.
@Tamanho, -- Tamanho máximo do arquivo em MB.
NULL, -- Data/hora de parada do trace
5 -- Número máximo de arquivos a serem criados no Trace.

-- Quais eventos serão adicionados ao trace.
EXEC sp_trace_setevent @TraceID, 10, 1, @on
EXEC sp_trace_setevent @TraceID, 10, 3, @on
EXEC sp_trace_setevent @TraceID, 10, 11, @on
EXEC sp_trace_setevent @TraceID, 10, 35, @on
EXEC sp_trace_setevent @TraceID, 10, 12, @on
EXEC sp_trace_setevent @TraceID, 10, 13, @on
EXEC sp_trace_setevent @TraceID, 45, 1, @on
EXEC sp_trace_setevent @TraceID, 45, 3, @on
EXEC sp_trace_setevent @TraceID, 45, 11, @on
EXEC sp_trace_setevent @TraceID, 45, 35, @on
EXEC sp_trace_setevent @TraceID, 45, 12, @on
EXEC sp_trace_setevent @TraceID, 45, 28, @on
EXEC sp_trace_setevent @TraceID, 45, 13, @on
EXEC sp_trace_setevent @TraceID, 12, 1, @on
EXEC sp_trace_setevent @TraceID, 12, 3, @on
EXEC sp_trace_setevent @TraceID, 12, 11, @on
EXEC sp_trace_setevent @TraceID, 12, 35, @on
EXEC sp_trace_setevent @TraceID, 12, 12, @on
EXEC sp_trace_setevent @TraceID, 12, 13, @on

-- Configurando filtro de Duration para 1000000 (em microsegundos).
SELECT @Duracao = 1000000
EXEC sp_trace_setfilter @TraceID, 13, 0, 4, @Duracao

-- Inicia o Trace.
EXEC sp_trace_setstatus @TraceID, 1

-- Mostra o ID do Trace criado.
SELECT @TraceID 'ID do Trace'
```

Listagem 2. Finalizando um Trace

```
EXEC sp_trace_setstatus 2, 0
EXEC sp_trace_setstatus 2, 2
```

Em versões anteriores do SQL Server, um pouco de criatividade e trabalho manual deve entrar em cena, caso o DBA almeje implantar uma monitoração que não cause praticamente nenhuma sobrecarga no servidor. Scripts compatíveis com o SQL Server 2000 e 2005 que listam processos lentos e bloqueados podem ser encontrados na Internet e facilmente adaptados para alimentarem uma tabela ou arquivo texto. A seguir são apresentadas algumas sugestões:

- **sp_blocker_pss08:** Criado pela própria Microsoft para suprir esta lacuna, trata-se de um script que tira uma “foto” do banco de dados em períodos pré-determinados, salvando os resultados de queries, locks e conexões em um arquivo texto. Pode ser encontrado no site da empresa (vide seção [Links](#));
- **sp_whoisactive:** Este script, criado pelo DBA Adam Machanic, é conhecido na comunidade por ser sucinto e superior ao comando nativo *sp_who*, além de possuir retro compatibilidade com o SQL Server 2005. Pode ser encontrado no site que está na seção [Links](#);
- **Desenvolver uma rotina customizada:** utilizando tabelas de sistema como a *sysprocesses* (SQL Server 2000) e DMVs como a *sys.dm_exec_requests*, o DBA pode criar um script que atenda

sus necessidades apenas com dados relevantes ao cenário a ser monitorado.

Extended Events

Os Extended Events tornam a captura de informações mais performática e customizável (ver **BOX 2**). Também conhecidos como XEvents, apresentam uma curva de aprendizado mais complexa que o SQL Server Profiler, mas é essencial para qualquer DBA que administre um ambiente SQL Server na versão 2008 ou superior.

O conceito chave dos Extended Events é a flexibilidade: pode-se criar uma sessão englobando qualquer evento, e este evento pode ser armazenado em um ou mais destinos, como uma tabela ou log no Event Viewer.

BOX 2. A evolução dos Extended Events

Implementados na versão 2008 do SQL Server, os Extended Events são mais numerosos a cada edição lançada, confirmando a tendência da Microsoft de abandonar o SQL Server Profiler. Apesar disso, eles ganharam uma interface gráfica apenas na versão 2012, uma deficiência que pode ser suprida na versão anterior com a instalação do complemento “SQL Server 2008 Extended Events Addin”, desenvolvida em código-aberto. Como uma das novidades da próxima versão, são previstos 145 novos eventos para o SQL Server 2014 (codinome Hekaton), totalizando 763 Extended Events.

Os eventos são subdivididos em Packages e Channels. Desta forma, se torna mais fácil localizar o evento desejado para iniciar um rastreamento. A query da **Listagem 3** retorna todos os eventos disponíveis.

Com os eventos selecionados, podemos consultar as colunas disponíveis através da DMV `sys.dm_xe_object_columns`.

Listagem 3. Retornando todos Extended Events disponíveis.

```
SELECT p.name AS package, c.event, k.keyword, c.channel, c.description
FROM
(
    SELECT event_package=o.package_guid, o.description,
    event=c.object_name, channel=v.map_value
    FROM sys.dm_xe_objects o
    LEFT JOIN sys.dm_xe_object_columns c ON o.name = c.object_name
    INNER JOIN sys.dm_xe_map_values v ON c.type_name = v.name
        AND c.column_value = cast(v.map_key AS nvarchar)
    WHERE object_type='event' AND (c.name = 'channel' OR c.name IS NULL)
) o
LEFT JOIN
(
    SELECT event_package=c.object_package_guid, event=c.object_name,
    keyword=v.map_value
    FROM sys.dm_xe_object_columns c INNER JOIN sys.dm_xe_map_values v
    ON c.type_name = v.name AND c.column_value = v.map_key
        AND c.type_package_guid = v.object_package_guid
    INNER JOIN sys.dm_xe_objects o ON o.name = c.object_name
        AND o.package_guid=c.object_package_guid
    WHERE object_type='event' AND c.name = 'keyword'
) k
ON
k.event_package = c.event_package AND (k.event = c.event OR k.event IS NULL)
INNER JOIN sys.dm_xe_packages p ON p.guid=c.event_package
WHERE (p.capabilities IS NULL OR p.capabilities & 1 = 0)
ORDER BY channel, keyword, event
```

Ao escolher o destino da sessão, deve-se ter em mente se o mesmo ocorre de forma síncrona (Ex: Log Event Viewer) ou de forma assíncrona (Ex: Ring Buffer ou arquivo XML). Para efeitos de performance, a gravação assíncrona ocorre em segundo plano e causa menor impacto no banco de dados.

A sessão `system_health` é inicializada automaticamente com o SQL Server. Trata-se de uma espécie de “caixa preta”, que armazena informações de deadlocks (em substituição às Trace Flags 1204 e 1222), sessões com erros de severidade alta, falhas de memória, entre outros eventos. Esta sessão foi aprimorada na versão 2012 do SQL Server, atuando de forma simplificada com a procedure `sp_server_diagnostics`.

Testes mostram que o impacto de um Extended Event pode ser quase nulo, se comparado ao impacto de um SQL Server Profiler (ver **Tabela 1**).

Teste	Requisições/segundo	Duração do teste
Baseline	2345	0:13:35
Trace file	1920	0:16:40
Profiler local	215	2:29:00
Profiler remoto	260	2:03:20
XEvents event_file	2102	0:15:15
XEvents ring_buffer	2073	0:15:40
XEvents streaming	1546	0:16:35

Tabela 1. Teste comparando a performance entre o Trace, Profiler e Extended Events.

Fonte: www.sqlperformance.com

Criando uma sessão XEvent via T-SQL

Na **Listagem 4** é demonstrada a criação de uma sessão que grava queries com duração maior que cinco segundos. Certifique-se de que a conta de serviço do SQL Server possui permissão de gravação e leitura na pasta selecionada.

Criando uma sessão XEvent via Management Studio

Para tornar o processo de criação de uma sessão de Extended Event menos dolorosa, uma interface gráfica foi implantada na versão 2012 do SQL Server.

Conforme expõe a **Figura 4**, ao abrir o item *Management* do banco de dados no SQL Server 2012, pode-se encontrar o subitem *Extended Events*. Em seguida, selecione a opção *New Session*.

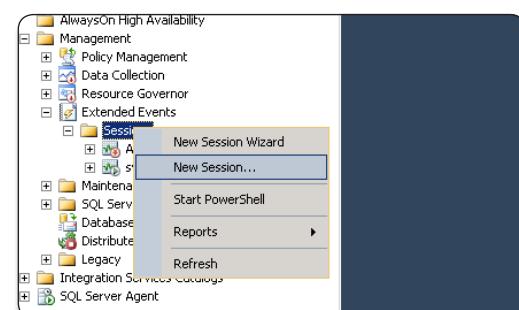


Figura 4. Criando uma nova sessão de Extended Events

Monitorando uma instância SQL Server

A janela (Figura 5) de configuração da Sessão é aberta. Existem alguns templates pré-configurados para facilitar a criação do evento. Como será criada uma nova sessão, selecione a opção <Blank>.

Ao lado esquerdo da janela (Figura 6), selecione o item *Events*. Feito isso, procure pelo mesmo evento que foi utilizado no exemplo anterior: *sql_statement_completed*.

Clicando em *Configure*, a janela exibida na Figura 7 será carregada, possibilitando então selecionar os campos que serão coletados.

Listagem 4. Criando e executando uma sessão de Extended Event.

```
-- Criando a sessão
CREATE EVENT SESSION QueriesLentas
ON SERVER
-- Adicionando o evento sql_statement_completed
ADD EVENT sqlserver.sql_statement_completed
(
    -- Adicionando a ação (campos a serem gravados)
    ACTION (sqlserver.sql_text, sqlserver.tsq_stack)
    -- Duração em milissegundos de queries lentas (5 segundos)
    WHERE sqlserver.sql_statement_completed.duration > 5000
)
-- Adicionando o destino assíncrono (arquivo)
ADD TARGET package0.asynchronous_file_target
SET filename='C:\TEMP\QueriesLentas.xet', metadatafile='C:\TEMP\QueriesLentas.xem'
GO

-- Habilitando a sessão criada acima
ALTER EVENT SESSION QueriesLentas ON SERVER
STATE = START
GO

-- Query de teste
WAITFOR DELAY '00:00:06'
SELECT 'Query lenta'

-- Desabilitando a sessão
ALTER EVENT SESSION QueriesLentas ON SERVER
STATE = STOP
GO

-- Lendo o arquivo XML
SELECT event_data_XML.value('(event/data[1])[1]','VARCHAR(100)') 'DBID',
event_data_XML.value('(event/data[2])[1]','INT') 'OBJECT_ID',
event_data_XML.value('(event/data[3])[1]','INT') 'object_type',
event_data_XML.value('(event/data[4])[1]','INT') 'cpu',
event_data_XML.value('(event/data[5])[1]','INT') 'duracao em microssegundos',
event_data_XML.value('(event/data[6])[1]','INT') 'leituras',
event_data_XML.value('(event/data[7])[1]','INT') 'escritas',
event_data_XML.value('(event/action[1])[1]','VARCHAR(512)') 'texto_sql',
event_data_XML.value('(event/action[2])[1]','VARCHAR(512)') 'xml',
CAST(event_data_XML.value('(event/action[2])[1]','VARCHAR(512)') AS XML),
value('(frame/@handle)[1]','VARCHAR(50)') 'handle'
FROM
(
    SELECT CAST(event_data AS XML) event_data_XML,*
    FROM sys.fn_xe_file_target_read_file
    ('C:\TEMP\QueriesLentas*.xet',
    'C:\TEMP\QueriesLentas*.xem',
    NULL, NULL) T
)
GO

-- Excluindo o evento
DROP EVENT SESSION QueriesLentas
ON SERVER
GO
```

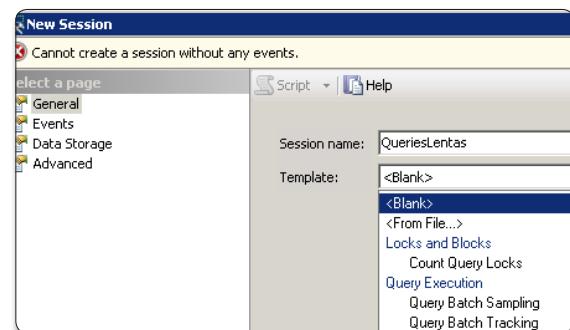


Figura 5. Configurações gerais da sessão

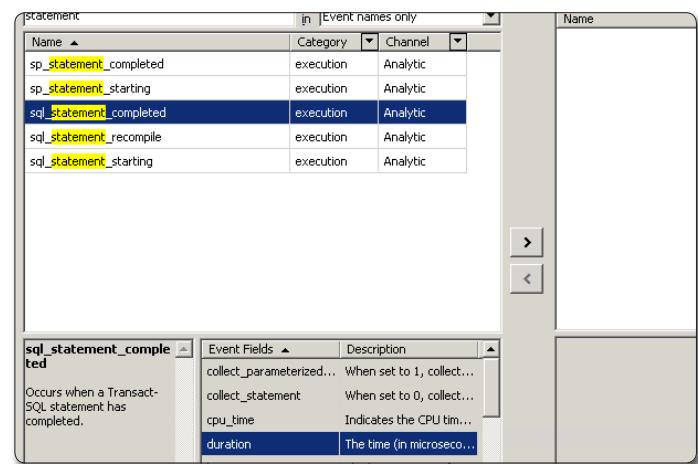


Figura 6. Selecionando o(s) evento(s) que serão utilizados na sessão

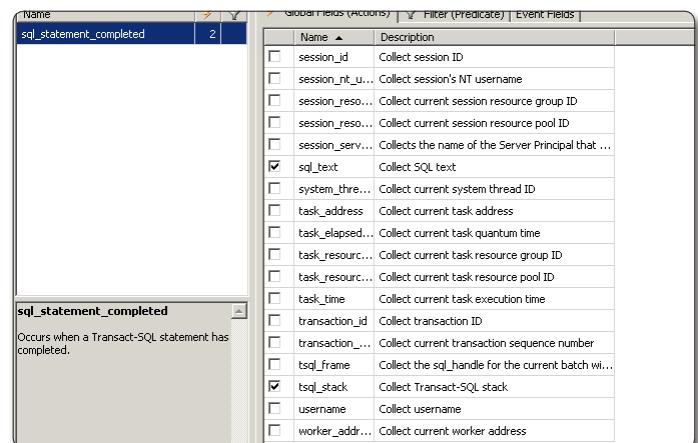


Figura 7. Todos os campos disponíveis para o evento sql_statement_completed

Na aba *Filter* (observe a Figura 8), será feita a configuração do filtro de duração de cada query a ser coletada. Neste caso, maior que 5000 milissegundos.

Na seção *Data Storage* (Figura 9) será configurada a forma de gravação dos resultados da sessão. Seguindo o exemplo anterior, selecione *event_file*. Logo após, nas propriedades do *Target*, configure o caminho do arquivo, tamanho e rollover.

Por fim, na seção *Advanced*, encontramos configurações adicionais, como o uso de memória pelo Extended Event.

Esta customização pode ser interessante, caso seja uma sessão com consumo excessivo de recurso (ex: captura de todas as queries de um determinado usuário). Após fechar a janela de configuração, iniciamos a sessão selecionando *Start Session*, como demonstra a **Figura 10**.

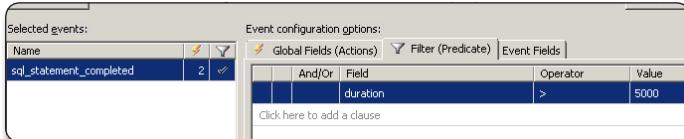


Figura 8. Configurando o predicho

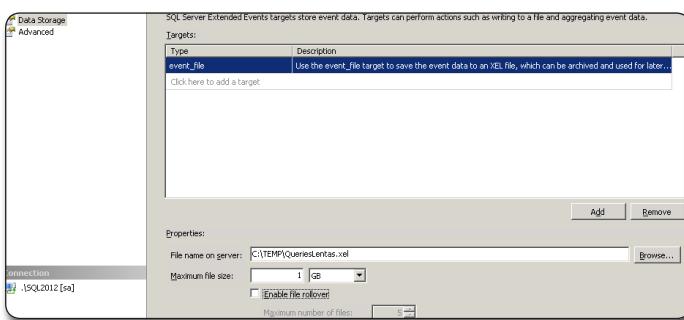


Figura 9. Configurando o(s) Target(s) para armazenar o resultado da sessão

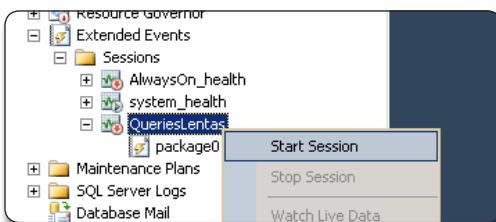


Figura 10. Iniciando a sessão recém-criada

Data Collector

O recurso Data Collector, também conhecido como Data Management Warehouse, surgiu na versão 2008 do SQL Server e é uma poderosa ferramenta para monitoração de ambientes de banco de dados. Esta ferramenta é oferecida a partir da edição Standard e é um DW de baixo impacto, podendo-se fazer um drill-down em diversas informações:

- **Servidor:** Contadores de CPU, rede, memória e I/O (separado por disco);
- **Estatísticas do banco de dados:** Transações/segundo, Logins/segundo, Requisições/segundo, Conexões/segundo, entre outros;
- **Queries:** Queries mais custosas ordenadas por consumo de CPU, consumo de I/O, duração, leituras e escritas;
- **Volume de banco de dados:** Armazena o tamanho do banco de dados, calculando a média de crescimento diário e a tendência de crescimento (negativa, estática ou positiva).

As informações são apresentadas em formato de relatório e a solução pode ser instalada em um servidor central, de forma

que sejam agregadas informações de diversas instâncias em um repositório único.

Sem dúvidas, um dos recursos mais interessantes do Data Collector é o gráfico de barras por período de tempo (apresentado na **Figura 11**) com os diversos tipos de Waits. Esta representação gráfica já é utilizada por diversos softwares comerciais de monitoração do SQL Server e é uma forma prática e intuitiva de analisar um problema de desempenho. Neste exemplo, o gráfico demonstra que a maior parte da espera do SQL Server está em recursos de CPU, e às 15h ocorreu uma pequena espera gerada pela execução de um Backup.

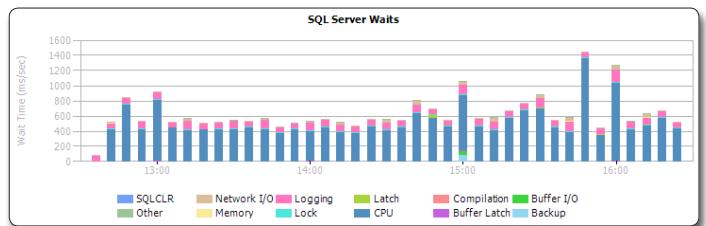


Figura 11. Relatório de SQL Server Waits

Criando um repositório Data Collector

Será demonstrada a criação de uma base Data Collector. Neste exemplo, a mesma instância será tanto o repositório quanto o alvo da monitoração.

1. Como indica a **Figura 12**, conecte na instância que será o repositório utilizando o SQL Server Management Studio. Feito isso, selecione *Data Collection* com o botão direito e clique em *Configure Management Data Warehouse*;

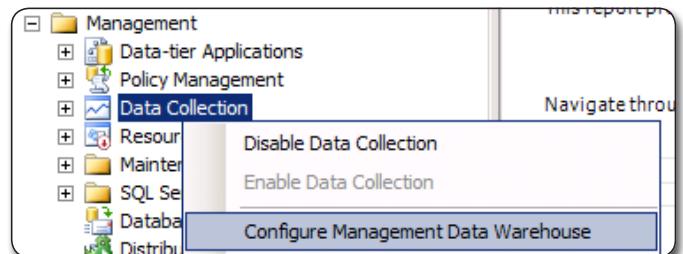


Figura 12. Acessando o painel de configuração do Management Data Warehouse

2. Após avançar a tela inicial do wizard, selecione *Create or upgrade a management data warehouse* para criar o repositório;
3. Na tela seguinte, selecione o banco de dados vazio a ser utilizado para tal função, ou crie um novo banco com um nome intuitivo (Exemplo: MDW);
4. Na janela *Map Logins and Users*, o assistente permitirá a escolha ou criação de um login que possa ser associado às roles *mdw_admin* (Administrador), *mdw_reader* (Leitura de relatórios) ou *mdw_writer*. Caso não deseje relacionar nenhum usuário a estes privilégios, apenas selecione *Next >*;
5. Para encerrar, clique em *Finish*. Nesse momento o SQL Server cria o banco de dados e realiza as configurações necessárias.

Monitorando uma instância SQL Server

Configurando uma instância para o Data Collector

Após criarmos o repositório, configuraremos a mesma instância para que envie dados de monitoração. Este processo deve ser repetido em todos os servidores que o DBA deseja monitorar.

Entre novamente na tela de configuração do Management Data Warehouse. Escolha a segunda opção, *Set Up Data Collection*.

Nesta tela (Figura 13) deve ser informado o servidor de repositório, que neste caso é o mesmo. Atentar a pasta de Cache. É recomendado escolher ou criar um diretório em um drive onde não existam arquivos de dados ou logs do SQL Server, para que não haja concorrência.

Por fim, clique em Finish. Nesse momento o SQL Server cria os Jobs de envio de dados e faz as configurações necessárias.

Ao abrir o item *Data Collection* no SQL Server Management Studio, pode-se visualizar quatro itens de monitoração que foram criados, como demonstra a Figura 14.

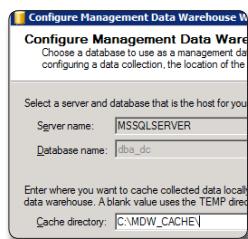


Figura 13. Configurando a instância a ser monitorada

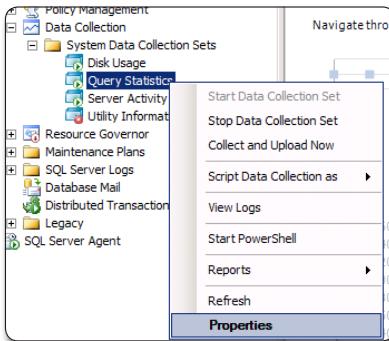


Figura 14. Configurando o contador Query Statistics

Ferramentas comerciais

Encontramos diversas ferramentas no mercado que são destinadas a monitoração de instâncias SQL Server. Entre as mais conhecidas estão o Precise i3 for SQL Server, Quest Spotlight e Ignite Confio.

A grande maioria dessas ferramentas promete gerar um impacto mínimo e serem pouco invasivas, possuindo arquitetura semelhante ao SQL Data Collector, que centraliza o processamento e o armazenamento de dados coletados em um servidor, capturando contadores de inúmeras instâncias.

Trata-se de uma opção interessante para monitorar uma grande quantidade de instâncias SQL Server 2000 e 2005, pois nestas versões encontramos poucas alternativas nativas para monitorar o banco de dados.

Conclusão

Neste artigo, foram apresentadas diversas formas de monitoração de um ambiente SQL Server, priorizando o baixo impacto em servidores de produção. Uma importante função do administrador de banco de dados é estar atualizado sobre novas tecnologias que são apresentadas a cada edição do SQL Server, e os recursos dos Extended Events são vastos e impossíveis de serem dissertados em apenas um artigo.

Cabe ao DBA se aprofundar e dominar esta função que recebe atualizações a cada edição, e que já pode substituir o SQL Server Profiler de forma plena, diminuindo o impacto em bancos de dados.

É importante ressaltar que monitorar um ambiente pode surgir inicialmente de uma necessidade reativa, como a captura de queries lentas e/ou que drenam recursos do servidor. Porém, à medida que estes problemas são sanados, o administrador de banco de dados deve aplicar estes recursos de monitoração de forma proativa, prevendo futuros gargalos e problemas que possam causar indisponibilidade e interrupção dos negócios.

Autor



Stefano Takamatsu Gioia

stgioia@gmail.com

Atua como administrador de banco de dados há mais de 8 anos. É formado em Informática com ênfase em Gestão de Negócios pela FATEC-SP. Possui as certificações OCA MySQL, MCITS SQL Server 2008, MCITPro SQL Server 2008 e MCSA SQL Server 2012. Atua como consultor em banco de dados SQL Server na empresa DBACorp.



Links:

SQL Server 2005 – Introducing SQL Trace

<http://msdn.microsoft.com/en-us/library/ms191006%28v=sql.90%29.aspx>

SQL Server 2008 – Using SQL Server 2008 Extended Events

<http://msdn.microsoft.com/en-us/library/dd822788.aspx>

SQL Server 2008 – SSMS Add-in

<http://extendedeventmanager.codeplex.com/>

SQL Server 2012 – Extended Events

<http://msdn.microsoft.com/en-us/library/bb630282.aspx>

sp_blocker_pss08

<http://support.microsoft.com/kb/271509/pt-br>

sp_whoisactive

<http://www.sqlblog.com>

Você gostou deste artigo?

Dê seu voto em www.devmedia.com.br/sqlmagazine/feedback

Ajude-nos a manter a qualidade da revista!



Por dentro do Oracle Database 12c

Conheça as novidades e quais os novos desafios e paradigmas os DBAs irão enfrentar

Antes de entrar no assunto sobre o Oracle Database 12c, será feita uma breve introdução sobre os conceitos de Cloud Computing e Grid Computing.

O assunto mais falado no momento é a Computação em Nuvem (do inglês, *Cloud Computing*). Existem diversas definições sobre o termo Cloud Computing, mas todas elas são muito parecidas: a utilização de aplicativos e dados através da internet, de qualquer lugar, e em forma de um serviço, com a mesma facilidade de como se estivesse instalada em sua própria máquina. Esta arquitetura é suportada por diversos servidores interligados, estando fisicamente no mesmo local ou em lugares diferentes, numa estrutura onde é possível compartilhar os recursos físicos como: memória, processamento e armazenamento, a fim de garantir alta capacidade de processamento permitindo que os aplicativos sejam disponibilizados através da internet.

A principal característica do Cloud Computing é a utilização de recursos computacionais em forma de serviços, onde aplicativos (ou serviços) e dados podem ser acessados a qualquer momento de qualquer lugar do mundo, de qualquer computador e em qualquer horário através da internet, sem que haja a necessidade de instalar softwares ou armazenar dados localmente no computador. Esta forma de acesso remoto através da internet é que gerou o termo Cloud ou Nuvem.

Nesta arquitetura, todos os dados dos usuários ficam disponíveis na nuvem, e quem fornece a aplicação fica responsável pelas tarefas de administração da aplicação, tais como: desenvolvimento, manutenção, atualização, backups, etc. E um dos benefícios mais interessante para os clientes, é a facilidade de pagar conforme a demanda de utilização destes recursos.

Cloud Computing muitas vezes é confundido com o Grid Computing, o conceito de ambos é muito parecido e as duas tecnologias podem ser implementadas em conjunto. Mas existe uma diferença entre elas, que veremos a seguir.

Resumo DevMan

Porque este artigo é útil:

Neste artigo será apresentada uma introdução sobre o mais novo banco de dados da Oracle, o Oracle Database 12c. Veremos as novidades e melhorias do banco de dados que foi adaptado ao conceito de Cloud Computing, e quais serão os novos desafios e paradigmas para os administradores de banco de dados. Também serão apresentados alguns comandos para realizar tarefas de manutenção nesse banco, além dos caminhos necessários para tirar a certificação desta versão.

Primeiro vamos entender o conceito de Grid Computing. Grid é uma analogia às grades de energia, que disponibilizam a energia em nossas casas, porém, não sabemos exatamente de onde ela vem; apenas a utilizamos plugando os aparelhos na tomada. A partir deste conceito, o Grid oferece alta capacidade de armazenamento e desempenho computacional através de diversos computadores interligados, possibilitando um crescimento escalável, dividindo as tarefas entre diversas máquinas, formando uma máquina virtual para disponibilizar de forma flexível uma camada de recursos lógicos para as aplicações abstraindo os recursos físicos.

A diferença entre estas duas tecnologias está justamente no foco em que elas são aplicadas. No caso de Grid Computing, o foco está na utilização de recursos interligados trabalhando de forma colaborativa para disponibilizar os recursos para as camadas acima. Já o foco do Cloud Computing é reduzir os custos operacionais, através de uma arquitetura virtualizada e escalável com alto poder de processamento, onde as aplicações são disponibilizadas em forma de serviço para os clientes através da internet. Desta forma, podemos notar que a arquitetura do Cloud Computing utiliza também conceitos de Grid Computing através de multiprocessadores e clusters com o objetivo de entregar serviços de alta performance e desempenho.

Para os amantes de sistemas computacionais, esta arquitetura lembra remotamente o início da era da computação, quando se tinha apenas os grandes mainframes, centralizando todo o

processamento e as informações, e disponibilizando através dos “terminais burros”.

A partir desta breve introdução, será apresentado o banco de dados Oracle 12c e entenderemos como esta tecnologia chegou aos bancos de dados.

Oracle Database 12c

A Oracle lançou oficialmente a nova release do banco de dados, o Oracle 12c, em 25 de Junho 2013, trazendo muitas novidades e mudança de paradigmas na administração de banco de dados. De acordo com a Oracle, este é o primeiro banco de dados para a tecnologia Cloud, que possibilita uma redução de custos de TI através da consolidação de diversos bancos de dados em um mesmo servidor, compartilhando as mesmas estruturas de memórias através da tecnologia Multitenant Container Database. Veremos os detalhes de como isso funciona logo mais.

Em um ambiente comum, normalmente se encontra uma ou mais instâncias de bancos de dados na mesma máquina. Nesta arquitetura, cada instância possui as suas próprias estruturas de memória e processos backgrounds, que são gerenciados independentemente. Sendo assim, nos ambientes que tiverem duas ou mais instâncias no mesmo servidor, como cada instância funciona independente uma da outra, teremos estruturas de memória e processos background para cada uma delas, ou seja, repetidos, consumindo muitos recursos do servidor, principalmente a memória.

Através da consolidação de banco de dados, os bancos passam a ser administrados em uma única arquitetura, chamada de Multitenant Container Database (CDB), onde os recursos utilizados pelo Oracle são compartilhados entre vários bancos de dados, inclusive boa parte do dicionário de dados, além dos metadados. Entretanto, os dados de usuários são gerenciados separadamente. Este compartilhamento pode ser realizado através de uma estrutura chamada de Container, que pode ser definido como um repositório onde vários bancos de dados podem ser conectados ou desconectados, de forma transparente para as aplicações. Nesta arquitetura os bancos passam a ser entendidos como “bancos de dados plugáveis” (do inglês *pluggable database*).

Um banco de dados plugável (PDB: *Pluggable Database*) tem as mesmas funcionalidades de um banco de dados convencional que já conhecemos, ou seja, é um conjunto de schemas e objetos que são disponibilizados logicamente para as aplicações, porém, agora com a facilidade de ser armazenado em qualquer lugar. É esta capacidade que permite a portabilidade dos bancos de dados e sua entrada na era do Cloud Computing.

Contudo, ainda é possível utilizar o banco de dados Oracle 12c (versão 12.1) de forma convencional como nas versões anteriores, porém, caso esta opção seja escolhida, não é possível ter ao mesmo tempo os bancos de dados plugáveis na mesma instância. Os bancos de dados convencionais agora são conhecidos como *non-CDB*. O foco deste artigo é explorar as novas

funcionalidades e tendências da Oracle utilizando a arquitetura Multitenant Container Database. A seguir, será apresentado com mais de detalhes os seus componentes.

Arquitetura Multitenant

O assunto mais comentado no momento em fóruns, blogs e artigos na internet, é sobre a arquitetura do banco de dados 12c da Oracle chamada de *Multitenant Container Database (CDB)*, um novo paradigma de banco de dados desenvolvido para o Cloud Computing, através da consolidação de diversos bancos de dados em uma única instância. Mas antes de falarmos sobre o banco de dados, entenderemos o que é a arquitetura Multitenant.

A arquitetura Multitenant nada mais é do que uma única instância de um determinado software, executada em um servidor, com a finalidade de disponibilizar um serviço para vários clientes, onde cada cliente pode ser chamado de tenant (inquilino). Esta arquitetura é um dos principais recursos utilizados no Cloud Computing, e a Oracle incorporou este conceito para o desenvolvimento do banco de dados 12c, mudando o paradigma na administração de banco de dados, adaptando a tecnologia para o Cloud Computing, possibilitando a consolidação de diversos bancos de dados.

Essa nova arquitetura, separa os dados pertencentes aos usuários e aplicativos dos metadados que são utilizados pelo Oracle. Ela é composta por um Multitenant Container Database, onde fazendo analogia com o paradigma atual, nada mais é do que uma grande SGA e processos background que são compartilhados entre vários bancos, possibilitando gerenciar muitos bancos de dados na mesma plataforma. Com isso, é possível “plugar” e “desplugar” facilmente diversos bancos de dados, possibilitando consolidá-los sem a necessidade de alterar as aplicações, gerenciando vários bancos de dados como se fosse apenas um.

Até aqui, muitas novidades foram apresentadas, onde resumidamente os principais componentes dessa nova arquitetura do Oracle 12c são: uma instância CDB (fazendo analogia com o paradigma atual, nada mais é do que uma grande SGA e seus processos background, que são compartilhados entre vários bancos), e um Multitenant Container Database, que é o Container onde os PDBs são conectados, possibilitando gerenciar muitos bancos de dados como se fosse apenas um. A **Figura 1** apresenta estes detalhes.

Para facilitar a compreensão destes conceitos e entender os seus benefícios, veja o seguinte exemplo: um ambiente comum com 100 servidores, e cada servidor possui um banco de dados onde o consumo dos recursos de hardware é em média de 10%, e leve mais 10% de tempo para administração. Um grupo de DBAs deve gerenciar a SGA, processos backgrounds, datafiles, archives, temp, undo, conta de usuários, segurança, etc., para cada um dos bancos individualmente, ao passo que além dos DBAs os administradores de sistemas também deverão gerenciar 100 servidores diferentes. O custo total para tal atividade se torna muito alto e cansativo, além de não ser possível dar a mesma atenção para todos estes ambientes. A **Figura 2** ilustra este cenário.

Mesmo que estes vários bancos de dados fossem agrupados entre os servidores, e a quantidade caísse pela metade, por exemplo, de 100 servidores para 50, onde cada servidor possua duas instâncias, o problema na administração continuaria. Isto por que cada um dos bancos de dados continuará possuindo suas próprias estruturas de memória, processos background, datafiles, etc., que não podem ser compartilhados.

Trazendo estes bancos para um ambiente com o CDB, é necessário apenas criar um container, colocar os 100 bancos de dados dentro dele e gerenciar todos como se fosse apenas um. A Figura 3 mostra como ficaria este cenário CDB.

As aplicações continuarão acessando os bancos de dados normalmente como antes, nada é afetado e não é necessário realizar nenhuma alteração. A Oracle também sugere que sejam definidos dois papéis de administradores, um que seja responsável pelo CDB e outro que seja responsável pelos PDBs.

Não é mais necessário desperdiçar várias horas para realizar instalações e configurações de bancos de dados. Após a criação do CDB, é possível conectar vários bancos de dados neste container com muita facilidade. Também é possível fazer clones dos bancos de dados no próprio container em que ele está conectado ou conectá-lo em outro.

A aplicação de upgrades e patches muitas vezes pode ser uma tarefa exaustiva para o DBA, principalmente se o ambiente tiver dezenas ou até centenas de bancos de dados. Através do CDB muito tempo e esforços podem ser poupadados, pois, a atualização é feita apenas no container e automaticamente é aplicado em todos os bancos de dados que estão conectados a ele (diferente do que ocorre atualmente, quando é necessário aplicar a atualização em cada um dos binários do Oracle). Também é possível atualizar simplesmente desconectando um banco de dados de um container e conectando em outro container que possua as atualizações necessárias.

Os backups e disaster recovery também podem ser realizados no nível do CDB, ou seja, mesmo que existam diversos bancos de dados conectados no mesmo CDB, não é necessário configurá-los de forma individual. As rotinas do RMAN podem ser configuradas para executar no nível do CDB e todos os bancos que estiverem dentro do container serão contemplados. O interessante desta opção é que embora o backup seja feito no nível do container, caso seja necessário fazer o restore de algum dos bancos, não é necessário voltar todos eles, o restore pode ser realizado no nível de banco de dados. Todas estas atividades que foram apresentadas podem ser realizadas utilizando novos comandos SQL, de forma muito simples e em apenas alguns segundos, trazendo um enorme benefício na administração dos backups de vários bancos.

Atualmente as empresas que utilizam Oracle normalmente possuem centenas de bancos de dados espalhados em vários servidores nas mais diversas plataformas, separados entre Produção, Homologação e Desenvolvimento. Devido às melhorias na tecnologia de hardware, principalmente no aumento da quantidade de CPUs, os servidores agora são capazes de processarem uma carga muito maior do que antes. Desta forma, proporciona muitas vantagens na utilização das novas funcionalidades apresentadas no banco de dados Oracle 12c com a arquitetura Multitenant para esta geração de computação em nuvem.

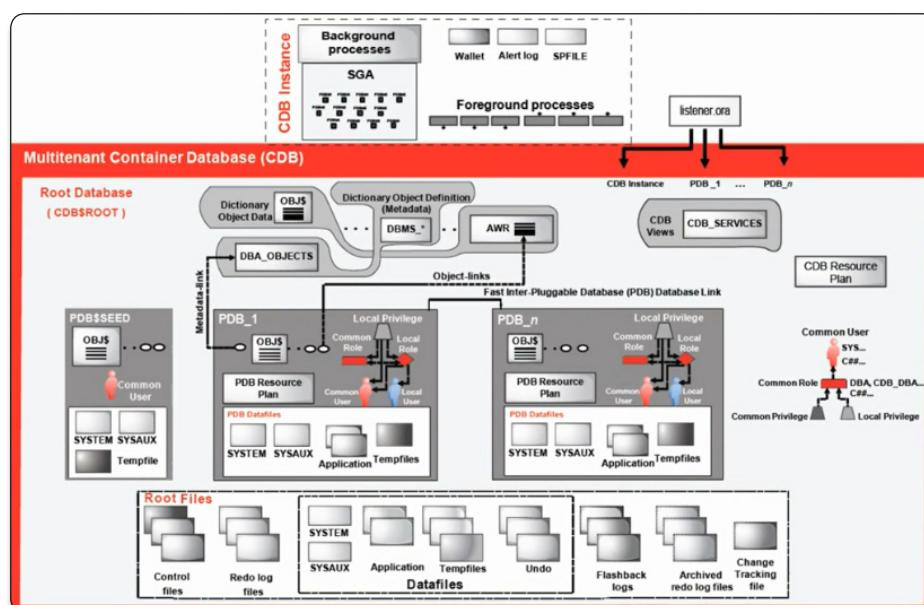


Figura 1. Arquitetura do Oracle Multitenant

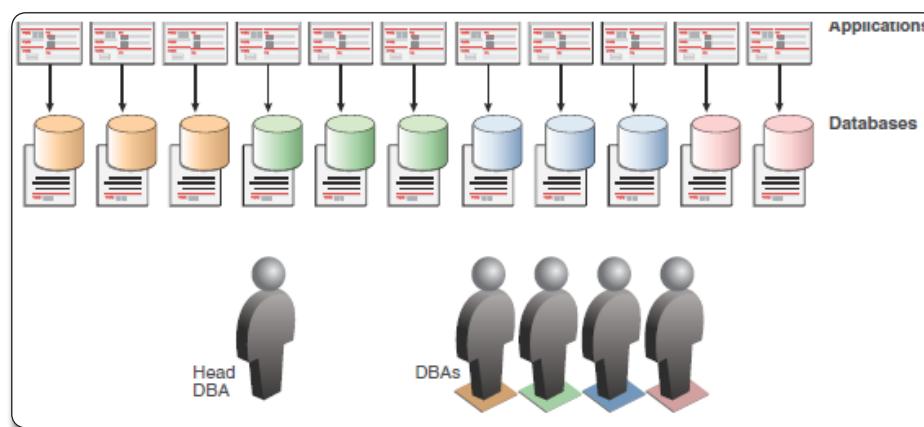


Figura 2. Ambiente antes da consolidação dos bancos de dados

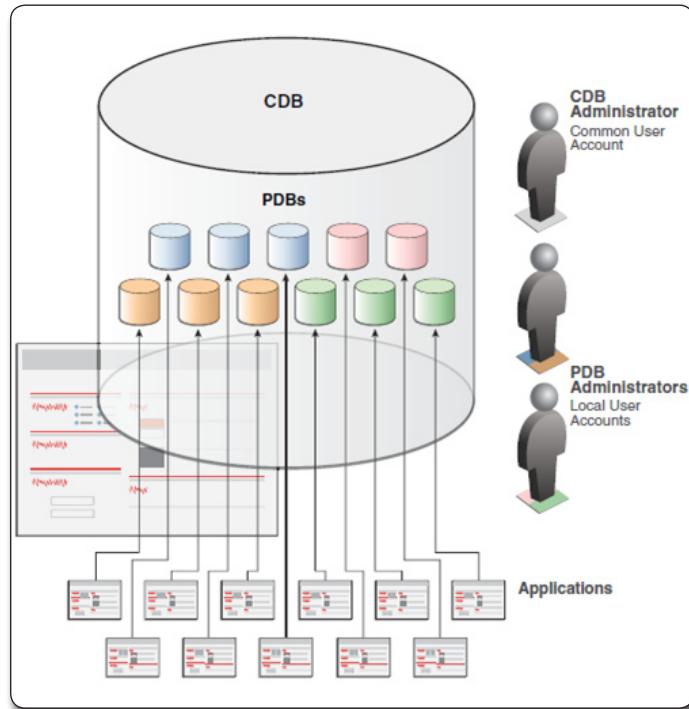


Figura 3. Ambiente após a consolidação dos bancos de dados

A consolidação de vários bancos de dados físicos de diferentes servidores em uma única instância e em um único servidor, utilizando o CDB, traz muitos benefícios, tais como:

- Redução de custo. Através da consolidação de hardware e o compartilhamento dos processos em memória e arquivos dos bancos de dados, traz reduções de custo consideráveis nos recursos de hardware, storage, disponibilidade, manutenções, administração;
- Facilidade e rapidez na movimentação de dados e códigos, desconectando e conectando rapidamente o PDB de um CDB para outro;
- Facilidade no gerenciamento e monitoração dos bancos de dados físicos, sendo apenas um, ao invés de dezenas ou centenas de bancos individuais. Simplifica também as estratégias de backup e disaster recovery;
- Separação de dados e código. Embora os bancos de dados estejam consolidados em uma única instância, os PDBs se comportam como um banco convencional, e as operações são realizadas sem afetar os outros PDBs;
- Separação de atividades entre o administrador do CDB (que gerencia o CDB e pode acessar os outros PDBs ao qual tem privilégios de acesso) e o administrador dos PDBs, que utiliza uma conta local para gerenciar os PDBs de forma individual, sendo que, um usuário local de um PDB não tem privilégios para acessar outro PDB, mesmo que estejam dentro do mesmo CDB;
- Facilidade na realização de Performance e Tuning porque é muito mais simples analisar a performance e métricas para um único banco do que para vários. Por exemplo, é mais simples realizar alterações em uma SGA do que em 100.

CDB Instance

A instance não é muito diferente do que já é conhecido e é constituída por: SGA, PGA, Background Processes, Foreground Processes, files: alertlog, spfile, wallet. A diferença é que agora os processos de memória da instância são compartilhados por vários bancos de dados, onde o buffer cache contém uma referência para cada bloco correspondente a cada PDB id que o bloco pertence (o mesmo vale para os child cursors).

Multitenant Container Database(CDB)

Um Multitenant Container Database (CDB) é um recipiente para múltiplos bancos de dados, onde através da opção Multitenant é possível hospedar múltiplos bancos de dados plugáveis (*Pluggable Database - PDB*), os PDBs também são chamados de containers, ou seja, dentro do CDB cada PDB é considerado um container independente.

Quando um CDB é criado, automaticamente também é criado um container chamado de root (CDB\$ROOT) e um container chamado de semente (PDB\$SEED), conforme mostra a **Figura 4**.

A partir desse ponto é possível criar e adicionar novos PDBs dentro do CDB. Do ponto de vista físico, cada PDB irá possuir as suas próprias tablespaces de dados (incluindo a SYSTEM e a SYSAUX) e seus respectivos datafiles, como pode ser visualizado na **Figura 5**, com dois bancos de dados: hrpdb e salespdbl.

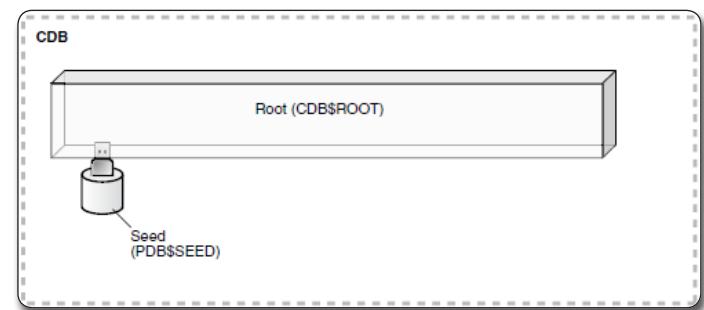


Figura 4. Componentes de um CDB após a sua criação

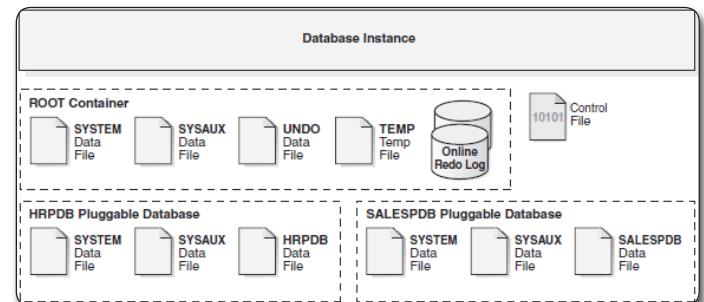


Figura 5. Arquitetura física de um CDB

A seguir são apresentados mais detalhes sobre cada um destes componentes:

- Um *root container* chamado de **CDB\$ROOT**, onde é armazenado os metadados e os dados de usuários comuns. É este container que contém os objetos utilizados pelo Oracle para administração dos bancos de dados e todos os PDBs pertencentes a ele;

- Um banco de dados modelo chamado de PDB\$SEED. Através dele é possível criar facilmente os outros bancos de dados plugáveis;
- Os redo log files são compartilhados em todo o CDB. As informações que são gravadas no redo recebem uma identificação de qual PDB ela pertence. O Oracle Golden Gate já está preparado para entender este formato do redo, quando utilizada a opção CDB. Desta forma, além dos redos, todos os PDBs compartilham também os mesmos archives;
- Os control files são compartilhados em todo o CDB e são atualizados conforme novas tablespaces e datafiles dos PDBs são adicionados;
- As tablespaces de UNDO são compartilhadas em todo o CDB;
- Uma tablespace temporária comum é necessária em todo o CDB. Porém, cada PDB pode ter a sua própria tablespace temporária para os seus próprios usuários locais;
- Cada PDB tem o seu próprio dicionário de dados, armazenado na sua própria tablespace SYSTEM contendo seus próprios metadados e a sua tablespace SYSAUX. As tablespaces podem ser criadas dentro dos PDBs conforme a necessidade da aplicação e cada datafile está associado com um único PDB.

É possível conectar um banco de dados convencional, ou seja, que não está relacionado com qualquer CDB, e transformá-lo em um PDB. Também é possível fazer o contrário, transformar um PDB em um banco de dados convencional, neste caso, é necessário utilizar o Oracle Data Pump.

A quantidade de bancos que podem ser conectados a um CDB é limitado, podendo ter até 252 PDBs, e um PDB\$SEED. Todos os containers possuem um identificador chamado de CON_ID, começando do zero até o limite máximo. Os três primeiros containers são defaults, como mostra a **Figura 6**.

Container ID	Rows pertain to
0	Whole CDB, or non-CDB
1	CDB\$ROOT
2	PDB\$SEED
All Other IDs	User-Created PDBs

Figura 6. Containers ID

Os usuários clients devem conectar nos PDBs através dos serviços do listener. Quando um PDB é criado, automaticamente um serviço também é criado e inicializado dentro do CDB associado a ele. Os serviços podem ser vistos através das views do CDB_SERVICES, e possui limite de até 512 serviços por todos os CDBs. A **Figura 7** mostra usuários distintos se conectando em bancos diferentes através dos serviços do listener.

Outra novidade é referente aos tipos de views do dicionário de dados utilizadas para verificar os objetos existentes no banco de dados. A **Figura 8** mostra a relação destas views.

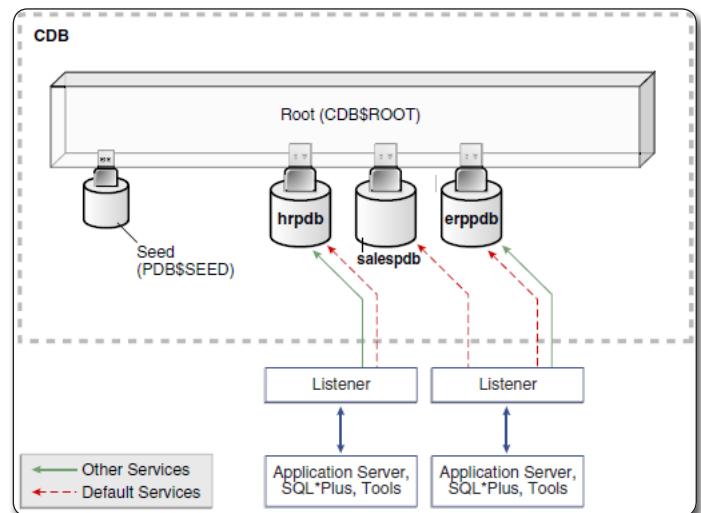


Figura 7. Usuários se conectando nos PDBs através do listener

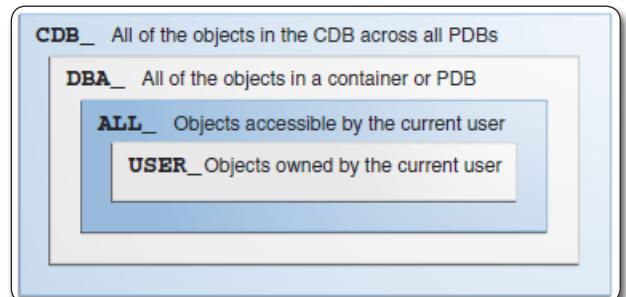


Figura 8. Usuários se conectando nos PDBs através do listener

Pluggable Database – PDB

Um Pluggable Database (PDB), como já vimos, possui a mesma estrutura de schemas e objetos como era nas versões anteriores, mas com a facilidade de poder conectá-los ou desconectá-los em diferentes CDBs, introduzindo um recurso de portabilidade de banco de dados. Embora não seja um conceito novo, e uma técnica muito similar ao attach/detach do Microsoft SQL Server, a Oracle acredita que pode fazer esta portabilidade melhor que os outros bancos de dados.

Usuários e novos privilégios de sistema

A forma como os usuários e os privilégios do banco de dados são administrados mudou um pouco com relação às versões anteriores. Agora existem dois grupos de usuários que são chamados de: Usuários Comuns (*COMMON USERS*) e Usuários Locais (*LOCAL USERS*).

Usuários comuns

Um usuário comum é um usuário criado no root container e automaticamente existirá também em todos os PDBs existentes e nos que serão criados futuramente. Cada usuário comum pode se conectar no root container e realizar operações normalmente, além de também poder acessar os PDBs aos quais possuem privilégios de acesso. Os usuários SYS e SYSTEM são usuários

comuns fornecidos pelo Oracle e continuam com o mesmo nome. Os usuários que são criados como usuário comum devem começar com a nomenclatura “C##” ou “c##”, por exemplo, um usuário comum pode se chamar “c##dba”.

Usuários Locais

Um usuário local é um usuário criado dentro de um único PDB e só pode realizar atividades dentro dele. Não pode ser criado no root container e também não pode se logar nele ou em qualquer outro PDB. O nome do usuário comum deve ser privado e único em cada PDB, e não devem começar com a nomenclatura “C##” ou “c##” de usuários comuns.

Para facilitar a compreensão, a **Figura 9** apresenta um cenário de como funciona estes acessos.

Na **Tabela 1** está à relação entre os bancos de dados e os usuários comuns e usuários locais.

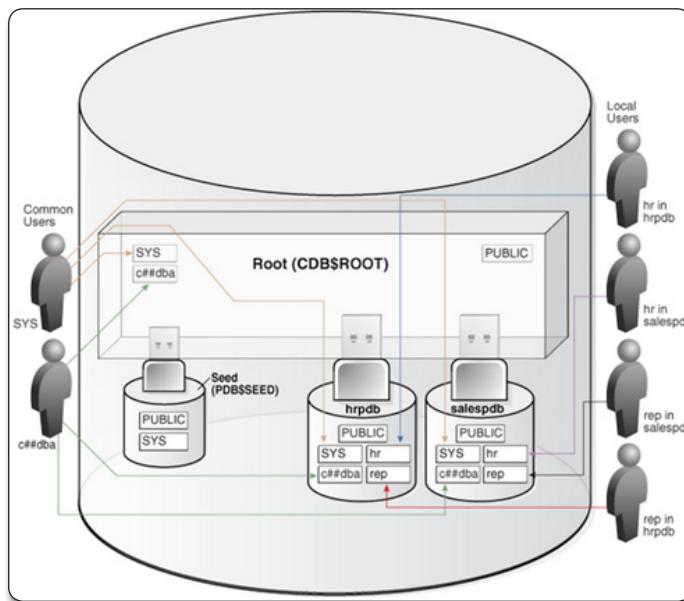


Figura 9. Arquitetura física de um CDB

	Usuários Comuns	Usuários Locais
Root (CDB\$ROOT)	SYS, c##dba	-
hrpdb	SYS, c##dba	hr, rep
salespdb	SYS, c##dba	hr, rep

Tabela 1. Bancos de dados X Usuários

Perceba que os usuários comuns **SYS** e **c##dba** são os mesmos em todos os bancos. O **SYS**, especificamente, é default do Oracle e já é criado automaticamente, entretanto o **c##dba**, além de ter sido criado como usuário comum também foi concedido a ele privilégios de acesso nos bancos **hrpdb** e **salespdb**, como mostrado na **Figura 9**.

No caso dos usuários locais, existem dois com o mesmo nome em bancos de dados diferentes: **hr** e **rep**; entretanto, cada um deles

são independentes um do outro, ou seja, são usuários completamente diferentes. Sendo assim, os usuários **hr** e **rep** são privados e pertencem apenas ao banco **salespdb**. E os usuários **hr** e **rep** são privados e pertencem apenas ao banco **hrpdb**.

Para ilustrar este cenário, na **Listagem 1** está uma demonstração da criação de um usuário no banco de dados **hrpdb** e após a criação, este usuário tenta se conectar no **salespdb**. Vejam o que ocorre.

Este erro ocorre porque o usuário **oracle** foi criado localmente no dentro do **hrpdb** e portanto só tem acesso a ele. Para ter acesso ao **salespdb** é necessário se conectar neste banco com um usuário privilegiado e também criar um usuário local para ele, conforme a **Listagem 2**.

Listagem 1. Criar usuário no hrpdb e tentar acessar o salespdb.

```
SQL> CONNECT SYSTEM@hrpdb
Enter password: *****
Connected.

SQL> CREATE USER oracle IDENTIFIED BY passwd;
User created.

SQL> GRANT CREATE SESSION TO oracle;
Grant succeeded.

SQL> CONNECT oracle@saledspdb
Enter password: *****
ERROR:
ORA-01017: invalid username/password; logon denied
```

Listagem 2. Criar usuário para acessar o salespdb.

```
SQL> CONNECT SYSTEM@saledspdb
Enter password: *****
Connected.

SQL> CREATE USER oracle IDENTIFIED BY passwd;
User created.

SQL> GRANT CREATE SESSION TO oracle;
Grant succeeded.

SQL> CONNECT oracle@saledspdb
Enter password: *****
Connected.
```

Roles Comuns e Roles Locais

As roles funcionam da mesma forma que os usuários, ou seja, a role comum é criada no root container e automaticamente existirá também em todos os PDBs existentes e nos que serão criados futuramente. Enquanto isso, a role local é criada dentro de um único PDB onde os seus privilégios só são válidos dentro dele.

De modo geral, podemos concluir que usuários e roles comuns são aqueles que têm efeito dentro do root container e em todos os PDB's existentes e futuros. Já os usuários e roles locais têm efeito apenas dentro do PDB no qual foram criados.

Para fixar estes novos conceitos, será dado um exemplo que demonstra os efeitos dos grants de roles e privilégios a partir do container root (CDB\$ROOT). Na **Tabela 2** temos o usuário SYSTEM se conectando no root container, criando o usuário comum c##dba e a role c##admin, e concedendo alguns privilégios, e por último atribuindo ao usuário c##dba a role c##admin.

	SQL> CONNECT SYSTEM@root
T1	Enter password: ***** Connected.
T2	SQL> CREATE USER c##dba IDENTIFIED BY password CONTAINER=ALL;
T3	SQL> GRANT CREATE SESSION TO c##dba;
T4	SQL> CREATE ROLE c##admin CONTAINER=ALL;
T5	SQL> GRANT SELECT ANY TABLE TO c##admin; Grant succeeded.
T6	SQL> GRANT c##admin TO c##dba; SQL> EXIT;

Tabela 2. Cenário – Parte 1

Observe os comandos destacados em vermelho com a cláusula CONTAINER=ALL, indicando que o comando será afetado no nível de todo o container. Esta cláusula é muito importante para o entendimento deste exemplo.

Os comandos na **Tabela 3** são executados no hrpdb.

Perceba que o usuário c##dba mesmo tendo sido criado no root container, não conseguiu acessar o hrpdb devido ao grant de CREATE SESSION (T3) não possuir a cláusula CONTAINER=ALL. Logo, o privilégio foi concedido apenas no container local, ou seja, no root e, por isso, ocorreu o erro ORA-01045 na tentativa de conexão. Portanto, para que o grant seja concedido em todos os containers (PDBs) é necessário informar explicitamente a cláusula CONTAINER=ALL, ou também, como apresentado nos comandos T8 e T9.

Mas isto ainda não é o suficiente para ele ter acesso às tabelas dentro do PDB, o que retornou o erro ORA-00942, ou seja, o usuário não possui privilégios nas tabelas deste banco, pois o grant que foi concedido no t5 é restrito ao root container e não se aplica no hrpdb. Desta forma, para que o usuário c##admin consiga ter acesso às tabelas do banco, é necessário executar os grants informando a cláusula CONTAINER=ALL, conforme a **Tabela 4**.

Somente após as permissões de conexão com o banco e os privilégios concedidos, é que o usuário conseguirá se conectar no banco e realizar consultas nas tabelas. Veja a **Tabela 5**.

Gerenciando bancos de dados Plugáveis (PDBs)

A criação de um PDB pode ser realizada utilizando a ferramenta gráfica DBCA ou através da linha de comandos do *sqlplus*.

T7	SQL> CONNECT c##dba@hrpdb Enter password: ***** ERROR: ORA-01045: user c##dba lacks CREATE SESSION privilege; logon denied
T8	SQL> CONNECT SYSTEM@hrpdb Enter password: ***** Connected.
T9	SQL> GRANT CONNECT,RESOURCE TO c##dba; Grant succeeded. SQL> EXIT
T10	SQL> CONNECT c##dba@hrpdb Enter password: ***** Connected.
T11	SQL> SELECT COUNT(*) FROM hr.employees; select * from hr.employees * ERROR at line 1: ORA-00942: table or view does not exist

Tabela 3. Cenário – Parte 2

T12	SQL> CONNECT SYSTEM@root Enter password: ***** Connected.
T13	SQL> GRANT SELECT ANY TABLE TO c##admin CONTAINER=ALL; Grant succeeded.
T14	SQL> GRANT c##admin TO c##dba CONTAINER=ALL; Grant succeeded.

Tabela 4. Cenário – Parte 3

T15	SQL> SELECT COUNT(*) FROM hr.employees; COUNT(*) ----- 107
-----	---

Tabela 5. Cenário – Parte 4

As **Listagens 3 a 6** demonstram como criar, clonar, remover e desconectar um banco de dados de um container CDB utilizando a linha de comando.

Conectar um PDB dentro do container CDB é similar à sua criação (veja a **Listagem 7**). A primeira coisa que deve ser realizada é verificar se o PDB é compatível com o CDB ao qual ele será conectado, executando a função DBMS_PDB.CHECK_PLUG_COMPATIBILITY, passando como parâmetro o arquivo XML e o nome do PDB que será criado no container.

Por dentro do Oracle Database 12c

Listagem 3. Criar um PDB.

```
SQL> CONNECT / AS SYSDBA  
Connected.  
  
CREATE PLUGGABLE DATABASE pdb01 ADMIN USER pdb01_adm IDENTIFIED BY  
passwd01;  
  
ALTER PLUGGABLE DATABASE pdb01 OPEN READ WRITE;
```

Listagem 4. Clonar um PDB.

```
ALTER PLUGGABLE DATABASE pdb01 CLOSE IMMEDIATE;  
  
ALTER PLUGGABLE DATABASE pdb01 OPEN READ ONLY;  
  
CREATE PLUGGABLE DATABASE pdb02 FROM pdb01;  
  
ALTER PLUGGABLE DATABASE pdb01 CLOSE IMMEDIATE;  
  
ALTER PLUGGABLE DATABASE pdb01 OPEN;  
  
ALTER PLUGGABLE DATABASE pdb02 OPEN;
```

Listagem 5. Remover um PDB.

```
ALTER PLUGGABLE DATABASE pdb01 CLOSE IMMEDIATE;  
  
DROP PLUGGABLE DATABASE pdb01 INCLUDING DATAFILES;  
  
DROP PLUGGABLE DATABASE pdb01 KEEP DATAFILES;
```

Listagem 6. Desconectar um PDB.

```
ALTER PLUGGABLE DATABASE pdb02 CLOSE IMMEDIATE;  
  
ALTER PLUGGABLE DATABASE pdb02 UNPLUG INTO '/u01/app/oracle12c/oradata/  
unplugged/pdb02.xml';
```

Listagem 7. Conectar um PDB.

```
SET SERVEROUTPUT ON  
DECLARE  
  l_result BOOLEAN;  
BEGIN  
  l_result := DBMS_PDB.check_plug_compatibility(  
    pdb_descr_file =>  
    '/u01/app/oracle12c/oradata/unplugged/pdb02.xml',  
    pdb_name      =>'pdb03');  
  
  IF l_result THEN  
    DBMS_OUTPUT.PUT_LINE('compatible');  
  ELSE  
    DBMS_OUTPUT.PUT_LINE('incompatible');  
  END IF;  
END;  
/  
compatible  
  
PL/SQL procedure successfully completed.  
  
CREATE PLUGGABLE DATABASE pdb03 USING '/u01/app/oracle12c/oradata/unplugged/pdb02.xml';  
  
ALTER PLUGGABLE DATABASE pdb03 OPEN;
```

Renomear e mover um datafile Online

Nas versões anteriores, quando era necessário fazer alguma manutenção no datafile de uma tablespace, era necessário realizar alguns procedimentos como colocar a tablespace em READ ONLY e depois o datafile offline. Mas agora podemos fazer todas estas atividades online e sem causar qualquer indisponibilidade. Veja os comandos na **Tabela 6**.

Renomear um datafile	ALTER DATABASE MOVE DATAFILE '<datafile>' TO '<datafile>';
Mover um datafile non-ASM para o ASM e o contrário	ALTER DATABASE MOVE DATAFILE '<filesystem>' TO '+DG_DATA_<datafile>'; ALTER DATABASE MOVE DATAFILE '+DG_DATA_<datafile>' TO '<filesystem>';
Mover um datafile no ASM	ALTER DATABASE MOVE DATAFILE '+DG_DATA_01_X' TO '+DG_DATA_02_Y';
Mover um datafile existente sobrepondo outro	ALTER DATABASE MOVE DATAFILE 'xxxx' TO 'xxxx' REUSE;
Mover um datafile existente para outro local mantendo o antigo	ALTER DATABASE MOVE DATAFILE 'xxxx' TO 'xxxx' KEEP;

Tabela 6. Recursos para manutenção de datafiles online

Oracle Enterprise Manager

O Oracle Enterprise Manager é uma ferramenta da Oracle utilizada para centralizar a monitoração, administração e gerenciamento da infraestrutura de TI. Nas versões anteriores, era conhecido como:

- Oracle Enterprise Manager Grid Control 10g;
- Oracle Enterprise Manager Grid Control 11g.

Na versão 12c, agora ele é conhecido como: Oracle Enterprise Manager Cloud Control 12c. Esta nova versão trouxe muitas novidades e melhorias, desde a interface gráfica que está com um novo visual até na performance, estabilidade e usabilidade. Além dos bancos de dados, muitos outros componentes da infraestrutura podem ser monitorados de forma centralizada utilizando apenas esta ferramenta, como mostra a **Figura 10**.

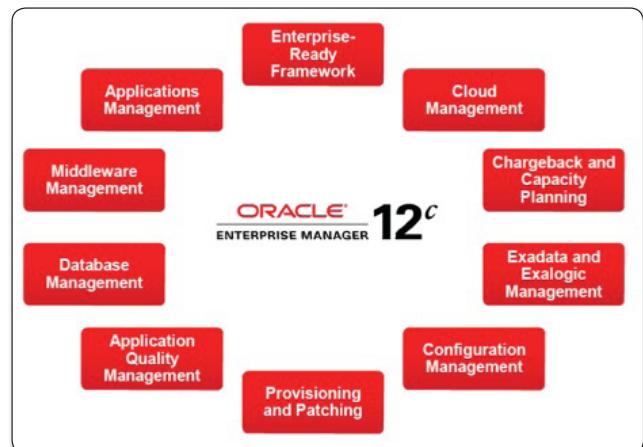


Figura 10. Oracle Enterprise Manager Cloud Control 12c

Certificações

As certificações: OCA e OCP já estão disponíveis para a versão 12c, conforme a **Figura 11** retirada do site da Oracle.

Para quem já possui umas das seguintes certificações: 9i, 10g ou 11g, também é possível fazer a atualização para a versão 12c. O exame que deve ser realizado é o 1Z0-60.

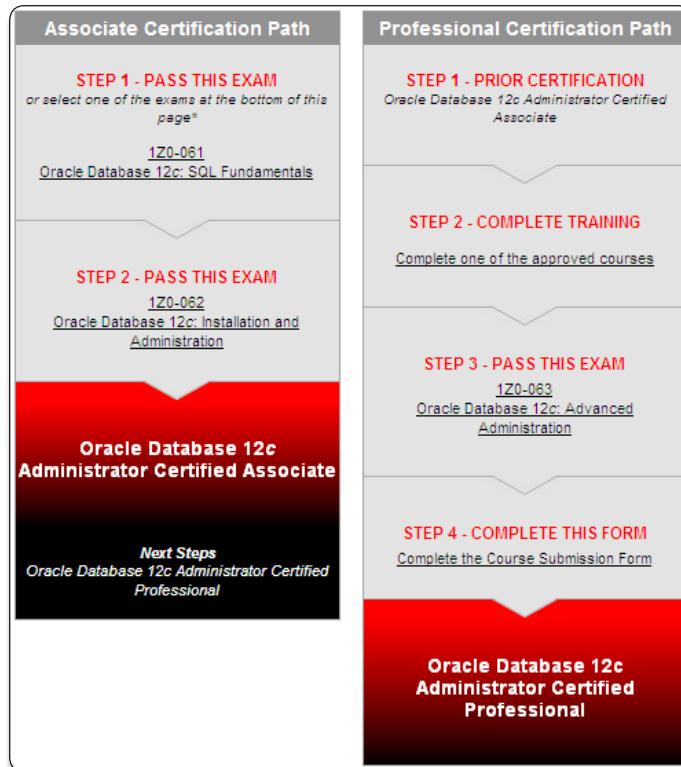


Figura 11. Certificações Oracle Database 12c – Para quem não tem nenhuma

Conclusão

Com base nas pesquisas realizadas e nas comparações apresentadas neste artigo, é possível identificar que as tecnologias voltadas para o Cloud Computing estão em alta no momento. A nova versão do banco de dados Oracle Database 12c chegou para acompanhar

estas mudanças que o mundo de tecnologia está enfrentando no momento, com muitas novidades e cheio de funcionalidades novas. A Oracle está investindo bastante nestes novos conceitos, e vale muito a pena investir, principalmente enquanto ainda é novidade e poucas pessoas têm o conhecimento.

Autor



Gilson Martins

gilson.pmartins@gmail.com

Possui oito anos de experiência como profissional de TI. Estudante de Engenharia da Computação com ênfase em Software pela Fundação Santo André (FAENG). Durante os últimos quatro anos tem se especializado em administração de banco de dados. Possui em seu currículo diversos cursos na área e grande experiência em diversos tipos de ambientes pequeno, médio e grande porte. Já trabalhou com SQL Server e atualmente é consultor Oracle focado em Performance e Tuning de banco de dados para uma grande montadora no ABC, atuando em ambientes de alta disponibilidade e criticidade.



Links:

Site Oracle-Base

<http://www.oracle-base.com>

Oracle Database Concepts 12c (12.1)

http://docs.oracle.com/cd/E16655_01/server.121/e17633/release_changes.htm

Oracle Database Administrators Guide 12c (12.1)

http://docs.oracle.com/cd/E11882_01/server.112/e25494.pdf

Oracle Database New Features Guide 12c (12.1)

http://docs.oracle.com/cd/E11882_01/server.112/e16638.pdf

Oracle Multitenant

<http://www.oracle.com/technetwork/database/multitenant/overview/index.html>

Você gostou deste artigo?

Dê seu voto em www.devmedia.com.br/sqlmagazine/feedback

Ajude-nos a manter a qualidade da revista!



Bancos conectáveis com o Oracle 12c

Conheça o novo conceito que chega ao mercado com a versão 12c

Até agora, a Oracle tem sido o banco de dados tradicional adequado para os datacenters empresariais dentro das paredes das empresas. Depois de seis anos, o lançamento mais aguardado era a nova versão do banco de dados, o Oracle Database 12c. Durante o Oracle Open World 2012, que é o principal evento da Oracle Corporation para o lançamento de novos produtos ao mundo, realizada em San Francisco, Larry Ellison anunciou que o banco de dados Oracle também estaria nas nuvens (ambiente cloud) pela introdução dos conceitos de bancos de dados conectáveis (pluggable databases) usando uma arquitetura de multi-inquilino (multitenant) e a fácil movimentação do banco de dados entre sistemas, plataformas ou releases.

Quando o banco de dados estiver nas nuvens, o mesmo deve ser independente de hardware, de plataforma e de versão de modo a ter todas as características de Plataforma como Serviço (PaaS) para o ambiente middleware. Portanto, os usuários devem estar familiarizados com os conceitos de "Contêiner de Bancos de Dados" ou CDB e "Banco de Dados Conectável" ou PDB. Iremos nos referir aos termos PDB e CDB para os conceitos de Banco de Dados Conectável e Contêiner de Banco de Dados respectivamente.

Vários inquilinos (Multitenant)

Antes de aprofundarmos em pormenores mais detalhados sobre CDB e PDB, vamos apresentar algumas informações básicas sobre multi-inquilino. A maioria das organizações usa o multi-inquilino através de uma lógica de nível de aplicação, ou seja, vários clientes ou diferentes entidades de informação dentro do mesmo banco de dados. Eles podem ser configurados com vários esquemas diferentes ou dentro de um mesmo esquema.

Tem dado muita dor de cabeça administrar ambiente quando vários bancos de dados estão sendo executados em uma única máquina. Os backups precisam

Resumo DevMan

Porque este artigo é útil:

O artigo descreve a nova arquitetura de bancos conectáveis que surgiu a partir da versão 12c do banco de dados Oracle. Além do conceito serão mostradas algumas consultas a novas tabelas do dicionário de dados relativas a esses conceitos. Com a arquitetura em ambiente nas nuvens, os administradores de banco de dados, com base nesse artigo, poderão conhecer mais os conceitos e usufruir desta nova funcionalidade disponibilizada pelo banco de dados Oracle 12c.

ser executados separadamente. Cada banco de dados tem seu próprio espaço de memória e cada banco tem seus processos background interno do Oracle (ver **Figura 1**). Isto aumenta a utilização do servidor que ele pode controlar em termos de cargas.

Ao consolidar em um contêiner de banco de dados, ao mesmo tempo mantendo todos eles como bancos de dados separados, tem-se uma grande vantagem da consolidação, desempenho, capacidade e perspectiva operacional. Dessa forma, podem-se gerenciar diversos bancos de dados como um só aumentando a utilização de recursos do servidor, ao mesmo tempo em que se reduz o tempo e os esforços exigidos nas atualizações, backups e recuperação de bancos de dados, entre outras inúmeras facilidades.

Também é disponibilizada a clonagem e provisionamento praticamente instantâneos dos bancos de dados, sendo a plataforma ideal para desenvolvimento e testes de bancos de dados nas nuvens.

A funcionalidade Multitenant trabalha com todos os recursos do Oracle Database como Real Application Clusters, Partitioning, Data Guard, Compression, Automatic Storage Management, Real Application Testing, Transparent Data Encryption, Database Vault, entre outros. Isso vai ajudar em um grande caminho para a consolidação de muitas empresas e, ao mesmo tempo, irá reduzir o consumo do servidor dando significativamente o máximo de retorno sobre investimentos (ROI) sobre as tecnologias de banco de dados de middleware. Percebe-se que esse novo recurso tem muitas vantagens para pequenos clientes, bem

como para clientes corporativos de grande porte, não importa a forma como olhamos para ele. Ele vai fazer as empresas se tornarem mais inteligentes em termos de utilização da capacidade de computação que eles têm.

Nova Arquitetura de Banco de Dados Conectável

Memória e processos necessários apenas no nível do Contêiner



Figura 1. Configuração de memória e processos num contêiner de banco de dados

CDB – Container Database

Nesse tipo de arquitetura se consegue separar toda a parte do CORE do banco de dados das aplicações existentes nele, ou seja, pelo dicionário de dados. Assim, podemos distinguir entre informações que pertence ao banco de dados das informações das aplicações, tornando assim possível definir um *contêiner* de banco de dados e posteriormente anexar/conectar novos bancos de dados conforme a demanda, centralizando e otimizando a utilização dos recursos de infraestrutura.

PDB – Pluggable Database

No Oracle 12c, em um ambiente de banco de dados conectável, podemos criar um único contêiner de banco de dados e conectar vários bancos de dados para este contêiner. Todos esses bancos de dados compartilham os mesmos processos internos e de segundo plano do Oracle, bem como a área de memória compartilhada, ao contrário das versões anteriores, em que cada banco tinha seus próprios processos background e memória compartilhada. Isso ajuda no processo de consolidação dos bancos de dados e reduz a sobrecarga de gerenciamento de vários bancos de dados separados em um mesmo servidor.

Uma forma fácil de compreender melhor esses dois conceitos seria pensarmos em um trem de carga com muitos vagões. Imaginamos que tenhamos 200 vagões. O trem pode ter várias cargas com conteúdos diferentes separados com destino certo para clientes diferentes, mas apesar disso tudo o motor que leva essas cargas é único. Não seria comprehensível termos 200 trens cada um levando um vagão, pois é mais eficiente consolidá-los em um único trem levando os 200 vagões. Dessa forma, vamos gastar menos combustível diminuindo os gastos com motoristas ficando muito mais fácil de gerenciar. O trem de carga é basicamente o CDB e cada vagão é o PDB.

Dicionário de dados distintos

Com esses novos conceitos, muda-se a forma de como ficará o gerenciamento do dicionário de dados de um CDB, bem como para

todos os PDBs. O CDB possui uma área onde tem um núcleo raiz do Oracle definida com o nome “root”. Isto terá um dicionário de dados próprio que será fornecido pela Oracle em novas versões e/ou patches a serem disponibilizados. Esse controle será feito exclusivamente pela Oracle. Cada PDB terá seu próprio dicionário ou metadados que pertencem aos bancos. Cada dicionário PDB fica definido dentro de seu próprio banco. Cada PDB também terá uma cópia “somente leitura” do dicionário do Oracle. Podemos assim chamá-lo de arquitetura de “Dicionário de Dados divididos” que permite que o PDB possa manter o seu próprio dicionário e torna mais fácil a portabilidade entre vários CDBs. Cada PDB é um sistema autônomo em si, uma definição clara declarativa de um aplicativo, que não sabe nada sobre outros PDBs dentro da mesma CDB.

Compatível: RAC, Gerente de Recursos etc.

Num ambiente com RAC, cada instância abre o CDB como um todo de modo que versões seriam iguais para o CDB, bem como para todos os PDBs. Portanto, os PDBs também são totalmente compatíveis com ambiente RAC. Para um banco de dados ser definido como conectável é necessário que ele seja totalmente compatível com todas as opções de banco de dados e recursos, incluindo gerenciador de recursos. Resource Manager (Gerenciador de Recursos) é estendido para criar, desligar, ligar, e clonagem, apagar ou até mesmo configurar o PDB em modo aberto.

Novo administrador: CDB Administrator

Um novo administrador foi introduzido na versão de lançamento 12.1. Este novo papel de administrador é chamado de “Administrador CDB”. Esse perfil será usado para gerenciar a nova área chamada de “root namespace” do CDB. Existem vários comandos que podem ser executados a partir da raiz de um CDB e todos os comandos são compatíveis num PDB. Você pode usar um dos vários comandos a seguir para descobrir qual contêiner você está atualmente conectado:

```
SQL> SELECT pdb FROM dba_services
```

```
SQL> SELECT Sys_Context('userenv','con_name') "MY_CONTAINER" FROM dual;
```

```
SQL> SHOW con_name
```

```
SQL> SELECT name, con_id FROM v$active_services ORDER BY 1;
```

Desconecte e reconecte PDB

Nessa nova versão 12c do banco de dados Oracle você pode desconectar um banco de dados a partir de um banco de dados contêiner (CDB) e, em seguida, conectar em outro banco de dados CDB. Este é um recurso muito bom. Isto pode ser útil em várias situações:

- Migração de bases de dados para uma nova plataforma;
- Bancos de dados migrando para um novo hardware;
- Migração de bases de dados para novas versões de banco de dados;

- Movendo bancos de dados para sistemas diferentes;
- Melhorar a alta disponibilidade dos bancos de dados.

Rápido provisionamento, aplicação de patches e atualizações

Podemos clonar um PDB no mesmo CDB ou em qualquer outro CDB. Os PDBs também podem ser provisionados muito mais rapidamente quando todos os CDB possuem uma “PDB seed” que funciona como um template padrão de onde o novo PDB pode ser provisionado quando for necessário.

Além disso, a função de reimplantação se torna muito mais fácil, pois podemos desligar o banco de dados de uma plataforma ou uma versão CDB e, em seguida, ligá-lo em outro CDB que está em outra plataforma ou versão. Isso fará com que os esforços de atualização de versão, aplicação de patches e reimplantação sejam muito mais rápidos! Quando você atualiza o CDB, todos os PDBs vão ficar atualizados também. Se você desejar controlar quando os PDBs devem ser atualizados, você pode criar outra versão CDB e, em seguida, desligar a partir da antiga versão e ligar para a nova versão do banco de dados.

Administração de banco de dados

Com o CDB, a administração de banco de dados torna-se mais fácil, pois pode ser gerenciado como um todo, por exemplo, para backup usando RMAN, a criação de recuperação de desastres usando Data Guard, etc. No entanto, podemos fazer backup de PDBs individualmente, caso seja necessário. Podemos também restaurar e recuperar um PDB a um ponto no tempo (*point in time recover*) fazendo recuperação separada no nível de PDB ligados a um CDB.

Além disso, podem ser feitos backups, configuração de data guard, recuperação de atualização de base em apenas um PDB. Quando se deseja controlar quando e como cada PDB será atualizado, podemos então criar um novo CDB e, em seguida, desconectar e reconectar o PDB sempre quando o PDB estiver pronto para ser migrado. Isto torna o trabalho mais fácil, organizado e com controle total sobre a operação.

Serviço de nomes únicos para PDBs

Um serviço é criado e iniciado dentro de um PDB que o identifica como o contêiner inicial, embora os metadados sejam armazenados separadamente para esse PDB. Mas esse serviço é único dentro do CDB.

Conectando a um CDB

Agora vamos ver alguns exemplos para a criação de um PDB. Neste exemplo, vamos considerar que temos um contêiner de banco de dados chamado CDBPRD. Vamos criar um banco de dados PDB nesse contêiner de banco de dados.

Na **Listagem 1** podemos conferir uma conexão com o contêiner de banco de dados.

Para verificarmos quantos bancos conectáveis estão contidos no contêiner de banco de dados, pode-se executar os comandos apresentados na **Listagem 2**.

Listagem 1. Exemplo de conexão com o contêiner de um banco de dados.

```
$ sqlplus sys/senha@DBSRV/cdb1 as sysdba
SQL*Plus: Release 12.1.0.0.2 Production on Wed Sep 4 09:45:42 2013
Copyright (c) 1982, 2013, Oracle. All rights reserved.

Connected to:
Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options
SQL> SET echo ON
SQL> SET linesize 200
SQL> SET pagesize 10000
SQL> SET serveroutput ON
SQL> COLUMN "DB DETAILS" format a100
SQL> SELECT
  2 'DB_NAME:' ||Sys_context('userenv','db_name') ||
  3 '/CDB?:' ||(SELECT cdb FROM v$database) ||
  4 '/AUTH_ID:' ||Sys_context('userenv','authenticated_identity') ||
  5 '/USER:' ||Sys_context('userenv','current_user') ||
  6 '/CONTAINER:' ||Nvl(Sys_context('userenv','con_Name'), 'NON-CDB')
  7 "DB DETAILS"
  8 FROM dual
  9 /
DB DETAILS
```

DB_NAME:cdb1 / CDB?:YES / AUTH_ID:SYS / USER:SYS / CONTAINER:CDB\$ROOT

Listagem 2. Identificando os bancos conectáveis que estão no contêiner.

```
SQL> SET serveroutput ON
SQL> COLUMN "RESTRICTED" format a10
SQL> SELECT v.name
  2   v.open_mode,
  3   Nvl(v.RESTRICTED 'n/a') "RESTRICTED"
  4   d.status
  5   FROM v$pdbs v
  6   inner join dba_pdbs d
  7   using (GUID)
  8 ORDER BY v.create_scn
  9 /
NAME      OPEN_MODE RESTRICTED STATUS
-----  -----  -----  -----
PDB$SEED  READ ONLY    NO     NORMAL
```

Criando um banco de dados conectável (PDB)

Já verificamos anteriormente que cada banco CDB possui um modelo padrão de criação de um PDB. Na **Listagem 3** apresentamos como é simples a criação de um PDB usando esse recurso.

Como podemos ver, colocamos intencionalmente o tempo de criação de um banco conectável, que nesse caso foi menos de um minuto para verificarmos quanto rápido um novo banco de dados pode ser implantado. Muitas vezes gasta-se muito mais tempo que isso para um administrador de banco de dados criar um banco nos modelos tradicionais. Dessa forma, se ganha um tempo extra que o DBA pode usar para outras atividades como ajustar desempenho de consultas, entre outras. No exemplo anterior usamos um banco baseado em file system, mas também pode ser usado da mesma forma em ambiente ASM, como demonstra a **Listagem 4**.

Depois da criação, ao verificarmos novamente a quantidade de bancos conectáveis no contêiner de banco de dados, temos o resultado exposto na **Listagem 5**.

Listagem 3. Exemplo de criação de um PDB.

```
SQL> SET TIMING ON
SQL> create pluggable database pdb01
2 admin user app_Admin identified by senha
3 file_name_convert = ('/pdbseed','/pdb01')
4 /
```

Pluggable database created.

Elapsed: 00:00:53.32

```
SQL> SET TIMING OFF
SQL>
SQL> alter pluggable database pdb01 open
2 /
```

Pluggable database altered.

Listagem 4. Exemplo de criação de um PDB, em ambiente ASM.

```
SQL> select con_id, tablespace_name, File_Name
2 from cdb_data_files
3 where file_name like '%/c01p/pdbseed/%'
4 or file_name like '%/c01p/pdb01/%'
5 order by 1,2
6 /
```

CON_ID	TABLESPACE_NAME	FILE_NAME
2	SYSAUX	/u01/app/oracle/oradata/cdbprd/ pdbseed/sysaux01.dbf
2	SYSTEM	/u01/app/oracle/oradata/cdbprd/ pdbseed/system01.dbf
3	SYSAUX	/u01/app/oracle/oradata/cdbprd/ pdb01/sysaux01.dbf
3	SYSTEM	/u01/app/oracle/oradata/cdbprd/ pdb01/system01.dbf

Listagem 5. Identificando os bancos conectáveis que estão no contêiner.

```
SQL> SET serveroutput ON
SQL> COLUMN "RESTRICTED" format a10
SQL> SELECT v.name,
2   v.open_mode
3   Nvl(v.RESTRICTED,'n/a')"RESTRICTED"
4   d.status
5  FROM v$pdbs v
6    inner join dba_pdbs d
7    using (GUID)
8  ORDER BY v.create_scn
9 /
```

NAME	OPEN_MODE	RESTRICTED	STATUS
PDB\$SEED	READ ONLY	NO	NORMAL
PDB01	READ WRITE	NO	NORMAL

Existe um comando que pode ser usado para abrir todos os bancos conectáveis dentro de um contêiner de banco de dados. Este comando é:

```
SQL> alter pluggable database all open;
```

Pluggable database altered.

Para podermos descartar um banco PDB, é utilizado um comando bastante simples, apresentado a seguir:

```
SQL> alter pluggable database pdb01 close;
```

Pluggable database altered.

```
SQL> drop pluggable database pdb01 including datafiles;
```

Pluggable database dropped.

Convertendo um banco não-CDB para banco conectável PDB

Podemos converter bancos que não estejam ligados a um CDB em um banco conectável (PDB). Para tanto, esse banco precisa estar na versão 12c. Assim, os bancos de dados 11g existentes terão de ser atualizados para 12c antes que eles possam ser parte de um CDB na versão 12c.

A primeira coisa que precisamos é conectar ao banco não-CDB e executar a procedure DBMS_PDB.DESCRIBE para construir um arquivo XML que conte os metadados que descrevem esse banco não-CDB. Essa execução só deve ser feita em modo de apenas leitura (READ ONLY). Veja a [Listagem 6](#).

Listagem 6. Conversão de banco não-CDB para PDB – Parte 1.

```
$ sqlplus sys as sysdba
```

```
SQL*Plus: Release 12.1.0.0.2 Production on Wed Sep 4 11:25:21 2013
```

Copyright (c) 1982, 2013, Oracle. All rights reserved.

Connected to:

Oracle Database 12c Enterprise Edition Release 12.1.0.1.0 - 64bit Production
With the Partitioning, OLAP, Data Mining and Real Application Testing options

```
SQL> shutdown immediate;
```

Database closed.

Database dismounted.

ORACLE instance shut down.

```
SQL> startup mount
```

ORACLE instance started.

Total System Global Area 801701888 bytes

Fixed Size 2293496 bytes

Variable Size 314573064 bytes

Database Buffers 478150656 bytes

Redo Buffers 6684672 bytes

Database mounted.

Database mounted.

```
SQL> alter database open read only;
```

Database altered.

BEGIN

```
  dbms_pdb.Describe(pdb_descr_file => '/home/oracle/testedb2.xml');
```

END;

/

PL/SQL procedure successfully completed.

Bancos conectáveis com o Oracle 12c

Agora desligamos e reabrimos o banco não-CDB em modo de leitura e gravação.

Depois, precisamos verificar se o banco não-CDB está pronto para a conversão em um banco conectável PDB que posteriormente será acoplado a um contêiner de banco de dados CDB.

Para isso, precisamos conectar ao CDB e executar o pacote DBMS_PDB.CHECK_PLUG_COMPATIBILITY onde será passada a localização do arquivo de metadados XML que foi gerado no passo anterior. Observe os comandos para realizar esse processo na **Listagem 7**.

Listagem 7. Conversão de banco não-CDB para PDB – Parte 2.

```
SQL> SET SERVEROUTPUT ON
SQL> DECLARE
 2 compatible CONSTANT VARCHAR2(3) :=
 3 CASE dbms_pdb.Check_plug_compatibility(
 4   pdb_descr_file => '/home/oracle/testedb2.xml',
 5   pdb_name => 'TESTEDB2'
 6 WHEN TRUE THEN 'YES'
 7 ELSE 'NO'
 8 END;
 9 BEGIN
10   dbms_output.Put_line(compatible);
11 END;
12 /
NO
```

PL/SQL procedure successfully completed.

Depois, o banco não-CDB será desligado e conectado a um CDB através da conversão para um PDB, conforme os comandos:

```
SQL> CREATE PLUGGABLE DATABASE testedb2 using '/home/oracle/testedb2.xml'
2 nocopy
3 tempfile reuse;
```

Pluggable database created.

O arquivo XML descreve com precisão os locais atuais dos arquivos do PDB. Como estamos usando os arquivos de dados não-CDB 12c existentes para criar o banco de dados conectável a um CDB, vamos usar a opção NOCOPY. Se quisermos, podemos deixar o banco de dados 12c não-CDB clonar para criar um novo banco de dados conectável PDB.

Podemos ver, através da **Listagem 8**, que o banco de dados TESTEDB2 é agora um PDB e está em estado montado.

Nesse momento executaremos o script *ORACLE_HOME/rdbms/admin/noncdb_to_pdb.sql*, que deve ser executado antes que o PDB seja aberto pela primeira vez.

Após sua execução, podemos então abrir o banco PDB testedb2 em modo de leitura e gravação, conforme expõe a **Listagem 9**.

Clonando um PDB

Na **Listagem 10** mostramos como facilmente podemos clonar um banco de dados dentro de um mesmo CDB ou de outro CDB.

Listagem 8. Verificação do banco de dados TESTEDB2.

```
SQL> SELECT con_id, name, open_mode FROM v$pdbs
CON_ID      NAME        OPEN_MODE
-----      -----
 2          PDB$SEED    READ ONLY
 3          PDB01      READ WRITE
 4          TESTEDB2  MOUNTED

SQL> SELECT pdb_name, dbid, con_id, status FROM cdb_pdbs;
PDB_NAME    DBID      CON_ID      STATUS
-----      -----      -----
PDB01       571204526  1          NORMAL
PDB$SEED    4061755545  1          NORMAL
TESTEDB2   1141538778  1          NEW
```

```
SQL> ALTER SESSION SET CONTAINER =testedb2;
```

Session altered.

```
SQL> show con_name
```

```
CON_NAME
-----
TESTEDB2
```

Listagem 9. Abrindo o banco PDB TESTEDB2.

```
SQL> ALTER SESSION SET CONTAINER =CDB$ROOT;
```

Session altered.

```
SQL> show con_name
```

```
CON_NAME
-----
CDB$ROOT
```

```
SQL> SELECT pdb_name, dbid, con_id, status FROM cdb_pdbs;
```

```
PDB_NAME    DBID      CON_ID      STATUS
-----      -----      -----
PDB01       571204526  1          NORMAL
PDB$SEED    4061755545  1          NORMAL
TESTEDB2   1141538778  1          NORMAL
3 rows selected.
```

Primeiramente precisaremos verificar a área do CDB envolvida.

Desconectando e reconectando um PDB

Essa característica é bastante útil, pois permite alta disponibilidade e escalabilidade dos sistemas de banco de dados, especialmente em um ambiente com infraestrutura nas nuvens. Veja um exemplo na **Listagem 11**.

Para acessar o contêiner de banco de dados CDB2, veja o código apresentado na **Listagem 12**.

Listagem 10. Como clonar um banco de dados.

```

SQL> SET echo ON
SQL> SET linesize 200
SQL> SET pagesize 10000
SQL> SET serveroutput ON
SQL> COLUMN "DB DETAILS" format a100
SQL> SELECT
  2 'DB_NAME:' ||Sys_context('userenv','db_name') ||
  3 '/ CDB?:' ||(SELECT cdb FROM v$database) ||
  4 '/ AUTH_ID:' ||Sys_context('userenv','authenticated_identity') ||
  5 '/ USER:' ||Sys_context('userenv','current_user') ||
  6 '/ CONTAINER:' ||Nvl(Sys_context('userenv','con_Name')) 'NON-CDB'
  7 "DB DETAILS"
  8 FROM dual
  9 /
DB DETAILS
-----
DB_NAME:cdb1 / CDB?: YES / AUTH_ID: SYS / USER: SYS / CONTAINER: CDB$ROOT

SQL> SET serveroutput ON
SQL> COLUMN "RESTRICTED" format a10
SQL> SELECT v.name,
  2   v.open_mode,
  3   Nvl(v.RESTRICTED 'n/a') "RESTRICTED",
  4   d.status
  5   FROM v$pdbs v
  6   inner join dba_pdbs d
  7   using (GUID)
  8 ORDER BY v.create_scn
  9 /
NAME      OPEN_MODE    RESTRICTED    STATUS
-----  -----  -----  -----
PDB$SEED  READ ONLY     NO        NORMAL
PDB01     READ WRITE    NO        NORMAL
TESTEDB2  READ WRITE    NO        NORMAL

SQL> alter pluggable database PDB01 close
2 /
Pluggable database altered.

SQL> alter pluggable database PDB01 open read only
2 /
Pluggable database altered.

SQL> SET TIMING ON
SQL> create pluggable database PDB02
2 from PDB01
3 file_name_convert = ('/pdb01','/pdb02')
4 /

Pluggable database created.

Elapsed: 00:00:31.90
SQL> SET TIMING OFF

SQL> alter pluggable database PDB01 close
2 /
Pluggable database altered.

```

```

SQL> alter pluggable database PDB01 open
2 /
Pluggable database altered.

SQL> alter pluggable database PDB02 open
2 /
Pluggable database altered.

SQL> SET serveroutput ON
SQL> COLUMN "RESTRICTED" format a10
SQL> SELECT v.name,
  2   v.open_mode,
  3   Nvl(v.RESTRICTED 'n/a') "RESTRICTED",
  4   d.status
  5   FROM v$pdbs v
  6   inner join dba_pdbs d
  7   using (GUID)
  8 ORDER BY v.create_scn
  9 /
NAME      OPEN_MODE    RESTRICTED    STATUS
-----  -----  -----  -----
PDB$SEED  READ ONLY     NO        NORMAL
PDB01     READ WRITE    NO        NORMAL
TESTEDB2  READ WRITE    NO        NORMAL
PDB02     READ WRITE    NO        NORMAL

SQL> select con_id,tablespace_name,File_Name
  2   from cdb_data_files
  3   order by 1,2
  4 /
CON_ID  TABLESPACE_NAME          FILE_NAME
-----  -----  -----
1       SYSAUX                  /u01/app/oracle/oradata/cdbprd/
                                sysaux01.dbf
1       SYSTEM                   /u01/app/oracle/oradata/cdbprd/
                                system01.dbf
1       USERS                    /u01/app/oracle/oradata/cdbprd/
                                users01.dbf
1       UNDOTBS                 /u01/app/oracle/oradata/cdbprd/
                                undotbs01.dbf
2       SYSAUX                  /u01/app/oracle/oradata/cdbprd/
                                pdbsseed/sysaux01.dbf
2       SYSTEM                   /u01/app/oracle/oradata/cdbprd/
                                pdbsseed/system01.dbf
3       SYSAUX                  /u01/app/oracle/oradata/cdbprd/
                                pdb01/sysaux01.dbf
3       SYSTEM                   /u01/app/oracle/oradata/cdbprd/
                                pdb01/system01.dbf
4       SYSAUX                  /u01/app/oracle/oradata/cdbprd/
                                testedb2/sysaux01.dbf
4       SYSTEM                   /u01/app/oracle/oradata/cdbprd/
                                testedb2/system01.dbf
5       SYSAUX                  /u01/app/oracle/oradata/cdbprd/
                                pdb02/sysaux01.dbf
5       SYSTEM                   /u01/app/oracle/oradata/cdbprd/
                                pdb02/system01.dbf

```

Bancos conectáveis com o Oracle 12c

Listagem 11. Como desconectar e conectar um PDB.

```
SQL> SET echo ON
SQL> SET linesize 200
SQL> SET pagesize 10000
SQL> SET serveroutput ON
SQL> COLUMN "DB DETAILS" format a100
SQL> SELECT
  2 'DB_NAME:' || Sys_context('userenv','db_name') ||
  3 '/ CDB?:' || (SELECT cdb FROM v$database) ||
  4 '/ AUTH_ID:' || Sys_context('userenv','authenticated_identity') ||
  5 '/ USER:' || Sys_context('userenv','current_user') ||
  6 '/ CONTAINER:' || Nvl(Sys_context('userenv','con_Name')) 'NON-CDB'
  7 'DB DETAILS'
  8 FROM dual
  9 /
DB DETAILS
```

DB_NAME: cdb1 / CDB?: YES / AUTH_ID: SYS / USER: SYS / CONTAINER: CDB\$ROOT

```
SQL> SET serveroutput ON
SQL> COLUMN "RESTRICTED" format a10
SQL> SELECT v.name,
  2   v.open_mode,
  3   Nvl(v.RESTRICTED,'n/a') "RESTRICTED",
  4   d.status
  5   FROM v$pdbs v
  6   inner join dba_pdbs d
  7   using (GUID)
  8 ORDER BY v.create_scn
  9 /
```

NAME	OPEN_MODE	RESTRICTED	STATUS
PDB\$SEED	READ ONLY	NO	NORMAL
PDB01	READ WRITE	NO	NORMAL
TESTEDB2	READ WRITE	NO	NORMAL
PDB02	READ	NO	NORMAL

```
SQL> alter pluggable database PDB01 close
  2 /
```

Pluggable database altered.

```
SQL> alter pluggable database PDB01
  2 unplug into
```

```
  3 '/u01/app/oracle/oradata/cdb1/pdb01/pdb01.xml'
  4 /
```

Pluggable database altered.

```
SQL> SET serveroutput ON
SQL> COLUMN "RESTRICTED" format a10
SQL> SELECT v.name
  2   v.open_mode,
  3   Nvl(v.RESTRICTED,'n/a') "RESTRICTED"
  4   d.status
  5   FROM v$pdbs v
  6   inner join dba_pdbs d
  7   using (GUID)
  8 ORDER BY v.create_scn
  9 /
```

NAME	OPEN_MODE	RESTRICTED	STATUS
PDB\$SEED	READ ONLY	NO	NORMAL
PDB01	MOUNTED	n/a	UNPLUGGED
TESTEDB2	READ WRITE	NO	NORMAL
PDB02	READ WRITE	NO	NORMAL

```
SQL> drop pluggable database PDB01 keep datafiles
  2 /
```

Pluggable database dropped.

```
SQL> COLUMN "RESTRICTED" format a10
SQL> SELECT v.name
  2   v.open_mode,
  3   Nvl(v.RESTRICTED,'n/a') "RESTRICTED"
  4   d.status
  5   FROM v$pdbs v
  6   inner join dba_pdbs d
  7   using (GUID)
  8 ORDER BY v.create_scn
  9 /
```

NAME	OPEN_MODE	RESTRICTED	STATUS
PDB\$SEED	READ ONLY	NO	NORMAL
TESTEDB2	READ WRITE	NO	NORMAL
PDB02	READ WRITE	NO	NORMAL



Listagem 12. Como acessar o contêiner de banco de dados CDB2.

```
SQL> SET echo ON
SQL> SET linesize 200
SQL> SET pagesize 10000
SQL> SET serveroutput ON
SQL> COLUMN "DB DETAILS" format a100
SQL> SELECT
  2  'DB_NAME:' ||Sys_context('userenv','db_name') ||
  3  '/ CDB?:'  ||(SELECT cdb FROM v$database) ||
  4  '/ AUTH_ID:' ||Sys_context('userenv','authenticated_identity') ||
  5  '/ USER:'  ||Sys_context('userenv','current_user') ||
  6  '/ CONTAINER:' ||Nvl(Sys_context('userenv','con_Name'), 'NON-CDB')
  7  "DB DETAILS"
  8  FROM dual
  9 /
DB DETAILS
-----
DB_NAME:cdb2 / CDB?: YES / AUTH_ID: SYS / USER: SYS / CONTAINER: CDB$ROOT

SQL> BEGIN
  2  IF not
  3    dbms_pdb.Check_plug_compatibility(
  4      pdb_desc_file =>
  5        '/u01/app/oracle/oradata/cdb1/pdb01/pdb01.xml')
  6  THEN
  7    raise_application_error(-20000,'PDB não é compatível para conectar');
  8  END IF;
  9 END;
10 /
```

PL/SQL procedure successfully completed.

```
SQL> create pluggable database PDB01
  2  using '/u01/app/oracle/oradata/cdb1/pdb01/pdb01.xml'
  3  move
  4  file_name_convert = ('/cdb1','/cdb2/')
  5 /

Pluggable database created.

SQL> alter pluggable database PDB01 open
  2 /

Pluggable database altered.

SQL> COLUMN "RESTRICTED" format a10
SQL> SELECT v.name
  2  v.open_mode
  3  Nvl(v.RESTRICTED,'n/a') "RESTRICTED"
  4  d.status
  5  FROM v$pdbs v
  6  inner join dba_pdbs d
  7  using (GUID)
  8  ORDER BY v.create_scn
  9 /
-----          -----          -----          -----
NAME        OPEN_MODE   RESTRICTED     STATUS
-----          -----          -----          -----
PDB$SEED    READ ONLY    NO           NORMAL
PDB01       READ WRITE   NO           NORMAL
```

Conclusão

O banco de dados Oracle 12c oferece a arquitetura de banco de dados conectáveis que são extremamente úteis para serem usados especialmente em ambiente de banco de dados em nuvem, podendo ser usado numa nuvem pública da Oracle, ou numa nuvem interna por trás de firewalls corporativos ou até mesmo em clientes com infraestrutura discreta com cluster na versão 12.1.

Autor



Ricardo Ferro

rferro@gmail.com

Atua no ramo de Tecnologia da Informação há mais de 15 anos. É bacharel em Ciência da Computação, pela UFPE. Possui certificação OCP – Oracle Certified Professional, além de ter realizado diversos cursos na área. Também trabalha com os bancos de dados Sql Server e MySQL. Atua como Gerente de Banco de Dados na empresa Linx na unidade Recife/PE.



Links:

Plug into the Cloud with Oracle Database 12c

<http://www.oracle.com/technetwork/database/plug-into-cloud-wp-12c-1896100.pdf>

Oracle Multitenant

<http://www.oracle.com/us/products/database/options/multitenant/overview/index.html>

Você gostou deste artigo?

Dê seu voto em www.devmedia.com.br/sqlmagazine/feedback

Ajude-nos a manter a qualidade da revista!



Alta disponibilidade no banco de dados Oracle

– Parte 3

Soluções para paradas não planejadas no Oracle



ESTE ARTIGO FAZ PARTE DE UM CURSO

Situações que forçam uma parada do banco de dados sempre são problemáticas, principalmente quando esta parada não era esperada, ou seja, não foi devidamente planejada. E por mais que tentemos eliminar esta possibilidade, ela sempre existirá e nos surpreenderá.

A grande questão é estar preparado para o momento que essa parada acontecer, independentemente de ter sido planejada ou não. E estar preparado para esta situação significa garantir que o sistema de banco de dados não fique indisponível, ou que fique indisponível o menor tempo possível/tolerado.

Conhecer e implementar as funcionalidades disponíveis é essencial para garantir que não haja surpresas. Neste artigo abordo o Oracle GoldenGate e Oracle Streams, ferramentas da Oracle para replicação de dados. Oracle Flashback é uma tecnologia muito interessante que permite “enxergar o passado” e, com isso, recuperar informações e devolver a aplicação a um estado consistente.

Já o ASM - Automatic Storage Management – oferece uma bela infraestrutura para o armazenamento dos dados com várias funcionalidades de proteção aos dados como redundância, por exemplo.

A utilização da Fast Recovery Area é uma política bastante interessante para que a recuperação do sistema de banco de dados seja feita de maneira rápida e, em conjunto com o RMAN – Recovery Manager – permite que o impacto da parada seja o menor possível.

O Data Recovery Advisor também é uma boa ferramenta que auxilia bastante o DBA nas tarefas preventivas.

Resumo DevMan

Porque este artigo é útil:

Situações que forçam uma parada do banco de dados sempre são problemáticas. A grande questão é estar preparado para o momento que essa parada acontecer. O artigo trata de técnicas de utilização do banco de dados Oracle em ambientes de alta disponibilidade, mais especificamente para garantir alta disponibilidade do banco de dados em paradas não planejadas. A discussão deste artigo é útil uma vez que algumas situações não previstas podem provocar uma parada não planejada no ambiente de banco de dados e, para algumas aplicações críticas, este tempo de parada não planejada pode ocasionar grandes perdas financeiras. Para minimizar ao máximo este tipo de situação é que este tema é útil.

Ele analisa os discos procurando por falhas de dados e apresenta possíveis soluções dos problemas que podem ou não ser executadas pelo DBA.

Já o Secure Backup oferece uma bela solução de backup centralizado utilizando as famosas fitas. É compatível com vários ambientes e sistemas de arquivos.

As Security Features auxiliam o DBA a restringir acesso aos dados garantindo que cada usuário consiga manipular apenas os dados que ele realmente possa manipular.

E finalmente apresento o LogMiner, uma ótima ferramenta que possibilita que o DBA tenha acesso a todas as informações contidas no redo log e também nos archived redo log. Todas as informações referentes a cada transação que tenha ocorrido no banco de dados estarão contidas nestes arquivos e, através do LogMiner o DBA poderá “garimpar” estas informações.

Em edições anteriores da SQL Magazine este tema foi discutido e alguns dos assuntos apresentados foram:

- Fast-Start Fault Recovery;
- Oracle Restart;

- Oracle Real Application Clusters e Oracle Clusterware;
- Oracle RAC One Node;
- Oracle Data Guard.

Neste artigo daremos continuidade ao curso, desta vez, as seguintes funcionalidades/ferramentas serão discutidas:

- Oracle *GoldenGate* e Oracle *Streams*;
- Oracle *Flashback*;
- Oracle Automatic Storage Management;
- Fast Recovery Area;
- Recovery Manager;
- Data Recovery Advisor;
- Oracle Secure Backup;
- Oracle Security Features;
- LogMiner.

São várias as situações em que paradas não previstas acontecem. Da mesma forma, várias soluções para mitigar estes problemas também estão à disposição. É muito importante conhecer estas soluções para escolher de maneira acertada a que melhor se adapte à realidade da empresa e ao nível de tolerância que se tem.

Oracle Golden Gate e Oracle Streams

Oracle *GoldenGate* é um produto da Oracle para a distribuição e integração de dados. É um *software* de alto desempenho que usa a replicação bidirecional de dados baseado em *log* para captura em tempo real, transformação, roteamento e entrega de transações de banco de dados entre sistemas heterogêneos, ou seja, entre bancos de dados diferentes. O Oracle *Golden Gate* ajuda a garantir uma disponibilidade contínua e integração em tempo real para os dados de missão crítica.

Oracle *Streams* é também um recurso de replicação de banco de dados muito flexível e poderoso que a Oracle continuará a suportar, mas não fará grandes melhorias em versões futuras. É possível continuar a usar uma implementação do Oracle *Streams* existente para maximizar o retorno sobre o investimento (ROI – *Return On Investment*). No entanto, é preciso considerar o Oracle *Golden Gate* como a estratégia de replicação de longo prazo para a organização.

A Oracle investirá em melhorias do Oracle *Golden Gate* com muitos dos melhores recursos do Oracle *Streams* para fornecer uma solução de distribuição de informações com o melhor dos dois mundos.

O Oracle *Golden Gate* é um produto de replicação de dados em tempo real, assíncrono, baseado em *log*, que move grandes volumes de dados transacionais em tempo real através de banco de dados, hardware e ambientes de sistema operacional heterogêneos com o mínimo de impacto.

Um ambiente típico inclui um processo de captura, extração e entrega. Cada um desses processos pode ser executado na maioria dos sistemas operacionais e bancos de dados, incluindo bancos de dados Oracle Database e não Oracle. Todos os dados ou uma porção deles podem ser reproduzidos, e os dados dentro

de qualquer um destes processos podem ser manipulados, não só para ambientes heterogêneos, mas também diferentes esquemas de bancos de dados.

Oracle *Golden Gate* suporta replicação *multi-master* e transformação de dados. Desta forma o Oracle *Golden Gate* permite garantir que os sistemas críticos estejam em operação 24x7.

A replicação *multi-master* é um método de replicação de banco de dado que permite que os dados sejam armazenados por um grupo de computadores e atualizado por qualquer membro do grupo. Todos os membros são sensíveis às consultas de dados de clientes. O sistema de replicação *multi-master* é responsável por propagar as modificações de dados feitas por cada membro para o resto do grupo, e resolver quaisquer conflitos que possam surgir entre as mudanças simultâneas feitas por diferentes membros.

A replicação *multi-master* pode ser contrastada com a replicação de *master-slave*, na qual um único membro do grupo é designado como o “*master*” para um dado segmento de dados e é o único nó com permissão para modificar esse item de dados. Outros membros que desejam modificar o item de dados devem primeiro contatar o nó mestre. Permitir que apenas um único mestre fizesse as alterações torna mais fácil alcançar a consistência entre os membros do grupo, mas é menos flexível do que a replicação *multi-master*.

Replicação *multi-master* também pode ser contrastada com failover clustering, onde servidores escravos passivos estão replicando os dados mestre, a fim de se preparar para assumir no caso do mestre parar de funcionar. O mestre é o único servidor ativo para interação com o cliente.

Os principais objetivos da replicação *multi-master* são o aumento da disponibilidade e menor tempo de resposta do servidor. Podemos apontar as seguintes vantagens:

- Se um nó mestre falhar, outros mestres continuam a atualizar o banco de dados;
- Nós mestre podem estar localizados em vários locais físicos, isto é, distribuídos através da rede.

Em contraponto, podemos apontar as seguintes desvantagens:

- A maioria dos sistemas de replicação *multi-master* são pouco consistentes, ou seja, violam propriedades ACID;
- Sistemas de replicação com uma carga muito alta de dados são complexos e aumentam a latência de comunicação;
- Questões como a resolução de conflitos podem se tornar intratáveis quando o número de nós envolvidos aumenta, aumentando também a latência.

O Oracle *Golden Gate* também é um excelente método para minimizar o tempo de inatividade durante a manutenção planejada, incluindo atualizações de aplicativos e banco de dados, além de migrações de plataforma. Ele otimiza o acesso à informação em tempo real e garante a alta disponibilidade pelos seguintes motivos:

- Suporta replicação envolvendo uma mistura heterogênea de banco de dados Oracle e bancos de dados não-Oracle;

Alta disponibilidade no banco de dados Oracle – Parte 3

- Mantém a disponibilidade contínua para sistemas de missão crítica através de:
 - Recuperação de desastres e proteção de dados: cria e mantém um *site de failover* imediato com dados sincronizados para minimizar o tempo de recuperação dos sistemas de missão crítica;
 - Operações com tempo de inatividade zero: permite operações ininterruptas durante as atualizações do sistema, migração e atividades de manutenção;
 - Distribuição de dados: sincroniza os dados para aplicações distribuídas em tempo real para melhorar a disponibilidade e escalabilidade;
- Permite a integração de dados em tempo real em toda a empresa através de:
 - Data Warehouse em tempo real: fornece captura e entrega dos dados mais recentes de mudança ocorridas entre os sistemas OLTP e Data Warehouse em tempo real;
 - Integração de dados operacionais: integra dados operacionais entre sistemas OLTP em tempo real.

O Oracle *Golden Gate* e Oracle *Active Data Guard* são funcionalidades estratégicas dentro do portfólio de *software* da Oracle e são complementares entre si. Quando utilizados em conjunto, Oracle *Golden Gate* e Oracle *Active Data Guard* oferecem uma proteção de dados única e solução de distribuição de informação muito boas.

Embora esses recursos geralmente se enquadrem na categoria de tecnologias de replicação, cada um tem uma área muito diferente de foco. O Oracle *Active Data Guard* é o produto estratégico da Oracle para a proteção de dados e recuperação em caso de desastres para o banco de dados Oracle.

É um superconjunto da funcionalidade do *Data Guard* incluído na versão Oracle *Database Enterprise Edition* portanto, o *Active Data Guard* também herda todas as funcionalidades do *Data Guard*:

- Operação transparente em todos os tipos de dados, atributos de armazenamento, DML e DDL;
- Simplicidade de gestão;
- Replicação unidirecional simples de todo o banco de dados;
- Proteção contra corrupção;
- Escolha entre proteção assíncrona ou síncrona (zero perda de dados);
- Alta disponibilidade durante eventos não planejados através de *failover* automático de banco de dados e de cliente;
- Tempo de inatividade minimizado através da implementação de atualizações do banco de dados, sistema e manutenção do site, ou atualização de tecnologia através de execução em modo *rolling fashion* entre bancos de dados primários e de *standby*.

O Oracle *Active Data Guard* exige uma licença separada e só pode ser usado com a versão Oracle *Database Enterprise Edition*. Também está incluído no Oracle *Golden Gate*.

Funcionalidades básicas do *Data Guard* não exigem uma licença separada, pois estão incluídas na versão Oracle *Enterprise Edition*.

Já o Oracle *Golden Gate* é um produto estratégico da Oracle para a distribuição e integração de dados.

O Oracle *Golden Gate* complementa o Oracle *Active Data Guard* com as suas capacidades de replicação heterogênea e bidirecional para permitir distribuição de dado, atualizações e migrações com tempo de inatividade zero, consulta em sistemas heterogêneos e soluções de banco de dados múltiplos.

Dependendo do cenário apresentado, pode-se escolher entre:

- Oracle *Active Data Guard* para uma solução simples de: alta performance, recuperação em caso de desastres, proteção de dados e alta disponibilidade;
- Oracle *Golden Gate* para a criação de uma solução de distribuição e sincronização de dados através da configuração de replicação Oracle-Oracle, ou uma solução mais flexível de alta disponibilidade para ambiente *multi-master*.

A **Figura 1** mostra uma configuração em que um banco de dados *standby* físico em *Data Guard* garante proteção de dados e acesso somente leitura da carga de trabalho do banco de dados primário. Também garante uma replicação heterogênea de vários subconjuntos de dados do banco de dados de produção para vários bancos de dados de destino. Ao invés de utilizar replicação através do *Golden Gate* no banco de dados de produção, o processo de captura do *Golden Gate* é transferido para o banco de dados *standby* físico, onde todas as alterações são capturadas a partir de *redo logs* e replicadas, evitando assim sobrecarga de processamento do *Golden Gate* no banco de dados de produção.

Uma informação bastante relevante é que o Oracle *Golden Gate* é um produto vendido de forma independente do banco de dados Oracle tanto para sistemas de gerenciamento de banco de dados de terceiros quanto bancos de dados Oracle. Está disponível tanto para a versão Oracle *Database Enterprise Edition* quanto para a versão Oracle *Database Standard Edition* e a licença para o Oracle *Golden Gate* inclui uma licença para o Oracle *Active Data Guard*.

O *Golden Gate* também é um excelente método para minimizar o tempo de inatividade durante uma manutenção planejada, incluindo atualizações da aplicação e do banco de dados e migrações de plataforma.

Oracle Flashback

A tecnologia de *flashback* oferece um conjunto de recursos que permitem exibições dos dados em diferentes pontos no tempo. Usando os recursos de *flashback* é possível consultar versões anteriores de objetos de esquema e dados históricos. Também é possível realizar a análise de mudanças e reparação de dados devido à corrupção lógica mesmo com o banco de dados *online*.

Oferece ainda uma interface SQL para analisar e reparar os erros humanos rapidamente. Permite uma análise refinada e reparação de dados localizados, como excluir o pedido do cliente errado. A tecnologia *flashback* também permite a correção de dados

mais generalizados de maneira rápida para evitar longo tempo de inatividade. Esta tecnologia é exclusiva do banco de dados Oracle e através dela é possível efetuar uma recuperação de todos os níveis, incluindo linha, transação, tabela, *tablespace* e até mesmo todo o banco de dados.

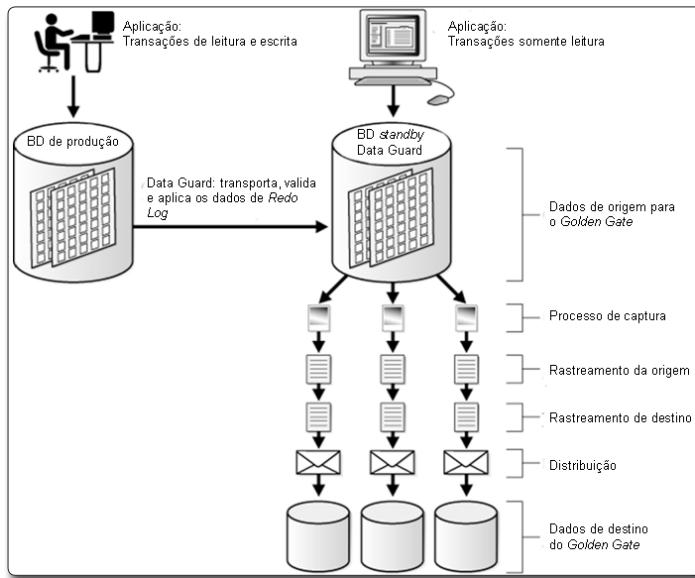


Figura 1. Estrutura de utilização de Data Guard e Golden Gate em paradas não planejadas

A maioria dos recursos de *flashback* utilizam dados de *undo*. Quando uma transação modifica os dados, o banco de dados Oracle copia os dados originais antes de modificá-lo. A cópia original dos dados modificados é chamado de dados de *undo*. Armazenar esta informação é importante pelas seguintes razões:

- Para desfazer as alterações não confirmadas feitas no banco de dados caso um *rollback* seja necessário. A reversão pode ser necessária porque um usuário quer desfazer as alterações de uma transação equivocada ou não intencional, ou pode ser parte de uma operação de recuperação;
- Para garantir consistência de leitura, o que significa que cada usuário pode obter uma visão consistente dos dados, mesmo quando outras mudanças possam ocorrer em relação aos dados. Com consistência de leitura, uma sessão de usuário não vê alterações não confirmadas feitas em outras sessões do usuário (conhecido como leituras sujas). Por exemplo, se um usuário executa uma consulta às 10:00 e a consulta tem a duração de 15 minutos, o resultado da consulta deve refletir o estado dos dados exatamente às 10:00, independentemente das operações de atualização ou inserção realizadas por outros usuários depois do começo da consulta;
- Para ativar algumas funcionalidades do Oracle *Flashback*, como o *flashback* de consulta e *flashback* de tabelas, que permitem que os dados sejam visualizados ou recuperados a partir de um ponto anterior no tempo.

Com o gerenciamento automático de *undo*, os dados de *undo* são armazenados em uma *tablespace* de *undo*. *Tablespaces* de *undo* têm

propriedades adicionais além daquelas de *tablespaces* permanentes. Pode haver vários *tablespaces* de *undo*, mas apenas um pode estar ativo para uma instância de banco de dados Oracle.

O banco de dados Oracle automaticamente garante que os dados de *undo* que estejam em uso por uma transação ativa nunca sejam substituídos até que a transação seja confirmada. Após a transação ter sido confirmada, o espaço ocupado pelos dados de *undo* pode ser reutilizado ou substituído. Neste caso, os dados de *undo* poderiam ser substituídos se o espaço no *tablespace* de *undo* se tornar insuficiente.

Mesmo depois de uma transação ter sido confirmada, é útil manter (e não substituir) os dados de *undo* para garantir o sucesso de recursos como Oracle *Flashback* e consistência de leitura para consultas de longa duração. Para este fim, o banco mantém e ajusta automaticamente um período de retenção de dados de *undo*. Os dados de *undo* de transações confirmadas (*commit*), cuja tempo de existência é inferior ao período de retenção, são mantidos para uso por consultas ou operações de *flashback*. Este período de retenção é definido pelo parâmetro *UNDO_RETENTION*.

Já outras funcionalidades (como *flashback* do banco de dados e *Block Media Recovery*) usam os *logs* de *flashback*:

- *Tablespace de Undo*: uma *tablespace* dedicada apenas ao armazenamento de informações de *undo* quando o banco de dados está em operação no modo de gerenciamento automático de *undo*;
- *Flashback Data Archive*: um arquivamento que é armazenado em um *tablespace* e contém todas as alterações efetuadas por cada transação para cada registro em uma tabela com duração igual ao tempo de vida útil do registro. Os dados arquivados podem ser mantidos por muito mais tempo do que o período de retenção oferecido por um *tablespace* de *undo*;
- *Flashback logs*: registros gerados pelo Oracle que são utilizados para a realização de operações de *flashback* de banco de dados ou *block media recovery*. O banco de dados só pode gravar *flashback logs* na área de recuperação rápida (conhecida como *Fast Recovery Area*). *Flashback logs* são escritos sequencialmente e não são arquivados. Eles não podem ser copiados para disco.

As funcionalidades de *flashback* são:

- *Flashback de consulta (flashback query)*;
- *Flashback de versão da consulta (flashback version query)*;
- *Flashback de transação (flashback transaction)*;
- *Flashback de consulta à transação (flashback transaction query)*;
- *Flashback de tabela (flashback table)*;
- *Flashback de exclusão (flashback drop)*;
- *Flashback a ponto de restauração (flashback restore point)*;
- *Flashback de banco de dados (flashback database)*;
- *Block media recovery usando flashback logs*;
- *Flashback data archive*.

Flashback Query

Flashback query permite a visualização dos dados tal como existiam no passado, usando o sistema de gerenciamento automático de *undo* para obter metadados e dados históricos das transações.

Os dados de *undo* são persistentes e resistem a um mau funcionamento do banco de dados ou mesmo um *shutdown*. A única funcionalidade que o *Flashback Query* não oferece é a possibilidade de consultar as versões anteriores de uma tabela, mas também oferece um poderoso mecanismo para recuperação de operações erradas.

As principais utilizações de *Flashback Query* incluem:

- Recuperação de dados perdidos ou incorretos de alterações confirmadas. Por exemplo, registros que foram apagados ou atualizados podem ser imediatamente reparados, mesmo depois de terem sido confirmados (*commit*);
- Comparação de dados atuais com os dados correspondentes em algum momento no passado. Por exemplo, por meio de um relatório diário que mostra as mudanças nos dados de ontem, é possível comparar os registros individuais de dados de tabela ou encontrar interseções ou uniões de conjuntos de registros;
- Verificação do estado de dados transacionais em um determinado momento, como verificar o saldo da conta em um determinado dia;
- Simplificação do projeto da aplicação, removendo a necessidade de armazenar certos tipos de dados temporais. Usando um *Flashback Query* é possível recuperar os dados passados diretamente do banco de dados.

Flashback Version Query

Flashback Version Query é uma extensão da SQL que pode ser usada para recuperar as versões de registros em uma determinada tabela que existia em um intervalo de tempo específico. *Flashback Version Query* retorna uma linha para cada versão do registro que existia no intervalo de tempo especificado. Para qualquer tabela, uma nova versão do registro é criada cada vez que a instrução *COMMIT* é executada.

Flashback Version Query é uma poderosa ferramenta que os administradores de banco de dados (DBAs) podem usar para análise e determinação da fonte de problemas. Além disso, os desenvolvedores de aplicativos podem usar a ferramenta para criar aplicativos personalizados para fins de auditoria.

Flashback Transaction

Flashback Transaction reverte uma transação e todas as suas transações dependentes. O procedimento DBMS_FLASHBACK.TRANSACTION_BACKOUT() reverte uma transação e suas operações dependentes enquanto o banco de dados permanece *online*.

Esta operação de recuperação usa dados de *undo* para criar e executar as transações de compensação que retornam os dados afetados ao seu estado original. Pode-se consultar a *view* DBA_FLASHBACK_TRANSACTION_STATE para ver se a transação foi revertida utilizando regras de dependência ou forçado por meio de:

- Reversão dos registros não-conflitantes;
- Execução de DMLs para reverter os registros afetados.

Flashback Transaction do Oracle aumenta a disponibilidade durante uma recuperação lógica através da rápida reversão de uma

transação ou um conjunto de transações e suas transações dependentes. É possível usar o comando para reverter as transações mesmo enquanto o banco de dados permanece *online*.

Flashback Transaction Query

O *Flashback Transaction Query* fornece um mecanismo para visualizar todas as alterações feitas no banco de dados no nível da transação. Quando usado em conjunto com o *Flashback Version Query*, oferece um meio rápido e eficiente para recuperação de erros humanos ou erro de aplicação. O *Flashback Transaction Query* aumenta a capacidade de realizar diagnóstico de problemas de maneira *online* no banco de dados informando qual o usuário do banco de dados que executou a alteração do registro e executa análises e auditorias sobre as transações.

Flashback Table

O *Flashback Table* recupera uma tabela para um ponto anterior no tempo. Fornece uma solução rápida e *online* para a recuperação de uma tabela ou conjunto de tabelas que tenham sido modificadas por um erro humano ou erro de aplicação.

Na maioria dos casos o *Flashback Table* evita a necessidade de administradores terem que executar operações de recuperação do tipo *point-in-time* mais complicados. Os dados da tabela original não são perdidos quando se usa o *Flashback Table* pelo fato de que se pode retornar a tabela ao seu estado original.

Flashback Drop

Exclusão de objetos no banco de dados por acidente é um problema tanto para os usuários de banco de dados quanto para os administradores de banco de dados.

Embora não haja nenhuma maneira fácil de recuperar tabelas eliminadas, índices, restrições (*constraints*) ou gatilhos (*triggers*), o *Flashback Drop* fornece proteção quando se está excluindo objetos.

Ao excluir uma tabela, ela é automaticamente colocada na lixeira (*Recycle Bin*). A lixeira é um recipiente virtual onde todos os objetos descartados permanecem. É possível continuar a consultar dados em uma tabela descartada. Ela é, na verdade, uma tabela de dicionário de dados contendo informações sobre os objetos descartados. Tabelas eliminadas e quaisquer objetos associados, tais como índices, restrições, tabelas aninhadas não são removidos e ainda ocupam espaço. Eles continuam contando nas cotas de espaço do usuário, até que seja removido da lixeira, ou seja, expurgado pelo banco de dados devido a restrições de espaço nas tablespaces.

Cada usuário pode ser considerado como tendo sua própria lixeira porque, a menos que um usuário tem o privilégio SYSDBA, os únicos objetos que o usuário tem acesso na lixeira são aqueles que o usuário possui.

Um usuário pode ver seus objetos na lixeira usando a seguinte instrução:

```
SELECT * FROM RECYCLEBIN;
```

Quando você remove um *tablespace* incluindo o seu conteúdo, os objetos no *tablespace* não são colocados na lixeira e o banco de dados limpa todas as entradas na lixeira para objetos localizados naquele *tablespace*. O banco de dados também expurga todas as entradas da lixeira para os objetos em um *tablespace* mesmo você não incluindo o conteúdo e o *tablespace* estar vazio. Da mesma forma:

- Quando você exclui um usuário, todos os objetos pertencentes ao usuário não são colocados na lixeira e quaisquer objetos deste usuário que estejam na lixeira são removidos;
- Quando você exclui um cluster, as tabelas que usam este cluster não são colocadas na lixeira e todas as tabelas antigas na lixeira que usavam este cluster são removidas;
- Quando você exclui um tipo, todos os objetos dependentes, como subtipos não são colocados na lixeira e quaisquer objetos antigos dependentes na lixeira são removidos.

Quando uma tabela excluída é movida para a lixeira, a tabela e seus objetos associados recebem nomes gerados pelo sistema. Isso é necessário para evitar conflitos de nomes que podem surgir se várias tabelas possuem o mesmo nome. Isso pode ocorrer nas seguintes circunstâncias:

- Um usuário apaga uma tabela, recria com o mesmo nome, em seguida exclui de novo;
- Dois usuários têm tabelas com o mesmo nome, e ambos os usuários excluíram suas tabelas.

A convenção utilizada para renomear o objeto é o descrito abaixo:

BIN\$id_unico\$versão

onde:

- id_único: é um identificador único de 26 caracteres exclusivo para esse objeto, o que torna o nome na lixeira único em todo o bancos de dados;
- versão: é um número de versão atribuído pelo banco de dados.

Flashback Restore Point

Quando uma operação de recuperação através de *Flashback* é executada no banco de dados, o DBA deve determinar o ponto no tempo, identificado pelo número de alteração do sistema (SCN – *System Change Number*) ou o *timestamp* (sequência de caracteres ou de informação codificada que identifica quando um determinado evento ocorreu, geralmente fornecendo a data e hora do momento do evento, às vezes chegando até uma pequena fração de segundo), para então recuperar os dados para aquele momento do tempo. *Flashback Restore Points* são rótulos que se pode definir para substituir o SCN ou *timestamp* utilizado para o *Flashback Database*, *Flashback Table* e operações do *Recovery Manager* (RMAN).

Além disso, um banco de dados pode ser revertido através de uma recuperação de banco de dados anterior e aberto como com

um comando OPEN RESETLOGS usando os pontos de restauração. Estes pontos de restauração permitem grandes alterações em bancos de dados como processos em lote, atualizações ou correções que podem ser rapidamente desfeitas garantindo que os dados de *undo* necessários para recuperar o banco de dados estejam mantidos.

Usar as funcionalidades do *Flashback Restore Point* garante os seguintes benefícios:

- A capacidade de restaurar rapidamente a um estado consistente, a um tempo antes de uma operação planejada que deu errado (por exemplo, um processo em lote que falhou, uma atualização de software Oracle ou uma atualização do aplicativo que apresentaram problema);
- A capacidade de resincronizar o banco de dados *snapshot standby* com o banco de dados primário;
- Um mecanismo rápido para restaurar um banco de dados de teste ou clonado para seu estado original.

Flashback Database

O *Flashback Database* fornece uma alternativa mais eficiente para a recuperação *point-in-time* do banco de dados. Com o *Flashback Database* é possível reverter os arquivos de dados atuais para o seu conteúdo em um tempo passado. O resultado é muito parecido com a restauração de dados a partir de *backups* de arquivos de dados e execução de recuperação de dados a um determinado momento no tempo (*point-in-time recovery*). No entanto, o *Flashback Database* evita a necessidade de restauração de arquivos de dados e aplicação de dados de *redo*.

Ativar o *Flashback Database* garante os seguintes benefícios:

- A capacidade de eliminar o tempo para restaurar um *backup* quando é necessário reparar um erro humano que tenha impactado o banco de dados de forma ampla;
- A capacidade de desfazer rapidamente os erros humanos utilizando o *real time apply* para sincronizar o banco de dados *standby* com o banco de dados primário.
- A capacidade de reinstanciar rapidamente o banco de dados *standby* após um *failover* do banco de dados.

O *failover* é o “chaveamento” para um sistema, servidor, componente de *hardware* ou rede redundante ou de espera (*standby*) devido à falha ou término anormal do aplicativo, servidor, sistema, componente de *hardware* ou rede ativo anteriormente. *Failover* e “chaveamento” (*switchover*) são, essencialmente, a mesma operação, só que o *failover* é automático e normalmente opera sem aviso, enquanto que o “chaveamento” requer intervenção humana.

Projetistas de sistemas normalmente fornecem capacidade de *failover* em servidores, sistemas ou redes que exigem disponibilidade contínua, o termo utilizado é de alta disponibilidade e um alto grau de confiabilidade.

No nível do servidor, automação por *failover* geralmente usa um cabo “*heartbeat*” que conecta dois servidores. Enquanto um “pulso” regular ou “batimento cardíaco” continua ativo entre o servidor principal e o segundo servidor, o segundo servidor

não iniciará seus sistemas. Também pode haver um terceiro servidor “sobressalente” com componentes sobressalentes para executar uma mudança “a quente” para evitar tempo de inatividade. O segundo servidor retoma o trabalho do primeiro logo que se detecta uma alteração na “pulsação” da primeira máquina. Alguns sistemas têm a capacidade de enviar uma notificação de failover.

Em 1992, em coordenação com a *Distributed Processing Technology* e *Black Box Cables*; Darryl Brown desenvolveu uma configuração *Failover PC Server* que permitiu o *failover* de sistemas espelhados ou subsistemas RAID 1 ativamente conectados a dois servidores. A configuração permitiu que um servidor de produção executasse o *failover* em poucos minutos no momento de uma falha de *hardware* do servidor, simplesmente ao mudar manualmente para o servidor redundante com um toque de um botão.

Alguns sistemas, intencionalmente, não executam o *failover* de forma totalmente automática, mas necessitam de intervenção humana. Esta configuração de “automatização com aprovação manual” é executada automaticamente quando um ser humano tenha aprovado o *failover*.

Failback é o processo de restauração de um sistema, componente ou serviço em um estado de *failover* de volta ao seu estado original (antes da falha).

O uso de *software* de virtualização permitiu práticas de *failover* para se tornar menos dependente do *hardware* físico.

Block Media Recovery usando Logs de Flashback

Após tentar reparar automaticamente blocos corrompidos, *block media recovery* pode, opcionalmente, recuperar uma cópia mais recente de um bloco de dados a partir de *logs de flashback* para reduzir o tempo de recuperação. Reparação automática de blocos permite que blocos corrompidos no banco de dados principal sejam reparados automaticamente assim que são detectados, utilizando blocos íntegros a partir de um banco de dados *standby* físico.

Além disso, um bloco corrompido encontrado durante a recuperação de instância não resulta em falha na recuperação de instância. O bloco é automaticamente marcado como corrompido e adicionado à lista de corrupção do RMAN, que pode ser encontrado na *view V\$DATABASE_BLOCK_CORRUPTION*. É possível, posteriormente, executar o comando RECOVER BLOCK do RMAN para restaurar a integridade do bloco associado. Além disso, esse comando restaura os blocos com base num banco de dados *standby* físico caso esteja disponível.

Flashback Data Archive

O *Flashback Data Archive* é armazenado em uma *tablespace* e contém todas as alterações transacionais de cada registro em uma tabela durante toda a vida útil deste registro. Os dados arquivados podem ser retidos por um período muito mais longo do que o período de retenção oferecido por um *tablespace* de *undo*.

Automatic Storage Management - ASM

O ASM é um sistema de arquivos verticalmente integrado e um gerenciador de volume diretamente no kernel do banco de dados Oracle, resultando em:

- Muito menos trabalho para fazer a provisão de recursos de armazenamento para o banco de dados;
- Maior nível de disponibilidade;
- Eliminação da despesa com instalação e manutenção de produtos de armazenamento especializados;
- Capacidades únicas para aplicações de banco de dados.

Para garantir um bom desempenho o ASM distribui os arquivos de dados ao longo de toda a infraestrutura de armazenamento disponível. Para se proteger contra perda de dados, o ASM estende o conceito de SAME (*Stripe and Mirror Everything* – Distribuição e Espelhamento de Tudo) e adiciona mais flexibilidade, pois pode refletir no nível de arquivo de banco de dados ao invés de refletir no nível do disco inteiro.

Mais importante ainda, o ASM simplifica os processos de configuração do espelhamento através da simples adição e remoção de discos. Ao invés de gerenciar centenas ou talvez milhares de arquivos (comuns em um grande *data warehouse*), o DBAs utiliza o ASM para criar e administrar um único objeto maior chamado de grupo de discos (*disk groups*). O grupo de discos identifica o conjunto de discos que são gerenciados como uma unidade lógica. Automação de nomeação de arquivos e localização dos arquivos de banco de dados garante aos administradores melhor gerenciamento de tempo e garantem a adesão às melhores práticas.

O mecanismo de espelhamento nativo do ASM (duas vias ou três vias) protege contra falhas de armazenamento. Com o espelhamento do ASM é possível fornecer um nível adicional de proteção de dados com o uso de grupos de falha (*fail groups*). Um grupo de falha é um conjunto de discos que compartilham um recurso comum (controlador de disco ou um *disk array* inteiro) que não pode ter uma falha tolerada. Depois de definido, um grupo de falha do ASM inteligentemente coloca cópias redundantes dos dados em grupos de falha separados. Isto assegura que os dados estejam disponíveis e de forma transparente protegidos contra falhas de qualquer componente no subsistema de armazenamento.

Usando o ASM é possível:

- Efetuar a distribuição e espelhamento ao longo das unidades e *arrays* de armazenamento;
- Automaticamente efetuar o re-espelhamento de uma unidade que falhou nas unidades restantes;
- Reequilibrar automaticamente os dados armazenados quando discos são adicionados ou removidos, enquanto o banco de dados permanece *online*;
- Suportar arquivos de banco de dados e arquivos diversos utilizando o sistema de arquivos ASM Cluster (Oracle ACFS – ASM Cluster File System);
- Permitir a simplicidade operacional no gerenciamento de armazenamento de dados;

- Gerenciar o Oracle *Cluster Registry* (OCR) e discos de *voting* (*voting disks*);
- Fornecer capacidade de leitura preferencial em discos que estejam fisicamente localizados mais perto do servidor em que se encontra a instância, o que dá um melhor desempenho para um *cluster* estendido;
- Suportar grandes bancos de dados;
- Suportar atualizações do ASM sem interrupção;
- Suportar granularidade fina em *tuning* e segurança;
- Prestar serviços de reparação rápida após uma falha de disco temporária por meio do ASM *Fast Mirror Resync* e reparação automática de blocos corrompidos caso haja uma cópia íntegra do bloco em um dos espelhos;
- Fornecer capacidade de recuperação de desastres para o sistema de arquivos, permitindo a replicação de sistemas de arquivos ACFS através da rede para um local remoto.

Fast Recovery Area (Área de Recuperação Rápida)

A área de recuperação rápida é um local de armazenamento unificado para todos os arquivos e atividades relacionados à recuperação do banco de dados. Após esse recurso estar ativado, todos os *backups* do RMAN, arquivos de *redo log*, *autobackups* do arquivo de controle (*control file*), *logs* de *flashback* e cópias dos arquivos de dados são automaticamente gravados em um sistema de arquivo específico ou grupo de discos do ASM e o gerenciamento deste espaço em disco é feito pelo RMAN e o servidor de banco de dados.

Realizar um *backup* em disco é mais rápido porque usar a área de recuperação rápida elimina o gargalo de gravação de fita. Mais importante, se a recuperação de mídia do banco de dados é necessária, então *backups* de arquivos de dados estão prontamente disponíveis. O tempo de restauração e de recuperação é reduzido porque não é necessário encontrar uma fita e um dispositivo de fita livre para restaurar os arquivos de dados e *archived redo log files* necessários.

A área de recuperação rápida oferece os seguintes benefícios:

- Local de armazenamento unificado de arquivos relacionados à recuperação;
- Gerenciamento do espaço em disco alocado para arquivos de recuperação, o que simplifica as tarefas de administração de banco de dados;
- *Backup* e recuperação rápidos e confiáveis baseados em disco;
- Capacidade de fazer *backup* e restaurar toda a área de recuperação rápida.

Recovery Manager - RMAN

O *Recovery Manager* (RMAN) é um utilitário de banco de dados para gerenciar o *backup* do banco de dados e, mais importante, a recuperação do banco de dados.

O RMAN elimina a complexidade operacional proporcionando melhor desempenho e disponibilidade do banco de dados. O RMAN determina o método mais eficiente de executar a operação de backup, restauração, recuperação e, em seguida, envia

estas operações para o servidor de banco de dados Oracle para processamento. RMAN e o servidor identificam automaticamente as modificações na estrutura da base de dados e ajustam dinamicamente a operação necessária para se adaptarem às alterações.

RMAN oferece os seguintes benefícios:

- *Failover* automático de canais em operações de *backup* e restauração;
- *Failover* automático para um *backup* anterior quando a operação de restauração encontra um *backup* ausente ou corrompido;
- Criação automática de novos arquivos de banco de dados e arquivos temporários durante a recuperação;
- Recuperação automática através de uma recuperação *point-in-time* anterior - recuperação através de *resetlogs*;
- Recuperação de bloco, que permite que o arquivo de dados permaneça *online* enquanto o reparo do bloco é efetuado;
- *Backups* incrementais rápidos usando o controle de alterações bloco (*block change tracking*);
- Operações de *backup* e recuperação mais rápidos através de paralelismo intra-arquivos e inter-arquivos;
- Maior segurança com o catálogo virtual privado;
- Redução do consumo de espaço na criação de um banco de dados através da rede, eliminando áreas intermediárias para armazenamento dos arquivos do *backup*;
- Incorporação de *backups* incrementais em cópias de imagens (*image copies*) em segundo plano, fornecendo recuperação atualizada;
- *Backup* e restauração otimizados tratando apenas os arquivos necessários;
- Política de retenção para garantir que os *backups* relevantes sejam mantidos;
- Habilidade para reiniciar operações de *backup* ou restauração que falharam anteriormente;
- *Backup* automático do arquivo de controle (*control file*) e do arquivo de parâmetro do servidor (*spfile*), garantindo que os metadados de *backup* estejam disponíveis em momentos de mudanças estruturais no banco de dados, falhas de mídia e desastres;
- *Backup online* que não requer que seja colocado o banco de dados em modo de *backup* quente (*begin backup*).

Data Recovery Advisor

O *Data Recovery Advisor*, ou Conselheiro de Recuperação de Dados, automaticamente efetua o diagnóstico de falhas de dados persistentes (em disco), apresenta opções adequadas de recuperação e comanda as operações de recuperação caso o DBA aceite o diagnóstico.

Pode-se usar o Data Recovery Advisor para solucionar problemas em bancos de dados primários, bancos de dados *standby* lógicos ou *snapshot*.

O Data Recovery Advisor inclui as seguintes funcionalidades:

- Diagnóstico de falhas: os primeiros sintomas de falha de banco de dados geralmente são mensagens de erro, alarmes, arquivos de rastreamento (*trace*), arquivos de *dump* e falhas na verificação da saúde geral do banco de dados. Avaliar estes sintomas pode

ser complicado, propenso a erros e demorado. O *Data Recovery Advisor* automaticamente diagnostica falhas de dados e informa o DBA sobre estas falhas;

- Avaliação do impacto da falha: após uma falha ser diagnosticada é preciso compreender a sua extensão e avaliar o seu impacto sobre as aplicações antes de definir uma estratégia de recuperação. O *Data Recovery Advisor* avalia automaticamente o impacto de uma falha e exibe em um formato de fácil compreensão;

- Criação dos *scripts* de recuperação: mesmo que uma falha seja diagnosticada corretamente, a seleção da correta estratégia de recuperação também pode ser propensa a erros e estressante. Além disso, há muitas vezes um grande prejuízo para uma tomada de decisão errada em termos de aumento do tempo de inatividade e perda de dados. *Data Recovery Advisor* determina automaticamente a melhor estratégia de recuperação para uma série de falhas e apresenta para o DBA;

- Verificação da viabilidade de recuperação: antes de apresentar as opções de recuperação, o *Data Recovery Advisor* efetua uma avaliação em relação ao ambiente e disponibilidade de componentes de mídia necessários para concluir a recuperação proposta;

- Automação da recuperação: caso o DBA aceite a opção de recuperação sugerida, o *Data Recovery Advisor* executa automaticamente a recuperação e verifica se a recuperação foi bem sucedida;

- Validação da consistência dos dados e recuperação de banco de dados: o *Data Recovery Advisor* pode validar a consistência dos dados, dos *backups* e dos dados de *redo*, com base na escolha do DBA;

- Detecção precoce de corrupção: através do *Health Monitor* (monitor de saúde), é possível agendar execuções periódicas do diagnóstico de verificação do *Data Recovery Advisor* para detectar falhas de dados antes de um processo de uma transação de banco de dados descubra a corrupção e sinalize um erro. Detecções precoces podem limitar os danos causados pela corrupção;

- Integração de validação de dados e recuperação: o *Data Recovery Advisor* é uma ferramenta única para validação de dados e recuperação.

É bom saber que o *Data Recovery Advisor* apenas suporta bancos de dados de instância única. Bancos de dados em ambiente RAC não são suportados.

Secure Backup

Oracle *Secure Backup* é uma solução de gerenciamento centralizado de *backup* em fita fornecendo proteção de dados heterogêneos em ambientes UNIX, Linux, Windows e *Network-Attached Storage*. Ao proteger o sistema de arquivos e banco de dados o Oracle *Secure Backup* fornece uma solução de *backup* de fita completo para o ambiente de TI.

Network-attached storage – NAS – (Armazenamento Anexado à Rede) é um armazenamento de dados do computador em nível de arquivo conectado a uma rede de computador que oferece acesso a dados para um grupo heterogêneo de clientes. NAS não só funciona como um servidor de arquivos, mas é especializado

para esta tarefa, quer pelo seu *hardware*, *software* ou a configuração desses elementos. NAS muitas vezes é fabricado como um dispositivo de computador - um computador especializado construído a partir do zero para armazenar e servir arquivos ao invés de simplesmente um computador de uso geral a ser utilizado para esta finalidade.

A partir de 2010 os dispositivos NAS ganharam popularidade como um método conveniente de compartilhar arquivos entre vários computadores. Os benefícios potenciais de armazenamento anexado à rede, em comparação a servidores de arquivos, incluem acesso mais rápido de dados, administração mais fácil e configuração simples.

Sistemas NAS estão em aparelhos conectados à rede que contenham um ou mais discos rígidos, muitas vezes organizados em recipientes de armazenamento redundantes ou RAID. O armazenamento anexado à rede elimina a responsabilidade de servir arquivos de outros servidores na rede. Eles geralmente fornecem acesso a arquivos usando protocolos de compartilhamento de arquivos de rede como: NFS, SMB / CIFS ou AFP.

Uma unidade NAS é um computador conectado a uma rede que fornece apenas serviços de armazenamento de dados baseados em arquivos para outros dispositivos na rede. Embora possa ser tecnicamente possível executar outro *software* em uma unidade NAS, ele não foi projetado para ser um servidor de propósito geral. Por exemplo, unidades NAS geralmente não têm um teclado ou um monitor, e são controlados e configurados ao longo da rede, muitas vezes utilizando um navegador.

Um sistema operacional completo não é necessário em um dispositivo NAS, muitas vezes um sistema operacional enxuto é usado. Por exemplo, FreeNAS, uma solução NAS código aberto projetado para *hardware* PC baratos e é implementado como uma versão reduzida do FreeBSD.

Sistemas NAS contêm um ou mais discos rígidos, muitas vezes organizados em recipientes de armazenamento redundantes ou RAID.

NAS utiliza protocolos baseados em arquivos, como NFS (popular em sistemas UNIX), SMB / CIFS (*Message Block / Common Internet File System Server*) (usado com sistemas MS Windows), AFP (usado com computadores Apple Macintosh), ou NCP (usado com OES e Novell NetWare). Unidades NAS raramente limitam os clientes a um único protocolo.

O Oracle *Secure Backup* é totalmente integrado com o RMAN para fornecer a camada de gerenciamento de mídia para RMAN, suportando versões do banco de dados desde o Oracle9i. Com pontos de integração otimizada, o Oracle *Secure Backup* e o RMAN fornecem a capacidade mais rápida e eficiente de *backup* em fita para o banco de dados.

É possível fazer *backup* de servidores distribuídos para dispositivos de fita local e remoto de um servidor administrativo central do Oracle *Secure Backup* usando políticas de *backup*, agendamento baseado em operações em momentos de menor carga de trabalho, ou de *backup* sob demanda para as necessidades imediatas. Com a arquitetura cliente/servidor altamente escalável, o Oracle *Secure*

Backup oferece proteção de dados local e remoto, usando *Secure Sockets Layer* para comunicação intradomínios segura e autenticação do servidor de mão dupla.

Secure Sockets Layer (SSL) e seu sucessor *Transport Layer Security* (TLS) são protocolos criptográficos que são projetados para garantir a segurança da comunicação através da Internet. Eles usam certificados X.509 e criptografia assimétrica para assegurar a contraparte a quem eles estão conversando e para trocar a chave simétrica. Esta chave de sessão é usada para cifrar dados que fluem entre as partes. Várias versões de protocolos estão em uso generalizado em aplicações, tais como navegação na web, correio eletrônico, mensagens instantâneas e voz sobre IP (VoIP).

Como consequência da escolha de certificados X.509, certificados de autoridade e de uma infraestrutura de chave pública são necessárias para verificar a relação entre um certificado e seu proprietário, bem como para gerar, assinar e administrar a validade dos certificados. Enquanto isto pode ser mais benéfico do que a verificação das identidades através de uma rede de confiança, as divulgações de vigilância em massa de 2013 tornaram mais amplamente conhecidas que os certificados de autoridade são um ponto fraco do ponto de vista de segurança, permitindo ataques *man-in-the-middle*.

Na visão do modelo TCP/IP, SSL e TLS criptografam os dados de conexões de rede em uma subcamada inferior de sua camada de aplicação. Na equivalência ao modelo OSI, SSL/ TLS são inicializados na camada 5 (camada de sessão), então trabalha na camada 6 (camada de apresentação): primeiro a camada de sessão faz um *handshake* usando uma cifra assimétrica a fim de estabelecer as configurações de criptografia e uma chave compartilhada para aquela sessão então, a camada de apresentação codifica o restante da comunicação utilizando um código simétrico e aquela chave de sessão. Em ambos os modelos, SSL e TLS trabalham em nome da camada de transporte subjacente, cujos segmentos transportam dados criptografados.

Oracle *Secure Backup* oferece os seguintes benefícios:

- Otimização de *backup* em fita para o banco de dados Oracle, fazendo o *backup* apenas dos blocos usados e aumentando a performance do *backup* de 10% a 25%;
- Gerenciamento baseado em políticas que permitem aos administradores de *backup* controlar os *backups* mais precisamente;
- Compartilhamento dinâmico de unidades para aumentar a utilização de recursos de fita;
- Storage Area Network (SAN – ver **BOX 1**) heterogêneo, permitindo que NAS, UNIX, Windows e Linux compartilhem unidades de fita e mídia;
- *Backup* do sistema de arquivos no nível de arquivo, diretório, sistema de arquivos ou no nível de partição *raw* (*raw device* – ver **BOX 2**) com agendamento de backup completo ou incremental;
- A integração com o Oracle *Enterprise Manager*, fornecendo uma interface gráfica;
- Criptografia de *backup* em fita;
- Amplo suporte a dispositivos de fita, tanto novos quanto antigos, em ambientes SAN e SCSI;

- Suporte a *Network Data Management Protocol* (NDMP – ver **BOX 3**) para *backup* altamente eficiente de arquivos NAS;
- Modelo de licenciamento de baixo custo e escalável que reduz os custos de TI e considerações operacionais.

BOX 1. SAN – Storage Area Network

A Storage Area Network (SAN) é uma rede dedicada que dá acesso a um dispositivo de armazenamento no nível de bloco consolidado. SANs são usadas principalmente para tornar os dispositivos de armazenamento, como disk arrays, bibliotecas de fitas e jukeboxes ópticos acessíveis aos servidores de modo que os dispositivos aparecem como dispositivos conectados localmente ao sistema operacional. A SAN normalmente tem sua própria rede de dispositivos de armazenamento que geralmente não são acessíveis através da rede local por outros dispositivos. O custo e a complexidade de SANs caíram no início de 2000 para níveis que permitem a adoção mais ampla tanto em empresas pequenas quanto em ambientes comerciais de médio porte.

A SAN não fornece abstração de arquivos, apenas as operações no nível do bloco. No entanto, sistemas de arquivos construídos em cima de SANs fornecem acesso no nível de arquivo, e são conhecidos como sistemas de arquivos SAN ou sistemas de arquivos de discos compartilhados.

Os usos mais comuns de SAN incluem o fornecimento de dados transacionalmente acessados que exigem acesso de alta velocidade no nível de bloco às unidades de disco rígido, tais como servidores de e-mail, bancos de dados e servidores de arquivo de alto uso.

BOX 2. Raw Device

Um raw device (dispositivo bruto) é um tipo especial de arquivo de dispositivo de bloco que permite acessar um dispositivo de armazenamento (como uma unidade de disco rígido) diretamente, ignorando caches e buffers do sistema operacional (embora os caches de hardware ainda possam ser utilizados).

Aplicações como um sistema de gerenciamento de banco de dados podem usar dispositivos brutos diretamente, permitindo-lhes gerenciar a forma como os dados são armazenados em cache, ao invés de passar essa tarefa para o sistema operacional.

BOX 3. NDMP - Network Data Management Protocol

NDMP é um protocolo criado pelas empresas NetApp e Legato, destinado a transporte de dados entre dispositivos NAS e dispositivos de backup. Isto elimina a necessidade de transportar os dados através do próprio servidor de backup, aumentando assim a velocidade e diminuindo a carga e a partir do servidor de backup.

A maioria dos softwares de backup multiplataforma, como Bacula, Amanda, CA ARCserve, EMC NetWorker, EMC Avamar, Symantec NetBackup, Backup Exec (desde v11d), CommVault Simpana, Virtos SOS Backup, IBM Tivoli Storage Manager, Hitachi Data Systems (HDS), a Quest Software NetVault backup, Syncsort DPX, Zmanda, HP Data Protector e outros suportam este protocolo.

Oracle Security Features

A melhor proteção contra erros humanos é prevenir que aconteça. A melhor maneira de prevenir os erros humanos é restringir o acesso do usuário somente aos dados e serviços realmente necessários para executar as funções do negócio. O banco de dados Oracle possui um vasto conjunto de ferramentas de segurança para controlar o acesso aos dados de aplicativos, autenticação de usuários de banco de dados e permitir que os administradores possam conceder a estes usuários apenas os privilégios necessários para exercer as suas funções.

Além disso, o modelo de segurança do banco de dados Oracle garante a capacidade de restringir o acesso a dados no nível de linha através do *Virtual Private Database*, isolando ainda mais os usuários do banco de dados dos dados que não precisam acessar.

Oracle *Virtual Private Database* (VPD) permite aumentar a segurança no nível de linha ou de coluna. Uma política de segurança estabelece métodos para proteger um banco de dados da destruição accidental ou mal-intencionada de dados ou danos à infraestrutura de banco de dados.

VPD é útil quando as proteções de segurança como privilégios e funções não são suficientemente refinados. Por exemplo, é possível permitir que todos os usuários acessem a tabela FUNCIONARIOS, porém criar políticas de segurança que restrinjam o acesso aos dados dos empregados que sejam do mesmo departamento que o usuário que está acessando a tabela.

Essencialmente, o banco de dados adiciona uma cláusula WHERE dinâmica para uma instrução SQL executada na tabela, view ou sinônimo ao qual uma política de segurança do VPD tenha sido aplicada. A cláusula WHERE permite que apenas usuários cujas credenciais passam pela política de segurança possam acessar os dados protegidos.

O banco de dados Oracle garante os seguintes benefícios de segurança:

- Controle de autenticação para validar a identidade de entidades que utilizam redes, bancos de dados e aplicações. Sessões de rede entre bancos de dados, tais como sessões de transporte de redo, também são autenticadas;
- Controle de autorização para garantir limites ao acesso e ações ligadas por identidades de usuários de banco de dados e funções;
- Controle de acesso a objetos, garantindo proteção independentemente da entidade que tente acessar ou alterar este objeto;
- Controle de auditoria para monitorar e coletar dados sobre as atividades de banco de dados específicos, investigar atividade suspeita, impedir que usuários (ou outros) executem atividades inadequadas e detectar problemas com a autorização ou a implementação de controle de acesso;
- Gerenciamento de políticas de segurança usando perfis;
- Criptografia de dados que residem no banco de dados e backups, ou transferidos de e para os bancos de dados.

LogMiner

Arquivos de log do Oracle contêm informações úteis sobre as atividades e a história do banco de dados. Os arquivos de log contêm todos os dados necessários para executar a recuperação do banco de dados, e também gravar todas as alterações feitas aos dados e metadados no banco de dados.

O LogMiner é uma ferramenta totalmente relacional que permite que os arquivos de redo log sejam lidos, analisados e interpretado usando comandos SQL.

Ao usar o LogMiner é possível analisar os arquivos de redo log para:

- Rastreamento ou auditoria de alterações nos dados;
- Fornecimento de informações complementares para ajuste e planejamento de capacidade;
- Recuperação de informações críticas para a depuração de aplicações complexas;
- Recuperação de dados apagados;
- Fornecimento uma interface simplificada para ajudar a solucionar problemas e resolver as falhas lógicas.

Outras características do LogMiner são:

- Identificar quando a corrupção lógica no banco de dados, tais como erros no nível da aplicação, possa ter ocorrido;
- Determinar as ações necessárias para executar a recuperação de granulação fina no nível da transação;
- Fornecer o ajuste de desempenho e planejamento de capacidade através de análise de tendências;
- Analisar o comportamento do sistema e auditar o uso de banco de dados através da interface relacional do LogMiner para refazer os arquivos de log.

Conclusão

Neste artigo apresentamos algumas funcionalidades/ferramentas que podem auxiliar o DBA para minimizar ou, até mesmo, eliminar o tempo de inatividade do sistema de banco de dados em situações não previstas, ou seja, paradas não planejadas.

Acredito que muitos de vocês já passaram por situações como uma queda de energia no site principal em que o no-break não “aguentou” ou não estava com a manutenção “em dia” e falhou.

Infelizmente situações como esta são mais comuns do que parece. E não adianta o DBA ficar tranquilo por pensar que a performance do banco de dados está boa e não há ninguém reclamando, pois tenha certeza que quando o sistema estiver inoperante, este mesmo DBA será o primeiro a ser chamado.

Através do Oracle GoldenGate e Oracle Streams a solução de replicação de dados está garantida e, numa situação de perda total do site principal, a replicação poderá ser utilizada para assumir as funcionalidades do sistema e reestabelecer todos os serviços, evitando assim um tempo de inatividade muito alto.

Já o Oracle Flashback dá uma grande versatilidade ao DBA para reparar alterações executadas erroneamente no banco de dados por parte dos usuários (sim, isso acontece com certa frequência) sem a necessidade de restaurar e recuperar todo um backup para “trazer de volta” uma única tabela para um momento anterior à alteração. Basta utilizar o flashback e consultar a tabela neste momento do passado e reverter as alterações. Mas o flashback vai ainda além, pois caso um patch ou upgrade falhe, é possível voltar todo o banco de dados a um momento anterior à manutenção através desta funcionalidade.

O ASM também ajuda muito em situações de parada, pois oferece uma infraestrutura muito boa para o armazenamento dos dados além de oferecer um bom ganho de performance para a aplicação como um todo. Escrevi uma série de artigos “destrinchando” o ASM, vale a pena procurar.

Temos ainda a *Fast Recovery Area* que, em conjunto com o RMAN garante um impacto bem pequeno na hora de recuperar um banco de dados, pois elimina a necessidade de buscar os dados de *backup* em fitas e tudo o mais.

E que tal ter um conselheiro monitorando possíveis falhas? Bem, é exatamente isso que o *Data Recovery Advisor* faz e, além de monitorar estas possíveis falhas, recomenda possíveis soluções.

Imaginando um ambiente de muitos servidores de bancos de dados com diversas instâncias Oracle e muita informação para gerenciar, uma solução centralizada de *backup* pode ser uma boa alternativa, pois nenhuma recuperação é possível sem que haja um bom *backup*. O *Secure Backup* é esta solução, pois permite armazenar os *backups* em fita e é compatível com vários ambientes operacionais e sistemas de arquivos.

Mas, por mais que tenhamos boas soluções implementadas para restaurar os serviços no menor tempo possível, é sempre melhor conseguir evitar que as causas dessa possível parada aconteçam e, para isso, as *Security Features* estão aí para dar um grande poder ao DBA para restringir acesso aos dados por parte do usuário permitindo-lhes acessar os dados que realmente necessitam.

E por último, apresentamos o *LogMiner*, uma ótima ferramenta que permite investigar os *redo logs* e *archived redo logs* para encontrar as transações que efetuaram o que não deveria ser feito, permitindo ao DBA desfazer o que precisa ser desfeito.

Autor



Ricardo Rezende

e-mail: ricarezende@gmail.com

Blog: <http://www.devmedia.com.br/ricardorezende>

Blog: <http://purl.org/ricarezende/blog>

Twitter: <http://twitter.com/ricarezende>



DBA Oracle certificado pela Oracle University (DBA 9i track e DBA OCP 10g). IBM Certified Database Associate – DB2 9. DBA Oracle na IBM do Brasil em projeto internacional administrando ambiente de produção de alta criticidade. Consultor independente de Bancos de Dados. Editor técnico da revista SQL Magazine. Mestrando em Ciência da Computação pelo Instituto de Computação da Universidade Estadual de Campinas – IC UNICAMP. Docente no curso de Administração de Banco de Dados na Dextraining.

Você gostou deste artigo?

Dê seu voto em www.devmedia.com.br/sqlmagazine/feedback

Ajude-nos a manter a qualidade da revista!



CURSOS ONLINE

A Revista Java Magazine oferece aos seus assinantes uma série de Cursos Online de alto padrão de qualidade.



CONHEÇA ALGUNS DOS CURSOS:

- **Curso de noSQL (Redis) com Java**
- **Curso Básico de JDBC**
- **Java Básico: Aplicações Desktop**
- **JSF com Primefaces**
- **Conhecendo o Apache Struts**

Para mais informações :

www.devmedia.com.br/curso/javamagazine
(21) 3382-5038

Avaliação de recursos do paralelismo

Uso de banco de dados para apoiar a avaliação

O conceito de paralelismo já entrou no ambiente dos processadores há muito tempo. Uma consideração importante aqui é a separação de dois mecanismos distintos: execução de instruções em paralelo (possibilitada pelos múltiplos núcleos), e processamento multitarefa. O último mecanismo pode ser encontrado em processadores de apenas um núcleo, onde o tempo de utilização do processador é dividido entre as diversas tarefas dando a impressão de que todas são executadas ao mesmo tempo. Já o mecanismo de instruções em paralelo só é possível em processadores com mais de um núcleo, onde cada núcleo vai realmente processar algo enquanto o outro processa outra instrução. É importante destacar que algumas instruções que dependem de resultados de instruções anteriores não podem ser executadas em paralelo.

Algumas melhorias nos processadores foram possíveis graças às novas tecnologias de materiais eletrônicos que permitiram o aumento de componentes em uma mesma área de *chip*. Outras melhorias foram possíveis através da lógica e organização de níveis de paralelismo de instruções, isto é, *pipelines* de dois ou mais estágios (ler **BOX 1**).

BOX 1. Pipeline

Internamente, o processador possui diversos estágios onde as instruções são interpretadas e executadas. Algumas instruções precisam passar por 2, 3, 4 ou mais estágios (podendo chegar a 14 dependendo do processador). O pipeline visa organizar as instruções de forma que mesmo que uma instrução não tenha sido realizada por completo, outra instrução inicie, desta forma otimizando o uso dos diversos estágios do processador. Como exemplo deste tipo de mecanismo podemos imaginar uma lavanderia, onde um conjunto de roupas deve passar por três estágios: lavar, secar e passar. Visando a otimização de uma lavanderia: devemos lavar um conjunto de roupas, em seguida podemos enviar este conjunto para o estágio de secagem, enquanto que um conjunto novo de roupas entra para lavagem, fazendo com que em determinado momento as três máquinas estejam em funcionamento.

O paralelismo oferecido pelas placas de vídeo difere em muito do paralelismo de instruções obtido por *pipeline*, pois o paralelismo visto em processadores gráficos é

Resumo DevMan

Porque este artigo é útil:

Um recurso importante a ser adotado em computação é o paralelismo. Em diversas áreas o paralelismo permite melhorias significativas.

O paralelismo de cada hardware, isoladamente, possibilita incremento de performance. Adotar um conjunto que tenha paralelismo em todos os níveis de hardware (disco, memória e processador) certamente é a opção ideal do ponto de vista de velocidade nas operações realizadas pelo banco de dados. Contudo, do ponto de vista financeiro (custos e investimentos), isso pode não ser tão simples assim.

Neste artigo serão descritos e realizados testes que utilizam recursos paralelos apenas de processador e uso de memória. O paralelismo em disco não será adotado, pois exige discos em níveis RAID 0 (zero) e configurações adicionais no sistema operacional. Para utilizar processamento paralelo, podemos adotar processadores gráficos oferecidos por placas de vídeo NVIDIA da série 8000 em diante ou ATI HD 7660 em diante. Neste artigo será apresentado um comparativo de processamento utilizando apenas o processador principal do computador e, em seguida, o processamento utilizando o processador principal em conjunto com o processador gráfico.

Neste contexto, este artigo é útil para quem deseja conhecer uma forma de processamento paralelo e aproveitá-la na resolução de tarefas que são realizadas no mesmo hardware em que está rodando seu banco de dados. Os processadores atuais disponibilizam vários núcleos (a grande maioria oferece quatro ou mais), com isso podemos explorar a execução de tarefas simultâneas em nossos programas. Além do processador principal, temos também os processadores das placas de vídeo. Estes geralmente com um número de núcleos (core) muito maior, algo entre 96 e 2688, dependendo do modelo.

A programação paralela apresentada neste artigo visa demonstrar o uso de processadores gráficos para realizar processamento de instruções independentes de forma simultânea. Desta forma, será possível direcionar o processamento de uma tarefa para o processador gráfico, enquanto o processador principal fica livre para outras tarefas.

possibilitado pelos múltiplos núcleos e não apenas pela lógica de organização das instruções e ocupação das unidades de processamento (*pipeline*).

Ambiente de testes

Para realização dos testes foi utilizado um computador convencional com a seguinte configuração: processador Intel Core 2 Quad Q6600 2.4 GHz, placa mãe Asus Maximus Formula, memória Corsair de 4GB FSB 800 MHz DDR2, disco rígido 500GB Sata II 16MB de buffer. A placa de vídeo utilizada foi uma GTX570 com o chip GF110 (2) da NVIDIA. Esta placa de vídeo é dotada de 480 núcleos CUDA (também denominados por processador de sombreamento - *Shader Processor*).

O termo CUDA vem da abreviação *Compute Unified Device Architecture*. O chip GF110 é um chip que foi lançado pela NVIDIA no final de 2010, suas características foram herdadas do seu antecessor GF100, que por sua vez recebeu as tecnologias do G80 e GT200. O chip GF100 trouxe ao mercado de processadores gráficos um conjunto composto por quatro grupos distintos de componentes, chamado *Graphics Processing Clusters* ou GPCS. Cada um destes quatro grupos possui quatro multiprocessadores denominados *Streaming Processors* (SMs). A **Figura 1** ilustra um SM com 32 CUDA core.

Para tornar as coisas simples, pense nisso assim: uma GF100 tem quatro GPCS, cada um dos quais está equipado com quatro SMs para um total de 16 SMs. Dentro de cada um destes SMs tem-se 32 núcleos CUDA com um total de 512 núcleos CUDA.

Como modificação aos seus antecessores, o chip GF110 possui duas configurações: as placas GTX570 possuem uma das 16 unidades SMs desativada, visando menor consumo de energia, desta forma apresenta 480 CUDA Core, enquanto que as GTX580 apresentam todos os 16 SMs ativados.

O sistema operacional utilizado foi Microsoft Windows Seven 64 Bits. O banco de dados foi o MySQL versão 5.1.44. Para desenvolvimento do algoritmo de testes foi utilizado o Microsoft Visual Studio 2010 C++. Para utilização do processador gráfico é necessário um conjunto de programas:

- CUDA Driver, para acessar o processador gráfico e a memória do processador gráfico;
- CUDA Toolkit é um conjunto de bibliotecas que permite a integração do C++ com o processador gráfico;
- DK com exemplos de código.

Para realização dos testes foram utilizadas três tabelas. As duas primeiras são idênticas e utilizam a *storage engine InnoDB* (ler **BOX 2**), a outra utiliza a *storage engine MyISAM*.

Como dito anteriormente, temos duas tabelas idênticas: uma irá armazenar os dados quando for testado apenas o processador, enquanto que a outra vai armazenar os dados quando o processador gráfico entrar em ação. A **Listagem 1** exibe o comando para criar esta tabela.

Dentre as duas tabelas que foram criadas, uma delas tem o nome “tabelaCPU” e a outra tem o nome “tabelaGPU”.

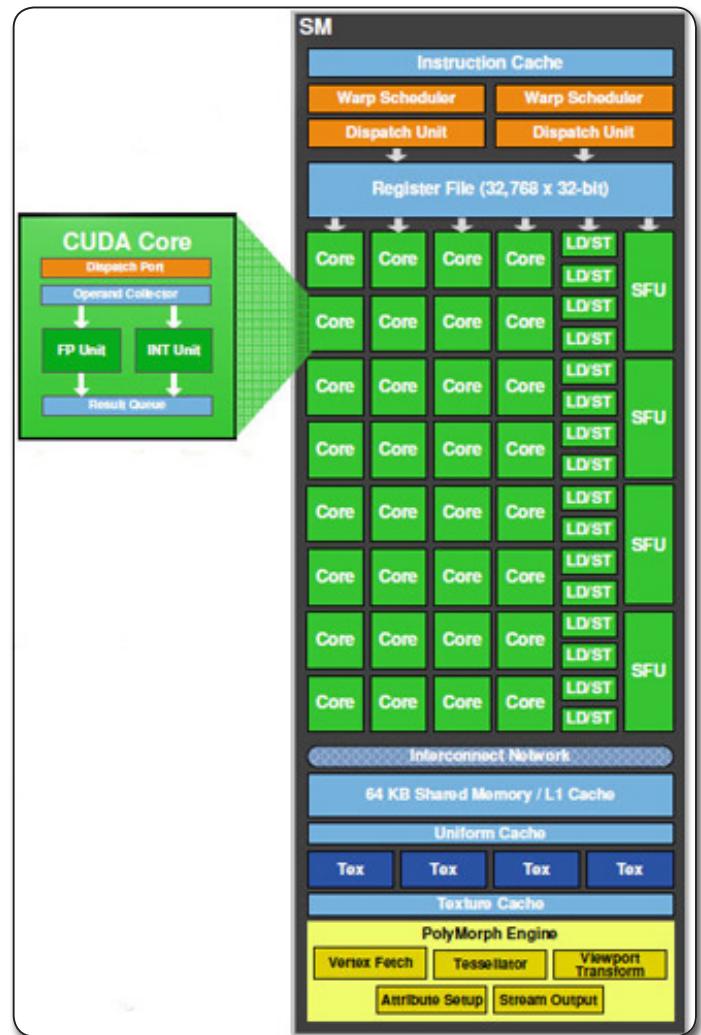


Figura 1. Ilustração de um Streaming Processor (SM) com 32 CUDA core

BOX 2. InnoDB

O motor de armazenamento InnoDB provê ao MySQL um mecanismo seguro com suporte a transações. Permite também a recuperação em caso de falhas. O InnoDB faz bloqueio em nível de registro e também fornece uma leitura sem bloqueio em SELECT, permitindo que vários usuários possam ler o mesmo registro. Estes recursos aumentam a performance e a concorrência de multiusuários. Não há necessidade de escalonamento de bloqueios no InnoDB, pois o bloqueio a nível de registro no InnoDB exige pouca memória. Este motor de armazenamento está disponível a partir da versão 4.0 do MySQL. Por se tratar de um motor com recursos mais avançados, requer mais espaço em memória e disco, além de se apresentar, em determinadas situações, significativamente mais lento.

Nos testes serão feitas inserções de registros nestas tabelas. Ao final de cada teste, será gravado o resultado do tempo necessário em uma terceira tabela. A **Listagem 2** apresenta o código de criação desta última.

A tabela criada através do comando SQL apresentado na **Listagem 2** será a responsável por armazenar o tempo que cada teste leva para ser realizado. Como serão realizados vários testes

Avaliação de recursos do paralelismo

(cada teste terá números de inserções diferentes), temos a coluna “quant” que irá armazenar o número de inserções de cada teste. A proposta é realizar testes com cargas distintas para verificar se em determinado momento o processamento na GPU demanda mais tempo.

Listagem 1. SQL para criação das tabelas que vão receber os dados

```
CREATE TABLE IF NOT EXISTS `tabelaCPU` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `nome` varchar(50) NOT NULL,
  `endereco` varchar(50) DEFAULT NULL,
  `valor` float DEFAULT NULL,
  `ativo` int(11) NOT NULL DEFAULT '1',
  `cidade` int(11) DEFAULT NULL,
  `dataCadastro` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 AUTO_INCREMENT=1;
```

Listagem 2. SQL para criação da tabela que irá receber os tempos de processamento

```
CREATE TABLE IF NOT EXISTS `estatisticas` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `quant` int(11) NOT NULL,
  `cpu` double(30,16) unsigned NOT NULL,
  `gpu` double(30,16) unsigned NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 AUTO_INCREMENT=1;
```

Para realização dos testes foi desenvolvido um algoritmo em C++ composto de cinco métodos, mais um método principal. O primeiro método realiza a soma (no processador principal) de dois arrays ou vetores, e a inserção de um registro na tabela “tabelaCPU”. Tal processo pode ser feito um número de vezes pré-estabelecido, recebido por parâmetro. No segundo método é feito o mesmo que no primeiro com a diferença de que entra em ação o processador gráfico (para somar os dois vetores) e a tabela que recebe o(s) registro(s) é a “tabelaGPU”. O terceiro método é responsável por limpar as três tabelas utilizadas pelos testes (este é utilizado apenas uma vez no início do teste). O quarto método grava na tabela “estatisticas” os tempos necessários para realização dos dois primeiros métodos. O último método foi desenvolvido exclusivamente para realizar o processamento no processador gráfico, utilizando o SDK fornecido pela NVIDIA. Por fim, o método principal faz as chamadas necessárias.

Comparativo realizado

O processamento paralelo visa à realização de várias operações simples simultaneamente. Tais operações são originadas geralmente de uma operação de maior complexidade que pode ser dividida em operações mais simples, e consequentemente ser realizada em menos tempo. Nem todas as operações são passíveis deste tipo de segmentação. Neste artigo, a tarefa complexa que se deseja realizar é a soma de dois vetores de 40.000 elementos. Nota-se que temos 40.000 operações matemáticas para serem realizadas, sendo que uma não interfere

no processamento da outra, permitindo desta forma a sua realização em paralelo.

Os resultados obtidos nos testes realizados pelo leitor podem variar dependendo da diferença entre o hardware apresentado anteriormente e o hardware do leitor.

O comparativo consiste na realização de uma soma de dois vetores, cada um deles com 40.000 elementos do tipo ponto flutuante (*float*). O resultado da soma será armazenado em um terceiro vetor que por sua vez terá apenas um de seus elementos inserido no banco de dados, na coluna “valor”. Como este experimento é para fins de testes de processamento, não será gravado todo o vetor resultante no banco de dados, pois o processo tende a levar muito tempo e demanda de disco, não de processamento e memória. A operação de soma dos 40.000 elementos será realizada pelo processador principal (CPU) ou pelo processador gráfico (GPU).

Vamos analisar as partes do algoritmo. Primeiro o método principal na **Listagem 3**.

Listagem 3. Método principal do algoritmo de testes

```
01. int main(int argc, char** argv){
02.     int sp=0;
03.     const int tam = 40000;
04.     float aCPU[tam], aGPU[tam], bCPU[tam], bGPU[tam], cCPU[tam], cGPU[tam];
05.     float valor;
06.     float tCPU=0;
07.     float tGPU=0;
08.     limpa();
09.     for (int pos=0;pos<tam;pos++){
10.         valor=pos*3;
11.         aCPU[pos]=aGPU[pos]=valor;
12.         valor=rand()/(float)RAND_MAX;
13.         bCPU[pos]=bGPU[pos]=valor;
14.         cCPU[pos]=cGPU[pos]=1.1;
15.     }
16.     while (sp<=2000){
17.         sp=sp+100;
18.         tCPU = testCPU(aCPU,bCPU,cCPU,sp,tam);
19.         tGPU = testGPU(aGPU,bGPU,cGPU,sp,tam);
20.         grava(sp,tCPU,tGPU);
21.         for(int i=0;i<tam;i++){
22.             if (cCPU[i]!=cGPU[i]) {
23.                 cout << "Erro" << endl;
24.             return;
25.         }
26.     }
27. }
28. system("PAUSE");
29. }
```

Nas linhas 2 e 3 são declaradas as variáveis para os seguintes contadores: quantidade de registros a serem inseridos (sp) e tamanho dos vetores (tam). Na linha 4 são criados os vetores. Na linha 5 é criada a variável “valor”. Esta irá receber o valor que será enviado a cada posição do vetor. Nas linhas 6 e 7 são criadas as variáveis para receber o tempo gasto para realizar cada processamento. Na linha 8 é chamado o método que limpa as três tabelas. A **Listagem 4** apresenta os comandos deste método.

Listagem 4. Método que realiza a limpeza das 3 tabelas

```

01. void limpa(){
02. //Os comandos dentro do bloco try são testados, caso algum erro ocorra,
03. //estes são exibidos na tela, conforme linhas 26 até 31
04. try{
05. //Criação dos ponteiros para conexão e sentença a ser enviada ao banco
06. sql::Driver *driver;
07. sql::Connection *con;
08. sql::PreparedStatement *pstmt;
09. //Cria a conexão
10. driver = get_driver_instance();
11. //Parâmetros de conexão desejados
12. con = driver->connect("tcp://localhost:3306","root","123");
13. //Conectando com o banco "bench001"
14. con->setSchema("bench001");
15. //Comando SQL para limpar a tabela "tabelaCPU"
16. pstmt = con->prepareStatement("TRUNCATE TABLE tabelaCPU;");
17. //Envio do comando para ser executado
18. pstmt->executeUpdate();
19. pstmt = con->prepareStatement("TRUNCATE TABLE tabelaGPU;");
20. pstmt->executeUpdate();
21. pstmt = con->prepareStatement("TRUNCATE TABLE estatisticas;");
22. pstmt->executeUpdate();
23. //Apagando os conteudos dos ponteiros pstmt e con
24. delete pstmt;
25. delete con;
26. } catch (sql::SQLException &e) {
27. cout << "# ERR: SQLException in " << __FILE__;
28. cout << "(" << __FUNCTION__ << ") on line " << __LINE__ << endl;
29. cout << "# ERR: " << e.what();
30. cout << "(MySQL error code: " << e.getErrorCode();
31. cout << ",SQLState: " << e.getSQLState() << ")" << endl;
32. }
33. }

```

Depois de limpar as tabelas, o método principal entra em um laço de repetição que inicia na linha 9 e vai até a 15, para inicializar os 3 vetores, sendo que o vetor resultante (cCPU e cGPU) recebe o valor 1.1 em todas as posições. Apenas para fins didáticos, este valor foi atribuído para verificar que no final este valor é realmente modificado. No vetor aCPU e aGPU foi colocado o valor 0 (zero) no primeiro elemento e o valor 39999 multiplicado por 3 no último elemento. No vetor bCPU e bGPU foi colocado um valor aleatório. Na linha 16 o teste é inicializado. Nesta linha podemos observar que serão feitos testes enquanto a variável “sp” tenha valor menor ou igual a 1000. Desta forma, serão feitos 20 testes, pois a variável recebe os valores múltiplos de 100 entre 0 e 2000. Conforme pode ser visto na linha 17, são 20 valores distintos. Tais valores são adotados para a quantidade de inserções.

Na linha 18 é chamado o método responsável por realizar a tarefa exclusivamente no processador principal (testCPU), o tempo necessário para realização do teste é armazenado na variável “tCPU”. Na linha 19 é chamado o método responsável por realizar a tarefa no processador gráfico (testGPU), da mesma forma é armazenado o tempo para realização da tarefa. A linha 20 (grava) armazena os dois tempos bem como a quantidade de inserções feitas. Por fim, na linha 21 é iniciado um laço de repetição apenas para conferir se os dois processamentos geraram o mesmo resultado.

Vejamos o método “grava” e em seguida o método “testCPU” e “testGPU”. A **Listagem 5** apresenta o método “grava”.

Listagem 5. Método que realiza a inserção na tabela de estatísticas

```

01. void grava(int q,double c,double g){
02. //Os comandos dentro do bloco try são testados, caso algum erro ocorra,
03. //estes são exibidos na tela, conforme linhas 26 até 31
04. try{
05. //Criação dos ponteiros para conexão e sentença a ser enviada ao banco
06. sql::Driver *driver;
07. sql::Connection *con;
08. sql::PreparedStatement *pstmt;
09. //Cria a conexão
10. driver = get_driver_instance();
11. //Parâmetros de conexão desejados
12. con = driver->connect("tcp://localhost:3306","root","123");
13. //Conectando com o banco "bench001"
14. con->setSchema("bench001");
15. //Comando SQL para inserir o registro na tabela "estatisticas"
16. pstmt = con->prepareStatement("INSERT INTO estatisticas(quant,cpu,gpu)
VALUES (?, ?, ?)");
17. //as próximas 3 linhas atribuem valores para o registro a ser inserido
18. pstmt->setInt(1,q);
19. pstmt->setDouble(2,c);
20. pstmt->setDouble(3,g);
21. //Envio do comando para ser executado
22. pstmt->executeUpdate();
23. //Apagando os conteudos dos ponteiros pstmt e con
24. delete pstmt;
25. delete con;
26. }catch (sql::SQLException &e) {
27. cout << "# ERR: SQLException in " << __FILE__;
28. cout << "(" << __FUNCTION__ << ") on line " << __LINE__ << endl;
29. cout << "# ERR: " << e.what();
30. cout << "(MySQL error code: " << e.getErrorCode();
31. cout << ",SQLState: " << e.getSQLState() << ")" << endl;
32. }
33. }

```

O método grava recebe três parâmetros: a quantidade de inserções, o tempo necessário para realizar a tarefa somente no processador principal e o tempo necessário para realizar a tarefa utilizando o processador gráfico em conjunto com o processador principal. Dentro do método é feita a conexão com o MySQL e a inserção do registro. Vamos agora ao método “testCPU”, que pode ser visto na **Listagem 6**.

O método “testCPU” recebe cinco parâmetros: o vetor “a” e “b”, já inicializados, o vetor “c”, a quantidade de vezes que o teste deve ser feito “sp” e tamanho dos vetores “tam”. Dentro do método podemos identificar um laço, linha 6 até 36, que repete os comandos responsáveis pela soma dos dois vetores e inclusão do registro na tabela “tabelaCPU”. Temos também a variável “t0” e “tf”, responsável pelo armazenamento dos tempos de início e fim que são calculados na variável “tempo_gasto”, retornada pelo método. O registro que é inserido na tabela armazena apenas o valor da posição 19999 do vetor “c”, pois o armazenamento de todo o vetor aumentaria a exigência de disco.

Observe agora a **Listagem 7**.

O método “testGPU” funciona com o mesmo princípio do método anterior, com a diferença que precisa realizar três passos adicionais: 1) enviar dados para placa de vídeo; 2) chamar o método que realiza o processamento no processador gráfico; 3) ler os resultados na memória da placa de vídeo e trazer para memória principal. Note que tais passos representam trabalho para o pro-

Avaliação de recursos do paralelismo

cessador principal, além de exigirem operações adicionais. Desta forma, caso o processamento enviado para o processador gráfico seja um processamento simples (e neste caso é um processamento simples), não teremos muitas vantagens.

A principal diferença é que no método “testCPU” temos um laço de repetição (linhas 7, 8 e 9) que não é necessário no método “testGPU”. Tal laço tem o efeito equivalente ao apresentado na linha 23 do método “testGPU”. Tal linha chama o método “vectorAdd”, disponibilizado pelo SDK da NVIDIA. Este método executa em paralelo a soma dos vetores.

A soma paralela dos vetores é processada da forma que o usuário desejar, utilizando mais ou menos linhas de execução (*thread*). Note que o método “vectorAdd” tem dois parâmetros entre os sinais “<<<” e “>>>”. Tais parâmetros indicam quantas *threads* serão processadas. Vamos definir 256 threads por bloco (*threadsPerBlock*), este valor é um valor padrão apresentado nos exemplos disponibilizados pela NVIDIA, podendo ser modificado. Quanto à quantidade de blocos, vamos definir o valor máximo possível, sendo que a multiplicação deste valor por 256 deve chegar a 40000 ou passar um pouco, caso seja necessário. O valor máximo

que podemos multiplicar o 256 para chegar a 40000 é dado pela fórmula da linha 17 (no método “testGPU”):

```
int blocksPerGrid =(tam + threadsPerBlock - 1) / threadsPerBlock;  
        (40000 + 256 - 1) / 256;  
        (40256 - 1) / 256;  
        40255 / 256;  
        157
```

Listagem 7. Método que realiza o teste no processador gráfico

```
01. double testGPU(float aGPU[], float bGPU[], float cGPU[], int sp, int tam){  
02.     int icont;  
03.     double tempo_gasto;  
04.     clock_t t0, tf;  
05.     t0 = clock();  
06.     for(int h=0;h<sp;h++){  
07.         for(icont=0;icont<tam;icont++){  
08.             cGPU[icont]=aGPU[icont]+bGPU[icont];  
09.         }  
10.         try {  
11.             sql::Driver *driver;  
12.             sql::Connection *con;  
13.             sql::PreparedStatement *pstmt;  
14.             string a="nome";  
15.             std::stringstream Resultado;  
16.             driver = get_driver_instance();  
17.             con = driver->connect("tcp://localhost:3306","root","123");  
18.             con->setSchema("bench001");  
19.             pstmt = con->prepareStatement("INSERT INTO tabelaCPU  
                (nome,endereco,versao) VALUES (?,?,?)");  
20.             Resultado << h;  
21.             pstmt->setString(1, a+" "+Resultado.str());  
22.             pstmt->setString(2, "string qualquer para teste.");  
23.             pstmt->setDouble(3, cCPU[19999]);  
24.             pstmt->executeUpdate();  
25.             Resultado.str("");  
26.             delete pstmt;  
27.             delete con;  
28.         } catch (sql::SQLException &e) {  
29.             cout << "# ERR: SQLException in" << __FILE__;  
30.             cout << "(" << __FUNCTION__ << ")" on line" << __LINE__ << endl;  
31.             cout << "# ERR:" << e.what();  
32.             cout << "(MySQL error code:" << e.getErrorCode();  
33.             cout << ",SQLState:" << e.getSQLState() << ")" << endl;  
34.         }  
35.     }  
36.     tf = clock();  
37.     tempo_gasto = ((double)(tf - t0)) / CLOCKS_PER_SEC;  
38.     return tempo_gasto;  
39. }
```

Listagem 6. Método que realiza o teste exclusivamente no processador principal

```
01. double testCPU(float aCPU[], float bCPU[], float cCPU[], int sp, int tam){  
02.     int icont;  
03.     double tempo_gasto;  
04.     clock_t t0, tf;  
05.     t0 = clock();  
06.     for(int h=0;h<sp;h++){  
07.         for(icont=0;icont<tam;icont++){  
08.             cCPU[icont]=aCPU[icont]+bCPU[icont];  
09.         }  
10.         try {  
11.             sql::Driver *driver;  
12.             sql::Connection *con;  
13.             sql::PreparedStatement *pstmt;  
14.             string a="nome";  
15.             std::stringstream Resultado;  
16.             driver = get_driver_instance();  
17.             con = driver->connect("tcp://localhost:3306","root","123");  
18.             con->setSchema("bench001");  
19.             pstmt = con->prepareStatement("INSERT INTO tabelaCPU  
                (nome,endereco,versao) VALUES (?,?,?)");  
20.             Resultado << h;  
21.             pstmt->setString(1, a+" "+Resultado.str());  
22.             pstmt->setString(2, "string qualquer para teste.");  
23.             pstmt->setDouble(3, cCPU[19999]);  
24.             pstmt->executeUpdate();  
25.             Resultado.str("");  
26.             delete pstmt;  
27.             delete con;  
28.         } catch (sql::SQLException &e) {  
29.             cout << "# ERR: SQLException in" << __FILE__;  
30.             cout << "(" << __FUNCTION__ << ")" on line" << __LINE__ << endl;  
31.             cout << "# ERR:" << e.what();  
32.             cout << "(MySQL error code:" << e.getErrorCode();  
33.             cout << ",SQLState:" << e.getSQLState() << ")" << endl;  
34.         }  
35.     }  
36.     tf = clock();  
37.     tempo_gasto = ((double)(tf - t0)) / CLOCKS_PER_SEC;  
38.     return tempo_gasto;  
39. }
```

Por questões de arredondamento e tipos de variáveis (inteiras), esta fórmula deve ser utilizada caso contrário os valores finais dos vetores não são compatíveis, isto é, a fórmula $40000/256$ resulta em 156. Mas a multiplicação não resulta em 40000, e sim em 39936, por isso temos o incremento de 256 subtraído de 1 unidade. Note que a utilização da fórmula prevê a utilização de valores quaisquer. Na **Listagem 8** temos o método “vectorAdd”, disponibilizado no SDK da NVIDIA.

Listagem 8. Método para soma de vetores em CUDA

```
1. __global__ void vectorAdd(const float *A, const float *B, float *C,
   int numElements){
2.     int i = blockDim.x * blockIdx.x + threadIdx.x;
3.     while (i < numElements){
4.         C[i]=A[i] + B[i];
5.         i+= blockDim.x * gridDim.x;
6.     }
7. }
```

O método “vectorAdd” inicia com a palavra reservada “`__global__`”. Isso indica ao compilador C++ que tal método será chamado pelo processador principal e deve ser executado pelo processador gráfico. Além desta palavra reservada, temos outras que podem ser utilizadas caso um método seja chamado pelo processador gráfico e executado pelo próprio processador gráfico (`_device_`), ou caso um método seja chamado pelo processador principal e executado pelo próprio principal (`_host_`). O método ainda recebe quatro parâmetros, são eles: o ponteiro para o primeiro e segundo vetor, sendo que estes dois já estão com os devidos valores, pois foram enviados para a memória de vídeo pelo método “`testGPU`”, nas linhas 13 e 14; o ponteiro para a posição onde deve ser armazenado o vetor resultante e o número de elementos dos vetores.

Internamente, o método “vectorAdd” tem uma instrução que faz com que a variável “`i`” assuma valores de 0 até 40255 em paralelo, linha 2. Na linha 4, temos a atribuição do valor da soma dos vetores “`a`” e “`b`” sendo armazenado no vetor “`c`” somente para valores de “`i`” menores que o valor do parâmetro “`numElements`”, ou seja 39999, começando em 0 (zero).

Vamos analisar agora os resultados do primeiro teste. Observe a **Tabela 1**.

Como resultado, temos o gráfico apresentado na **Figura 2**.

Percebemos que a realização do processamento no CPU em conjunto com o GPU não traz benefícios do ponto de vista redução de tempo, inclusive alguns testes demonstraram que o processamento demandou mais tempo. É válido lembrar que o processamento consiste de diversas etapas e algumas delas foram alocadas no processador gráfico. O motivo deste resultado se deve à necessidade de enviar os dados da memória principal para a memória do processador gráfico. Como o processamento a ser realizado não se trata de um processamento complexo, não vale a pena deslocar o processamento para o processador gráfico e carregar os resultados para a memória principal.

Quantidade de inserções	Tempo para executar no CPU	Tempo para executar no CPU juntamente com GPU
100	1,406	2,100
200	2,818	4,381
300	4,032	5,307
400	5,384	5,606
500	7,046	6,808
600	8,472	8,527
700	10,680	11,850
800	12,873	14,415
900	14,935	21,165
1000	39,717	50,537
1100	41,921	42,322
1200	54,513	70,863
1300	49,598	50,114
1400	54,275	57,174
1500	64,111	61,742
1600	57,750	59,078
1700	76,754	76,408
1800	65,679	66,468
1900	71,158	69,170
2000	73,055	73,914

Tabela 1. Tempos para processamento no CPU e no CPU + GPU

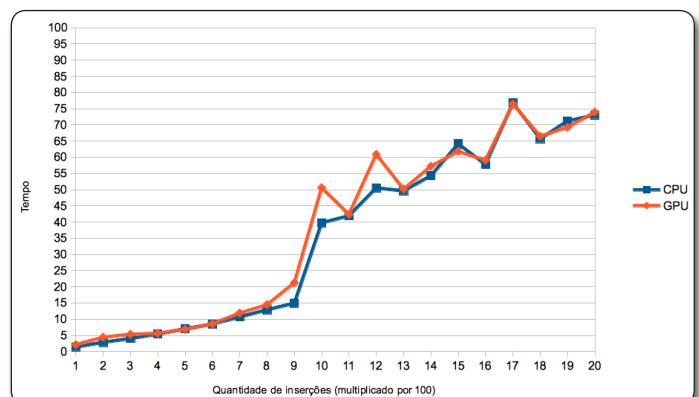


Figura 2. Tempos para processamento no CPU e no CPU + GPU

Neste caso, desejamos o resultado gerado no processador gráfico para enviar ao banco de dados.

Diante deste resultado percebemos a necessidade de realizar testes com processamento mais complexo. As **Listagens 9 e 10** apresentam esta alternativa.

Dadas estas duas modificações, temos um processamento mais complexo a ser realizado. Ambas as modificações consistem em realizar diferentes operações matemáticas. Depois destas modificações foram feitos novos testes, resultando nos tempos apresentados na **Tabela 2**.

Avaliação de recursos do paralelismo

Como resultado, temos o gráfico apresentado na **Figura 3**.

Neste segundo gráfico percebemos que quanto maior a quantidade de registros processados, mais a linha do CPU se distancia da linha GPU, indicando que o tempo para processamento exclusivamente no CPU é cada vez maior do que o processamento em conjunto com o GPU.

Quantidade de inserções	Tempo para executar no CPU	Tempo para executar no CPU juntamente com GPU
100	3,713	2,262
200	6,396	4,368
300	8,736	6,349
400	12,214	7,020
500	15,850	8,705
600	25,890	22,511
700	34,445	26,333
800	38,220	30,233
900	44,959	33,743
1000	50,747	36,723
1100	55,012	43,541
1200	60,668	44,257
1300	63,056	48,500
1400	71,223	53,656
1500	72,602	54,709
1600	79,420	59,077
1700	82,462	64,257
1800	85,347	66,846
1900	94,896	69,311
2000	97,484	71,792

Tabela 2. Tempos para processamento (agora mais complexo) no CPU e no CPU + GPU

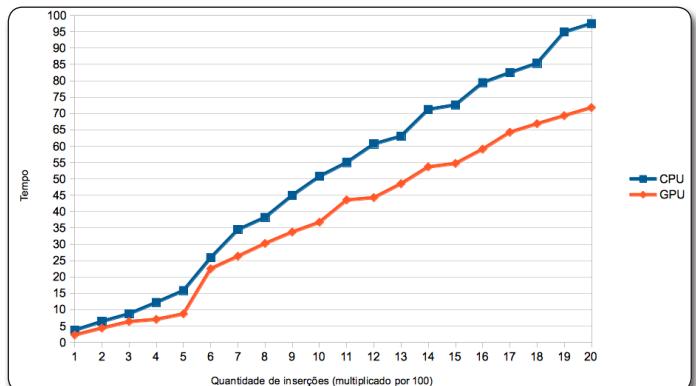


Figura 3. Tempos para processamento (agora mais complexo) no CPU e no CPU + GPU

Conclusão

Vimos neste artigo uma possibilidade de divisão de carga de processamento entre processador principal e processador gráfico. O objetivo em dividir uma carga de processamento entre estes dois processadores se concentra em: liberar o processador principal para realização de outras tarefas, redução de consumo e aproveitamento de um *hardware* de custo mais acessível.

Diante dos testes apresentados, podemos concluir que somente vale a pena deslocar o processamento para o processador gráfico quando este processamento realmente for um processamento complexo para o processador principal, pois o envio e a leitura dos dados surgem como um passo que pode demandar tempo e causar resultados indesejados. Para futuros testes podemos levar em consideração o custo de processadores principais e gráficos e estabelecer um comparativo entre os custos e benefícios que os mesmos podem oferecer.

Autor



Marcelo Josué Telles

marcelojtelles@gmail.com

Graduado em Licenciatura da Computação pela Universidade Feevale(2007), especialista em informática na Educação pela UFRGS(2011). Cursou disciplinas do Programa de Pós Graduação em Computação na UFRGS (PPGC e PGMICRO), como aluno especial. Trabalha com desenvolvimento de software nas linguagens PHP, Java e Visual Basic.



Links:

Computer Architecture. A Quantitative Approach. John L. Hennessy, David A. Patterson. Fifth Edition. USA: Elsevier, 2012.

<http://www.guru3d.com/articles-pages/geforce-gtx-570-review,1.html>

CUDA Toolkit

<https://developer.nvidia.com/cuda-toolkit>

Você gostou deste artigo?

Dê seu voto em www.devmedia.com.br/sqlmagazine/feedback

Ajude-nos a manter a qualidade da revista!



Somos tão apaixonados por tecnologia que o nome da empresa diz tudo.

Porta 80 é o melhor que a Internet pode oferecer para sua empresa.

Já completamos 8 anos e estamos a caminho dos 80, junto com nossos clientes.

Adoramos tecnologia.
Somos uma equipe composta de gente que entende e gosta do que faz,
assim como você.



Estrutura

100% NACIONAL.
Servidores de primeira linha, links de alta capacidade.



Suporte diferenciado

Treinamos nossa equipe para fazer mais e melhor. Muito além do esperado.



Serviços

Oferecemos a tecnologia mais moderna, serviços diferenciados e antenados com as suas necessidades.



1-to-1

Conhecemos nossos clientes. Atendemos cada necessidade de forma única. Conheça!



Porta 80

WEB HOSTING

Hospedagem | Cloud Computing | Dedicados | VoIP | Ecommerce |
Aplicações | Streaming | Email corporativo

porta80.com.br | comercial@porta80.com.br | twitter.com/porta80

SP 4063-8616 | RJ 4063-5092 | MG 4063-8120 | DF 4063-7486

IMAGINE TER 0.002 SEGUNDOS PARA ACESSAR UM DADO CRUCIAL.

Por isso, as empresas
preferem profissionais
Oracle formados no Infnet.

Faça os cursos oficiais

ORACLE WORKFORCE
DEVELOPMENT PROGRAM



PÓS-GRADUAÇÃO

MIT em GESTÃO DE BANCOS DE DADOS COM ORACLE

Desenvolva uma visão abrangente sobre Bancos de Dados alinhada a competências técnicas para administrar uma estrutura mista de bancos de dados e manter um ambiente ideal para o Oracle.

www.infnet.edu.br/posemoracle



FORMAÇÃO

DBA ORACLE

Domine a arquitetura de um banco de dados Oracle. Crie, gerencie, configure e recupere este banco utilizando ferramentas de monitoramento de desempenho e otimização de dados.

www.infnet.edu.br/oracle11g

ESCOLA SUPERIOR DA TECNOLOGIA DA INFORMAÇÃO
www.infnet.edu.br | 21 2122-8800

infnet
INSTITUTO