UniATENEU
Centro Universitário

Banco de Dados de Alta Performance

1ª Edição

UniATENEU
Centro Universitário

BANCO DE DADOS

Autoria:
Geraldo Barbosa do Amarante

ATENEU Online
EAD
CONECTANDO VALORES

UniATENEU
Centro Universitário

Expediente

Reitor:
Prof. Cláudio Ferreira Bastos

Pró-reitor administrativo financeiro:
Prof. Rafael Rabelo Bastos

Pró-reitor de relações institucionais:
Prof. Cláudio Rabelo Bastos

Pró-reitor acadêmico:
Prof. Herbert Gomes Martins

Direção EAD:
Prof. Ricardo Zambrano Júnior

Coordenação EAD:
Profa. Luciana Rodrigues Ramos

Ficha Técnica

Autoria:
Geraldo Barbosa do Amarante

Supervisão de produção EAD:
Francisco Cleuson do Nascimento Alves

Design instrucional:
Antonio Carlos Vieira

Projeto gráfico e capa:
Francisco Erbínio Alves Rodrigues

Diagramação e tratamento de imagens:
Ismael Ramos Martins

Revisão textual:
Antonio Carlos Vieira

Ficha Catalográfica
Catalogação na Publicação
Biblioteca Centro Universitário Ateneu

AMARANTE, Geraldo Barbosa do. Banco de dados. Geraldo Barbosa do Amarante. – Fortaleza: Centro Universitário Ateneu, 2020.

172 p.

ISBN: 978-65-88268-36-0

1. Sistemas Gerenciadores de Banco de Dados (SGBD). 2. Linguagem SQL. 3. Técnicas avançadas de dados. 4. Bancos de dados em alta disponibilidade e alta performance. Centro Universitário Ateneu. II. Título.

SEJA BEM-VINDO!

Caro estudante, este é o material da disciplina *Banco de dados*, que pretende ser o guia para o estudo e entendimento da tecnologia de banco de dados, tecnologia esta que é a base para todo e qualquer sistema computadorizado. De nada adiantaria sistemas sofisticados de software se não tivéssemos onde guardar as informações produzidas por eles, de uma forma segura e disponível.

Podemos dizer que o banco de dados é a "alma da empresa", não importando o seu porte no mundo dos negócios, seja uma empresa gigantesca, como a Amazon ou a Google, ou uma microempresa familiar, a questão da tomada de decisões baseada em dados independe do seu tamanho, é vital para a sua sobrevivência e crescimento no mercado.

A revista *The Economist*, em 2017, trouxe uma frase que transmite bem o quanto é importante gerir os dados da empresa, a frase dizia que *"o recurso mais valioso do mundo não é o petróleo, mas sim os dados"*. Isto significa que é crucial as empresas manterem as informações de qualidade sobre os seus clientes, fornecedores, suas transações contábeis, processos etc. de forma segura e disponível.

Este livro está dividido em quatro unidades. A primeira unidade tem o objetivo de dar uma visão geral dos sistemas gerenciadores de banco de dados (SGBD), fornecendo uma base conceitual. Também discutiremos o modelo relacional que é mais amplamente utilizado na maioria dos sistemas gerenciadores de banco de dados comerciais. Veremos também tópicos relacionados a um projeto de um pequeno banco de dados a ser implementado nas unidades subsequentes. Na segunda unidade, aprenderemos a instalar e configurar o SGBD MySQL, assim como a ferramenta MySQL Workbench para que seja possível desenvolver uma aplicação prática dos conceitos aprendidos utilizando a linguagem SQL. Na terceira unidade, aprofundaremos mais nas características e recursos do SGBD MySQL, para aprendermos a tornar os nossos bancos de dados mais eficientes e com alto desempenho. Na quarta e última unidade, discutiremos assuntos relacionados à distribuição de banco de dados, seu monitoramento e como melhorar o desempenho desta estrutura.

Bons estudos!

Estes ícones aparecerão em sua trilha de aprendizagem e significam:

ANOTAÇÕES
Espaço para anotar suas ideias.

CONECTE-SE
Convocar o estudante para interagir no fórum tira-dúvidas.

CURIOSIDADE
Informação curiosa relacionada ao conteúdo.

EXERCÍCIO RESOLVIDO
Atividade explicativa para guiar o estudante.

FIQUE ATENTO
Informação complementar ao texto principal.

LINK WEB
Indicação de sites.

MATERIAL COMPLEMENTAR
Texto ou mídias complementares ao assunto da aula.

MEMORIZE
Tópico ou fato importante de lembrar.

OBJETIVOS DE APRENDIZAGEM
Objetivos de estudo do capítulo ou unidade.

PRATIQUE
Exercícios para fixar os conteúdos.

REFERÊNCIAS
Fontes de pesquisa citadas no texto.

RELEMBRE
Resumo do conteúdo estudado.

SUMÁRIO

03

TÉCNICAS AVANÇADAS DE DADOS

04

BANCOS DE DADOS EM ALTA DISPONIBILIDADE E ALTA PERFORMANCE

Unidade 04 Bancos de dados em alta disponibilidade e alta performance

Apresentação

Nesta unidade, discutiremos a necessidade da evolução dos sistemas gerenciadores de banco de dados para permitir sua alta disponibilidade, pois não adianta uma alta disponibilidade se não for associada à alta performance. Poderíamos ter um banco de dados altamente disponível, mas que a resposta fosse tão lenta que se tornaria impraticável. Então, os dois termos estão intimamente relacionados.

Também estudaremos a teoria da replicação, cuja aplicação é basicamente a de copiar bancos de dados oriundos de um ou mais servidores, chamados servidores mestre, para outros servidores chamados de réplicas, que estarão geograficamente distantes, levando os dados mais próximos de seus usuários. Como exemplo, temos um sistema bancário com um ou mais servidores centralizados com todas as informações de transações bancárias e suas réplicas nas agências para permitir que clientes acessem em alta performance.

Estudaremos ainda a topologia de computadores organizados em cluster, cuja finalidade é manter os sistemas o maior tempo em operação. Nesta estrutura, as requisições aos sistemas são distribuídas aos nós, e havendo algum problema de falha em um deles, este é bloqueado, não recebendo mais as requisições e, desta forma, a estrutura continua a funcionar. Alguns aspectos destas duas estruturas de alta disponibilidade e alta performance serão discutidas em relação ao SGBD MySQL.

O monitoramento é também discutido, pois é de fundamental importância para manter a alta disponibilidade em funcionamento. Descreveremos uma organização baseada em ITIL para o atendimento de incidentes detectados pelo sistema de monitoramento.

Por fim, veremos algumas dicas simples sobre como evitar algumas práticas que podem ser ofensoras à alta performance de um banco de dados, impactando na velocidade de execução das consultas.

Objetivos de aprendizagem

- *Discutir a necessidade da evolução dos SGBDs para disponibilizarem ferramentas de replicação em função da evolução das redes de computadores, como a internet;*
- *Estudar os conceitos sobre replicação e estrutura em cluster e entender como elas permitem a alta disponibilidade e a alta performance;*
- *Entender a importância do monitoramento de instalações de computadores em alta disponibilidade, e como deve ser organizada uma estrutura de gestão para o tratamento dos incidentes e a própria manutenção das topologias desta natureza.*

1. A alta disponibilidade e a alta performance

O termo **alta disponibilidade**, muitas vezes, é utilizado como o sinônimo de **confiabilidade**, ou seja, tornar uma determinada solução computacional acessível o maior tempo possível e ao mesmo ser altamente tolerante a falhas, reduzindo ao máximo a parada dos respectivos sistemas, sejam estas paradas planejadas ou não.

A confiabilidade nos **sistemas computacionais** ocorre quando estes não somente se mantenham em operação o maior tempo possível, mas também atendam às **requisições** que são feitas pelos usuários, em um menor tempo possível. Podemos dizer que eles devem disponibilizar, conjuntamente, uma alta disponibilidade e uma alta performance.

Esta alta disponibilidade e alta performance podem ser alcançadas de diversas maneiras, resultando em diferentes níveis de disponibilidade e também de performance. Os níveis podem ser expressos como metas para alcançar algum estado superior de confiabilidade e, para isto, iremos necessitar de uma **estrutura de monitoramento**, juntamente com uma eficiente **gestão dos incidentes**, acrescida de uma **equipe de suporte** preparada para resolver estes incidentes e manter a estrutura em pleno funcionamento.

Basicamente, iremos necessitar de técnicas, ferramentas de monitoramento e pessoal técnico para aumentar a confiabilidade e torná-la possível para que a solução continue funcionando e os dados estejam disponíveis o maior tempo possível. O **tempo de atividade** é representado como uma proporção ou porcentagem da quantidade de quando a solução estiver operacional.

Poderemos obter alta disponibilidade e alta performance praticando os seguintes **princípios**:

- Identificar os pontos de falha através do monitoramento e eliminá-los;
- Implementar tolerância a falhas. Adicionar possibilidade de recuperação de falhas de forma imediata, através de redundâncias, desta forma, a solução ficará protegida e a recuperação rápida de falhas será possível;
- Utilizar réplicas para permitir que as requisições enviadas pelos usuários sejam atendidas de forma rápida.

2. A REPLICAÇÃO

Com o avanço da tecnologia das **redes de computadores**, foi possível disseminá-las, quebrando as fronteiras das corporações e proporcionando o surgimento das redes globais. A **internet** surge neste cenário, tornando-se um sucesso absoluto e, como consequência, aumentando a importância dos sistemas distribuídos.

O sucesso da internet é o fruto de muitos trabalhos de pesquisa em **sistemas distribuídos**, que ofereceram uma melhoria crescente no desempenho, velocidade e maior confiabilidade neste tipo de arquitetura de rede, que é flexível, gerenciável e conduziu a sua adoção pelas grandes **plataformas de desenvolvimento de software**.

Um fator primordial para a efetividade desta arquitetura é a possibilidade da **distribuição dos dados** em tempo real, ou seja, a **replicação das informações** nos mais variados destinos, tornando as informações disponíveis, com confiabilidade e com tolerância a falhas. A principal razão

para se utilizar das réplicas de dados é permitir o **armazenamento** de **dados críticos** em múltiplos servidores, permitindo que fiquem disponíveis mesmo que alguns apresentem falhas, pois existe um sistema que centraliza as informações possibilitando as correções que sejam necessárias.

Este benefício de poder manter as **cópias** sempre **atualizadas** exige a aplicação de técnicas e tecnologias de **sistemas distribuídos** que permitem esconder os aspectos que caracterizam uma replicação dos dados originados pelas transações feitas pelo usuário em um determinado local e que devem ser replicados para múltiplos destinos.

Toda e qualquer grande instituição, atualmente, com presença geográfica dispersa, necessita enviar seus **dados** para as várias **unidades** que a compõem. A principal razão pela qual se busca uma estrutura para utilizar réplicas é a necessidade da **disponibilização** do **sistema de banco de dados**, ou seja, implantar uma estrutura lógica que permita que os dados sejam acessados e modificados nos sistemas centralizados, assim como também nas várias unidades espalhadas, mantendo o banco de dados íntegro e gerenciável, permitindo que se tenha em cada uma das unidades componentes da instituição uma visão do todo.

Não somente existe a necessidade da disponibilização destes dados, oriundos de sistemas críticos nos vários locais geograficamente distantes, para a consulta de sistemas que são executados localmente, mesmo no caso que haja uma falha no sistema centralizado ou na própria estrutura selecionada para a criação destas réplicas. Mas há outras razões, por exemplo, a **centralização dos dados** que permite uma melhor gerência de *backups*; há também a necessidade de **disponibilização de informações** vindas das várias unidades para os sistemas de "*business intelligence*"; há a necessidade da **troca de informações** entre unidades, passando antes por um sistema de validação centralizado, e que fiquem disponíveis em vários locais, mesmo se o sistema centralizado sofrer falhas, como pode existir a necessidade de vários sistemas geograficamente espalhados enviar dados para o sistema centralizado, não somente por questões de segurança, no qual a principal razão para utilizar réplicas é aumentar a disponibilidade do sistema de banco de dados. Armazenando dados críticos em múltiplos sites, o sistema fica disponível mesmo se alguns deles estiverem em falha.

Outra razão é aumentar o **desempenho**. Já que existem muitas cópias de cada item de dados, uma transação pode encontrar o dado que ela precisa em um site mais próximo, quando comparado a um banco de dados com uma única cópia da informação.

A replicação de dados tem provado ser uma técnica útil em muitas aplicações em que o **acesso rápido** e **confiável** é imperativo. Porém, esses benefícios vêm em conjunto com a necessidade de atualizar todas as cópias dos itens de dados. Com isso, **operações de leitura** podem ser executadas mais rapidamente, mas **operações de escrita** são executadas mais lentamente.

Um banco de dados replicado é um banco de dados distribuído no qual cópias múltiplas de alguns itens de dados são armazenadas em sites diferentes, sendo que o sistema de banco de dados deve esconder os aspectos de replicação de dados das transações dos usuários.

2.1. Teoria da replicação

Numa estrutura de replicação, quando acontece uma transação no publicador, todas as **réplicas** devem ser **atualizadas**, e o sistema gerenciador de banco de dados é o responsável por esta operação. Esta atualização de réplicas pode ser feita de modo imediato, no momento que a transação inicia, ou pode só executar a atualização quando toda a transação acontece no publicador. Em função disto, existem dois **modelos básicos de replicação**, a imediata (*eager replication*) e a atrasada (*lazy replication*).

No primeiro caso, ou seja, na **replicação imediata**, todas as réplicas são mantidas **sincronizadas** e o procedimento acontece como se a replicação fosse parte da transação que originou a replicação. Apesar de ser uma forma ideal de manter as réplicas, este tipo reduz o desempenho das atualizações. Como o procedimento de atualização nos assinantes passa a fazer parte da transação, **mecanismos de controle** são acrescentados à transação, e isto significa mais tempo para concluir a operação. A replicação imediata não é uma boa opção para estruturas que possuem como nós dispositivos móveis, em função da característica destes dispositivos de se manterem desconectados. Para estes dispositivos, as atualizações devem ser feitas de forma assíncrona e depois da transação ter sido confirmada no publicador.

Nas **replicações atrasadas**, o sistema gerenciador de banco de dados utiliza uma **visão dos objetos não replicada**, para ser utilizada enquanto a transação acontece, para cada item de dado que a transação lê ou escreve, uma cópia destes dados é acessada pelo SGBD, podendo haver diferentes transações fazendo acesso a diferentes cópias dos objetos replicados do banco de dados.

É tarefa do **SGBD** atrasar a **distribuição** das operações de escrita nas cópias até que a transação termine e as cópias dos objetos replicados possam ser enviados aos destinos.

A replicação imediata, normalmente, utiliza um **esquema de bloqueios** para detectar e controlar a execução concorrente. A replicação atrasada, normalmente, usa um **esquema de concorrência** com múltiplas versões para detectar comportamento não serializável. Com isso, a replicação atrasada pode permitir que uma transação acesse um valor desatualizado.

No que diz respeito às atualizações das cópias, na replicação, estas podem ser feitas de duas maneiras: a atualização em grupo ou a atualização pelo mestre.

Na **atualização em grupo**, qualquer nó que possua uma cópia do objeto do banco replicado pode atualizá-la. Neste caso, esta atualização é conhecida como atualização em qualquer lugar (*update anywhere*). Os conflitos de atualização das réplicas têm mais chance de acontecer neste tipo de atualização.

Na **atualização pelo mestre**, cada objeto tem um nó mestre e somente ele pode atualizar a cópia primária do objeto. Todas as outras réplicas são para leitura somente. Outros nós querendo atualizar o objeto devem requisitar a atualização ao mestre.

As características da replicação atrasada com atualização em grupo incluem a possibilidade de qualquer nó poder atualizar qualquer objeto replicado. Quando uma **transação** é finalizada, uma transação é enviada a cada um dos **nós da estrutura** de replicação com a responsabilidade de fazer as atualizações feitas no publicador para cada réplica dos objetos do banco de dados. Aqui, existe um fator complicador, visto que é possível que dois dos nós atualizem o mesmo objeto e enviem suas atualizações para outros nós, portanto, um mecanismo de replicação deve detectar o problema e manter o sistema de replicação confiável para que não haja perda de atualizações.

No caso da replicação atrasada com atualização pelo mestre, cada um dos objetos replicados possui um **dono**. As atualizações devem ser feitas pelo dono e, posteriormente, estas devem ser propagadas para as outras réplicas envolvidas no processo de replicação. Nota-se que **diferentes objetos** têm **diferentes donos**. Da mesma forma que na replicação imediata, o esquema deferido com o mestre não possui problemas de reconciliação, ao invés disso, os conflitos são resolvidos por espera ou por "*deadlocks*" (impasse entre dois ou mais processos que ficam impedidos de continuar suas execuções, ou seja, ficam bloqueados).

- **Algoritmos de replicação imediata**

Na **replicação síncrona** tanto a alteração no objeto quanto a propagação desta alteração para os demais objetos fazem parte de uma mesma transação distribuída e atômica, ou seja, a atualização dos objetos de destino ocorre simultaneamente à atualização do objeto de origem e ela só é confirmada após a efetivação em todos os objetos envolvidos na replicação. Os objetos replicados se comunicam utilizando protocolos especializados, para manter o conjunto de dados consistentes.

- **Técnica *write-all***

Esta técnica requer que o sistema gerenciador de banco de dados processe cada **operação de escrita**, enviando os itens de dados para todas as réplicas, mesmo que alguma delas apresente problemas. Na ocorrência de uma falha, o processamento como um todo é atrasado, pois está técnica exige a escrita em todas as réplicas de uma única vez. Pode-se notar que quanto maior a estrutura de replicação, havendo falha, menor será a disponibilidade do sistema.

- **Técnica *write-all-available***

Nesta técnica, o sistema gerenciador de banco de dados deve **escrever** em todas as **réplicas** que se encontram disponíveis no momento, portanto, ignorando aquelas com falhas. O problema da disponibilidade do sistema é resolvido, mas surge uma questão que é a de causar problemas de **sincronização**. Nas cópias com falhas, as informações estarão desatualizadas em relação àquelas que conseguiram atualização. Para corrigir esta anomalia, existem vários algoritmos.

- **Algoritmos de cópias disponíveis**

Este algoritmo trata **dados replicados** usando variações da técnica *write-all-available*.

Portanto, toda operação de leitura é traduzida em uma operação de leitura em qualquer réplica e toda operação de escrita é traduzida em operações de escrita em todas as réplicas disponíveis na estrutura de replicação.

Esses algoritmos tratam **falhas** nos sites, mas não tratam falhas de **comunicação**.

Assume-se que existe um conjunto fixo de réplicas para cada item de dados replicados, conhecido por todos os sites e que este conjunto não muda dinamicamente.

Operações de escrita cujas respostas não foram recebidas são chamadas de *missing writes*. Se qualquer rejeição for recebida, ou se todas as operações de escrita cuja resposta não foi recebida, a operação de escrita é rejeitada e a transação deve abortar. Caso contrário, a operação de escrita é bem-sucedida.

- **Algoritmo de cópias disponíveis orientado a diretório**

No algoritmo descrito anteriormente, a distribuição das réplicas nos sites é estática. No caso do algoritmo aqui discutido, há a utilização de **diretórios** que definem o conjunto de sites que armazenam réplicas em uma estrutura de replicação em um determinado momento.

De uma forma resumida, para cada estrutura de replicação, existem diretórios que possuem, cada um, sua lista de réplicas. Estes diretórios são atualizados por duas transações especiais, uma de inclusão das réplicas nos diretórios e outra para excluí-las.

- **Algoritmo usando consenso do quórum**

O princípio deste algoritmo é a atribuição de um **número de versão** a cada réplica, que é igual a zero inicialmente. Quando o sistema de replicação processa uma operação de escrita, por exemplo, ele determina o maior número de versão de todas as réplicas nas quais ele vai escrever e acrescenta um, atribuindo um novo número de versão a cada que foi alvo de uma operação da replicação. Isso requer a leitura de todas as cópias no quórum de escrita antes de escreve em qualquer uma delas. O propósito dos quóruns é garantir

que operações de leitura e de escrita que acessam o mesmo item de dado também acessem pelo menos uma **réplica em comum** desse item de dado. Todo par de operações conflitantes será sempre sincronizado por algum **escalonador**, aquele que controla o acesso à cópia pertencente à interseção dos quóruns.

Uma desvantagem deste algoritmo surge quando uma transação deve acessar **cópias múltiplas** de cada item de dado que ela quer ler, até mesmo quando existe uma cópia do dado em seu site original. Em muitas aplicações, as transações leem mais de que escrevem. Nesse caso, o desempenho desse algoritmo não é considerado bom. Outro problema reside no fato que todas as réplicas de um item de dado devem ser conhecidas a princípio. Uma réplica pode se recuperar de uma falha, mas uma nova réplica não pode ser criada de forma ágil, porque alteraria a definição dos quóruns.

- **Algoritmo de partição virtual**

Este algoritmo foi projetado de tal forma que uma **transação** nunca precise acessar mais de uma **cópia** de um item de dados para leitura. Com isso, a cópia disponível que estiver mais perto pode ser usada para leitura. Como no algoritmo usando consenso de quórum, cada réplica tem um peso e cada item de dado tem padrões de leitura e escrita.

A ideia básica deste algoritmo é manter uma **visão** para cada **site** para os quais há a necessidade de comunicação. Na realidade, há a utilização da técnica *write-all*, ou seja, ele traduz uma operação de escrita no item de dado replicado, em operações de escrita em todas as réplicas, e traduz uma operação de leitura, também no item de dados, em uma operação de leitura em qualquer das réplicas. Manter essas visões de maneira consistente não é um problema simples, pois falhas nos sites e na comunicação ocorrem espontaneamente.

O algoritmo de partição virtual usa uma transação especial chamada **Atualização de Visão** (*View Update*) para superar essa dificuldade, que é requisitada pelo próprio sistema gerenciador de banco de dados.

- **Algoritmo de replicação de dados adaptativa**

Este tipo de replicação modifica o esquema de **replicação dos objetos** de acordo com as modificações no **padrão de leitura e escrita** dos objetos. O algoritmo modifica continuamente o esquema de replicação, convergindo para um esquema otimizado.

Normalmente, os esquemas de replicação são estabelecidos no momento do **projeto**, ou seja, de forma estática, permanecendo desta forma até que seja necessário modificar o projeto. A modificação do projeto se dá quando há a necessidade de modificar o número de réplicas ou a sua localização.

Este tipo de algoritmo só é apropriado para sistemas de replicação com padrões de leitura e escrita de objetos do **banco de dados fixos**, se houver um comportamento dinâmico haverá sérios problemas de desempenho.

A técnica de ler uma cópia e escrever em todas impede operações de escrita de executarem quando ocorrem falhas no sistema. Para permitir essas operações de escrita mesmo em caso de falha, propõe um protocolo chamado de ***Primary Missing Writes***, que pode ser combinado com a replicação dinâmica.

- **Algoritmos de replicação atrasada**

Na **replicação assíncrona**, a atualização dos objetos de destino não ocorre simultaneamente à atualização do objeto de origem, existindo uma latência nessa operação, pois as atualizações são registradas, enfileiradas e executadas nos objetos de destino somente em um segundo momento, em uma outra transação separada. A replicação assíncrona visa resolver as **limitações** existentes na **replicação síncrona**, porém, para isto, ela sacrifica a consistência dos dados, tendo em vista que a atualização é executada apenas localmente, como se não existissem réplicas, isto possibilita uma rápida resposta sobre a transação, porém, causa diferenças entre os objetos replicados por um período de tempo.

- **Protocolos de acesso multivisão para replicação em larga escala**

Este tipo de protocolo é utilizado em uma arquitetura organizada em **árvore estruturada**. A raiz da árvore representa os itens de dados que serão distribuídos, enquanto as folhas da árvore representam os sites individuais que receberão as réplicas de dados.

Os agrupamentos de sites formam os ***clusters***, que são os nós internos da estrutura de replicação. Esta formação deve seguir o **princípio de recurso limitado**.

A demanda de serviço independe do número de nós do sistema, mas existem regras que devem ser seguidas:

- O número de site em cada *cluster* deve ser pequeno, geralmente, menor que 10;
- O custo de comunicação é menor entre sites dentro de um cluster;
- Os sites em um *cluster* devem estar relacionados entre si, de tal modo que as solicitações de serviço dos usuários de um *cluster* sejam locais a este *cluster*.

Na estrutura de replicação que utiliza este protocolo existe também o conceito de **site coordenador**, um para cada *cluster*. Este site coordenador atualiza as cópias primárias no *cluster*. Cada transação atualiza vários dados replicados que podem ter diferentes origens, para isto, o *cluster master*, que é onde a transação se inicia, precisa enviar mensagens de atualização para cada site primário do item de dado replicado.

O site coordenador coordenará as operações de **atualização das cópias primárias** no *cluster* com os coordenadores dos outros *clusters*.

Além disso, o site coordenador de um cluster é também responsável por gerar as **transações de propagação** para atualizar as cópias dos dados nos sites membros ou nos *clusters* de nível mais baixo. A transação é executada e confirmada independentemente da sua transação original. O site coordenador é responsável por reemitir a transação se esta falhou anteriormente. É importante notar que o site coordenador pode ser independente quanto aos dados. Em geral, existe somente um site coordenador por *cluster*, representando todos os dados replicados no cluster. O site coordenador de um *cluster* deve ter uma comunicação entre ele e os outros sites, de forma eficiente e confiável.

A estratégia básica das arquiteturas que utilizam este protocolo é de propagar as **atualizações** dos dados replicados em um **modelo *cluster-by-cluster***, desmembrando as transações originais em um conjunto de transações independentes e confirmadas e que estejam relacionadas entre si. O processo de atualização inicia através do ***cluster*** de mais **alto nível** que contém todas as cópias dos dados replicados, que deverão ser atualizados e propagados para os *clusters* de **nível mais baixo**. Em cada *cluster*, o protocolo primeiro atualiza as cópias de dados dos sites membros e as cópias

primárias dos *clusters* membros. A seguir, gera transações para os *clusters* membros, que continuam o processo de atualização até que todas as cópias tenham sido atualizadas. De forma diferente das transações de atualização, as **transações de leitura** podem ser processadas com mais flexibilidade. Elas podem escolher a leitura das cópias dos dados ao longo dos diferentes níveis de *clusters* onde a cópia exista, decidindo isso a partir de seus requerimentos correntes, de performance e consistência. Neste protocolo, será feita a distinção de três tipos diferentes de sites existentes em um *cluster*: o **site mestre**, onde a transação é iniciada, o **site coordenador**, onde as atualizações dos dados replicados são coordenadas, e os **sites escravos**, onde as operações de leitura e escrita são realmente implementadas.

- **Replicação *two-tier***

Neste tipo de replicação são identificados dois tipos de **nós**: os **móveis**, que estarão desconectados a maior parte do tempo, pois se conectam, armazenam uma réplica do item de dados replicado e depois se desconectam; e os nós que se mantem sempre **conectados**.

Nos nós móveis podem existir uma versão mais recente da réplica do item de dados replicados e versões experimentais que são atualizadas por transações experimentais e, neste caso, podem estar desatualizadas.

Quanto às **transações**, elas podem ser de **base**, que só acessam aos dados mestre, e as **experimentais**, que produzem versões experimentais e transações de base a serem executadas posteriormente. Os usuários estão cientes de que todas as atualizações são experimentais até que a transação se torne uma transação base e se esta falhar, o usuário deve revisar e submeter novamente a transação.

Quando um **nó móvel** se conecta a um **nó base**, ele executa os seguintes procedimentos:

- Descarta suas versões experimentais, já que elas serão atualizadas pelos mestres;
- Envia atualizações nas réplicas dos objetos cujo mestre é o nó móvel para o nó base conectado;
- Envia todas as transações experimentais ao nó base para serem executadas na ordem em que confirmaram no nó móvel;
- Aceita atualizações nas réplicas vindas do nó base;

- Aceita a resposta de sucesso ou fracasso de cada transação experimental.

O **nó base** na outra ponta da conexão, quando solicitado por um nó móvel, executa os seguintes procedimentos:

- Envia as transações de atualização atrasadas ao nó móvel;
- Aceita transações de atualização atrasadas dos objetos cujos nós mestre são o nó móvel;
- Aceita a lista de transações experimentais, suas mensagens de entrada e seus critérios de aceitação. Executa cada transação experimental de acordo com a ordem em que elas confirmaram no nó móvel. Se a transação base falhar nos critérios de aceitação, ela é abortada e uma mensagem de diagnóstico é enviada ao nó móvel;
- Depois que o nó base compromete uma transação base, ele propaga as atualizações de forma atrasada para todos os outros nós, usando o mecanismo deferido com mestre;
- Quando todas as transações experimentais forem reprocessadas como transações base, o estado nó móvel converge para o estado do nó base.

As principais propriedades do esquema de replicação utilizando a **estrutura de replicação *two-tier*** são:

- Os nós móveis podem fazer atualizações experimentais no banco de dados;
- Transações base executam com serialização usando uma única cópia, então, o estado do sistema base mestre é o resultado de uma execução serializável;
- Uma transação se torna durável quando a transação base completa sua execução;
- Réplicas em todos os nós conectados convergem para o estado do sistema base.

- **Protocolos usando grafos de replicação**

Este protocolo tem como características principais:

- Para cada item de dado a ser replicado, existe uma cópia primária em um site primário que é o responsável por atualizações nesse item de dado. As outras cópias são chamadas cópias secundárias;
- O banco de dados local em cada site garante as propriedades de atomicidade, consistência, isolamento e durabilidade para as transações e gerencia os *deadlocks* locais;
- O site no qual uma transação é submetida é chamado de site original da transação e ela só pode atualizar um dado se ela foi originada no site primário deste dado;
- Quando a cópia primária de um dado é atualizada, essa operação deve ser propagada para as cópias secundárias que possuem a cópia do dado, depois que a transação original for comprometida.

- Modelos de replicação

- **Replicação mestre-escravo** (*master-slave*)

Neste tipo de replicação, um dos objetos é considerado como **mestre** e as **alterações** só podem ser realizadas nesse objeto. Os demais objetos são considerados como **escravos** e só podem ser utilizados para **leitura**. Neste modelo de replicação, o **fluxo das informações** sempre se dá do objeto mestre para os objetos escravos, configurando-se como uma arquitetura de menor complexidade. Eventualmente, pode-se ter nesse tipo de modelo o cascateamento entre objetos escravos, isto é, um objeto escravo que recebe as modificações do objeto mestre pode repassar essas modificações para outros objetos escravos.

- **Replicação multimestre** (*multimaster*)

Na replicação multimestre (*multimaster*), todos os **objetos** podem ser **modificados** e essas modificações são transmitidas para os demais objetos. Neste modelo de replicação, o fluxo das informações pode partir de qualquer objeto participante para os demais e, por isso, a complexidade de controle da replicação é maior.

2.2. Replicação no MySQL

Como já foi discutido na parte teórica, o problema básico que a replicação se propõe a resolver é o de manter os **dados** de um **servidor mestre**, que recebe as transações de dados, sincronizado com vários outros servidores que são as réplicas, recebendo dados destes servidores mestres. Muitas réplicas podem se conectar a um único mestre e permanecer em sincronia com ele, e uma réplica pode, por sua vez, atuar como mestre, isto só depende da configuração da estrutura. No caso do MySQL, ele oferece dois **tipos de replicação**:

- Replicação baseada em instrução;
- Replicação baseada em linha.

Ambos os tipos de replicação funcionam gravando **informações** em **arquivos de *log*** do servidor mestre e as transfere para os servidores réplicas (os servidores réplicas, geralmente, são chamados de "escravos"). A **operação de replicação** é assíncrona, ou seja, a cópia dos dados da réplica não tem garantia de estar atualizada a qualquer momento. Não há garantias de quão grande é a **latência** na réplica, ou seja, o tempo para a atualização das réplicas pode variar muito, dependendo de inúmeros aspectos, inclusive da disponibilidade da rede.

A seguir, iremos fazer algumas considerações sobre a replicação no MySQL:

- Muito utilizada para a distribuição de dados, em termos de consumo de largura de banco nas redes de computadores, a replicação baseada em instruções tem uma maior economia, quando comparada com a replicação baseada em linha;
- A replicação é muito útil para a manutenção de cópias de dados em localização de dados geograficamente diferentes, por exemplo, na construção de um "*site backup*", no qual havendo um problema no banco de dados principal, este possa assumir as aplicações;
- Pode-se fazer replicação para servidores de *business intelligence*, em plataformas onde o banco de dados de produção está sobrecarregado, as consultas para a geração de relatórios podem ser direcionadas para uma das réplicas, portanto, fazendo um **balanceamento de carga** de processamento, desafogando o servidor principal;

- A replicação apesar de não ser um substituto para as cópias de segurança (*backup*), ela pode ser uma técnica valiosa para manter os dados em vários servidores;

- No caso da gerência de falhas e da alta disponibilidade, a técnica de replicação pode ser um fator que irá colaborar grandemente com o seu sucesso.

A **replicação no MySQL**, de uma forma resumida, pois as operações de replicação foram discutidas na parte teórica no início das discussões sobre replicação, funcionam da seguinte forma:

- No servidor mestre, as transações normais do banco de dados acontecem e são escritas em um arquivo de *log*; estes registros podem ser chamados de eventos de *log* binários. Mesmo que as instruções nas transações sejam intercaladas, durante a execução, elas serão serializadas na gravação do *log* binário;

- Com os arquivos de log gravados no servidor mestre, as réplicas copiam estes eventos *log* binário do mestre para seu arquivo de log correspondente (*relay log*);

- Com os eventos de *log* binário copiados para os seus arquivos de *log*, as réplicas reproduzem estes eventos nos seus bancos de dados, aplicando as alterações em seus próprios dados.

Nesta arquitetura de replicação utilizada pelo MySQL, não há acoplamento entre os **processos de busca** no servidor mestre e a **reprodução** de destes eventos na réplica, daí eles serem assíncronos.

Aqui, devemos chamar a atenção para o fato de que na réplica, a replicação é serializada, não importando se no servidor mestre o processo de atualização do banco de dados tenha sido em paralelo. Isto pode ser um fator causador de problemas de desempenho no mecanismo de replicação.

- **Configuração da replicação no MySQL**

Como vimos na parte teórica da replicação, ela é um processo bastante complexo, no entanto, para configurar a replicação no MySQL não chega a ser uma tarefa difícil, caso o cenário projetado não seja de alta complexidade. No caso de um **cenário básico**, onde a estrutura é composta de um servidor mestre com algumas réplicas, para fazer a **configuração** deste tipo de cenário, os passos são os seguintes:

- Configurar as contas que serão utilizadas na replicação em cada um dos servidores;
- Configurar o servidor mestre e suas réplicas;
- Configurar e testar as réplicas, de modo que se conectem ao mestre com sucesso e faça a replicação sem erros; instrua a réplica para se conectar e replicar a partir do mestre.

Neste caso da estrutura básica, muitas das **configurações** já vêm como **padrão** nas telas de configuração da replicação do MySQL, bastando apenas aceitá-las.

3. *Cluster*

Em computação o termo *cluster* significa um **conjunto de computadores conectados** funcionando como se fosse um único, portanto, tendo-se como resultado a soma de seus recursos. Cada um dos computadores que compõem um *cluster* são chamados de nós (*nodes*). Esta **estrutura de nós** garante uma alta disponibilidade acrescida de uma alta confiabilidade, pois um dos nós falhando, o processamento é transferido para um outro, sem haver o comprometimento no funcionamento da estrutura.

Via de regra a necessidade deste tipo de configuração de computadores se verifica quando temos aplicações que exigem **processamento** em **alta escala**, e com alta disponibilidade. Existem também as estruturas com o propósito de **balanceamento de carga** de processamento.

Podemos classificar as estruturas de cluster, como mostrado a seguir:

- ***Cluster* de alto desempenho**

O objetivo deste tipo de cluster é permitir o processamento de um **grande número de instruções** em tempo aceitável, ou seja, ele é recomendado para aplicações exigentes, por exemplo aplicações científicas que precisam analisar uma grande variedade de dados, vindos de diversas fontes, e realizar cálculos complexos que exigem um certo poder de processamento por parte dos computadores envolvidos.

- ***Cluster* de alta disponibilidade**

A escolha por este tipo de cluster está sempre relacionado a **sistemas** cujo funcionamento é **crítico**, que precisam estar em funcionamento o maior tempo possível, evitando ao máximo as paralisações, por exemplo, os sistemas de missão crítica que exigem um índice de funcionamento anual igual ou superior a 99,99%, ou seja, o sistema apresenta uma alta disponibilidade.

Podemos dizer que a disponibilidade é a percepção do usuário final sobre a aplicação. Portanto, disponibilidade é a **acessibilidade** que o usuário tem em uma aplicação ou serviço e para atender a esta exigência dos usuários são necessários diversos recursos, como as ferramentas de monitoramento que são extremamente importantes, pois identificam problemas nos nós, ou as falhas de conexão.

- ***Cluster* de balanceamento de carga**

A característica principal deste tipo de *cluster* é a **distribuição** mais uniformemente possível dos **recursos** de processamento entre os nós, pois o importante é que cada nó do *cluster* atenda a uma requisição e não que compartilhe requisições com outros nós. Quando se fala em balanceamento, fica implícita a ideia de um equilíbrio entre as requisições que são enviadas ao *cluster*, sempre direcionando as requisições para os nós que estão com uma menor quantidade de tarefas. Muitas são as oportunidades de utilização deste tipo de *cluster*, mas o mais comum é na internet, exatamente por causa da característica de distribuição de tarefas, tendo, portanto, uma tolerância maior ao aumento instantâneo do número de requisições.

- **Considerações sobre a utilização dos clusters**

  - A **relação custo/benefício** é muito observada, já que se pode utilizar computadores mais baratos e ter resultados tão bons ou até superiores à utilização de servidores sofisticados;

  - Não há nenhuma exigência que utilizemos qualquer tipo de equipamento de um determinado fornecedor. Portanto, não há o risco de se ter problemas quanto a **fornecedores** ou **prestadores de serviço**. Havendo problemas com um determinado tipo de equipamento, poderá ser substituído por outro sem maiores exigências.

- A **configuração** de uma estrutura de *cluster* pode não ser muito simples, dependendo de como foi projetada, mas por outro lado a utilização de **supercomputadores** também pode apresentar trabalho em sua configuração além da exigência de pessoal especializado;
- Existem soluções de *cluster* disponibilizados em **software livre**, o que facilita o uso em escolas e o treinamento de pessoal para sua implantação;
- Uma grande vantagem desta estrutura é a possibilidade de se poder adicionar ou retirar nós de um *cluster*, sem prejudicar ou interromper o seu funcionamento;
- Há também uma relativa facilidade de **customização** para atender a determinadas características de certas aplicações;
- O *cluster* pode atender a **ambientes computacionais** bem simples, como também a ambientes computacionais bem sofisticados;
- O **aumento do número de nós** em um *cluster* aumenta também sua complexidade e sua necessidade de manutenção, além de exigir um maior espaço físico;
- Com o crescimento do *cluster*, a **estrutura de comunicação** também deve ser verificada, pois pode acontecer de ocorrerem gargalos na rede, em função de aplicações exigentes que necessitam de uma alta taxa de transferência de dados;
- A base de um *cluster* é sempre uma **rede local**, com isto, há a limitação de não poderem ser incluídas computadores localizados geograficamente distantes;
- O **planejamento** será sempre importante quando da decisão por utilizar um *cluster*, dependendo da aplicação que irá trabalhar na estrutura, ele pode não ser a melhor solução.

> “Com o crescimento do *cluster*, a estrutura de comunicação também deve ser verificada...”

- ***Cluster* no MySQL**

Atualmente, os sistemas gerenciadores de dados modernos trazem **funcionalidades** para trabalhar com seus servidores em *cluster*, no caso do MySQL, ele disponibiliza uma solução completa de alta disponibilidade conhecida como ***cluster* InnoDB**, que foi projetado para facilitar a configuração, o uso e as rotinas de manutenção. Para isto, vários componentes trabalham juntos para disponibilizar estas funcionalidades avançadas, os principais são:

➢ **Replicação de grupo (*group replication*)**

Adiciona um protocolo de comunicação ativo que fornece níveis elevados de disponibilidade, incluindo também um mecanismo de tolerância a falhas que trabalha de forma automática.

➢ **MySQL Shell**

É uma ferramenta cliente que permite que se crie interfaces para o *cluster* innoDB, utilizando linguagens como o SQL, Java script e linguagem Python.

➢ **X DevAPI**

Ferramenta de interface de programas de aplicação que permite que aplicativos interajam com os dados de forma programática.

➢ **AdminAPI**

É uma interface de programação de aplicações especial que permite a interação do *cluster* innoDB através do MySQL Shell.

➢ **MySQL router**

É um *middleware* que permite o roteamento transparente entre uma aplicação e os servidores MySQL. Pode ser utilizado em uma grande variedade de aplicações, oferece alta disponibilidade e escalabilidade.

> “...os sistemas gerenciadores de dados modernos trazem funcionalidades para trabalhar com seus servidores em *cluster*...”

4. Monitoramento

O monitoramento em uma estrutura de bancos de dados de **alto desempenho**, com servidores de banco de dados configurados em *cluster* e com replicação, exige uma organização da equipe responsável por sua manutenção. Iremos definir a seguir uma estrutura de monitoramento, baseada em ITIL. O **ITIL** é um conjunto de práticas de gerenciamento de projetos em equipes de suporte. Esta sigla significa "Biblioteca de infraestrutura de tecnologia da informação" (*techonology infraestructure library*) e surgiu na Inglaterra nos anos 80, com o objetivo de padronizar as melhores práticas para a gestão em tecnologia da informação. Estas práticas, atualmente, possuem alta credibilidade nas organizações ao redor do mundo, trazendo enormes benefícios para o trabalho das **equipes de manutenção** de **grandes e complexas estruturas**. Podemos citar algumas delas:

- Otimização dos processos de gestão das equipes de tecnologia da informação;
- Aumento da produtividade das equipes de manutenção;
- Há uma melhor comunicação entre as equipes, através da padronização de uma nomenclatura quando se trata das requisições e incidentes;
- Tempo reduzido no tratamento dos incidentes;
- Os níveis de serviços são elevados;
- Redução de custos nos processos internos;
- Entre outros.

No caso do monitoramento de uma **estrutura em *cluster* e replicada**, de nada adianta monitorar se não conseguirmos agir sobre os problemas, então, em grandes estruturas, teremos no mínimo as equipes a seguir:

> "O ITIL é um conjunto de práticas de gerenciamento de projetos em equipes de suporte."

- **Equipe de monitoração da replicação do *cluster***

A equipe de monitoração da replicação e também do *cluster* é responsável pela utilização das ferramentas de monitoração do SGBD e também do *cluster*. Esta atividade consiste em utilizar uma ferramenta de monitoramento para verificar as replicações que apresentam falha ou desempenho crítico e, ainda, as ocorrências no funcionamento dos servidores em *cluster*. Ao ser detectada alguma falha, a equipe de monitoração abre um incidente, detalhando a mensagem de erro apresentada pela ferramenta de monitoramento e envia o incidente para a equipe de classificação dos incidentes.

- **Equipe de classificação de incidentes**

A equipe de classificação de incidentes é responsável por estudar o que exatamente ocorreu no incidente para identificar o perfil do profissional de suporte que irá atuar na resolução do problema. Como exemplo, uma falha na estrutura de replicação não necessariamente é um problema da replicação em si, pode ter ocorrido uma queda no circuito de comunicação entre o servidor mestre e suas réplicas.

- **Equipe de especialistas em replicação**

Esta equipe é composta por especialistas em banco de dados SGBD, dominando todas as técnicas e ferramentas utilizadas na configuração e manutenção das replicações. Assim como uma equipe de ambiente operacional especialista em estruturas de *cluster*. Esta equipe além de ser responsável pela resolução dos problemas detectados pelo monitoramento, será também responsável pela implantação de mudanças necessárias nas estruturas, seja ela estrutura de replicação, ou na estrutura de *cluster*.

- **Equipe de mudanças**

A equipe de mudança formada por especialistas em replicação e em *cluster* receberá as solicitações de mudanças vindas de especialistas que estudaram os problemas detectados pelo monitoramento, elegem as mudanças necessárias e as executam.

- **Cliente**

O cliente é qualquer usuário que utilize os sistemas e que tenha permissão para solicitar mudanças, tenham condições de analisar as requisições de construção e teste e validá-las, autorizando a sua execução.

4.1. Descrição dos processos de monitoramento e suporte

O processo de **gerência** da **estrutura de replicação** é baseado no **ITIL**, e deve existir um sistema de gestão de chamados que permitirá que as várias equipes troquem documentos nas várias fases do processo de monitoramento e manutenção da arquitetura de replicação.

- **Realizar procedimentos básicos de manutenção da replicação**

Responsável: equipe de monitoramento da replicação.

A equipe de monitoração ao perceber um alarme de falha na ferramenta de monitoramento da replicação ou do *cluster* deve fazer uma verificação geral para certificar-se que a indisponibilidade não foi temporária, portanto, não havendo a necessidade de acionamento dos especialistas.

- **Consultar ferramentas de monitoração da replicação**

Responsável: equipe de monitoração da replicação.

O trabalho corriqueiro da equipe de monitoração é obter os dados sobre o que ocorre tanto na estrutura do *cluster* quanto na estrutura de replicação, e isto deve ser feito durante 7 dias por semana 24 horas por dia. Havendo a identificação de falhas deve ser gerado um incidente.

- **Receber informações de problemas na estrutura de replicação**

Responsável: equipe de monitoração da replicação.

Pode acontecer que usuários do sistema identifiquem problemas que a ferramenta de monitoração não consegue identificar, por exemplo, uma atualização errada de registros, o que significa que os registros originais também estão errados. Este tipo de anomalia não pode ser identificado pela ferramenta de monitoramento, então, o cliente deve informar diretamente à equipe de monitoramento que o problema existe, para que ela inicie o procedimento de resolução do incidente.

> “O processo de gerência da estrutura de replicação é baseado no ITIL...”

- **Gerar documento de ocorrência de incidente na replicação ou no *cluster***

Responsável: equipe de monitoração da replicação.

Detectado o incidente, a equipe de monitoração faz os procedimentos iniciais para certificar-se de que não houve uma indisponibilidade temporária e gera um documento contendo as mensagens retornadas pela ferramenta de monitoração, ou no caso de ter sido informado pelo cliente, relatar os detalhes descritos pelo cliente para que o especialista seja informado corretamente.

- **Enviar documento de abertura de incidente para equipe de classificação**

Responsável: equipe de monitoração da replicação.

Quando a equipe de monitoração identifica um problema, deve abrir um incidente e enviar à equipe de classificação para que ele seja enviado para o especialista com o perfil necessário para resolver o incidente.

- **Receber documento de abertura de incidente de replicação ou do *cluster* da equipe de monitoração**

Responsável: equipe de classificação.

A equipe de classificação recebe da equipe de monitoração o incidente aberto e verifica para qual especialista ele deverá ser enviado, visto que existem vários especialistas com diferentes perfis, portanto, no caso selecionar uma especialista em *cluster* ou em replicação. Ela também verifica se a descrição do problema está legível e certifica-se que ela estará clara para o especialista.

- **Classificar incidente**

Responsável: equipe de classificação.

A equipe de classificação envia o incidente aberto para a fila de incidentes do especialista que possui o perfil para atuar nele, ou seja, é um especialista em cluster ou em replicação, conforme o caso.

- **Enviar incidente para a fila de atendimento do especialista em replicação e *cluster***

Responsável: equipe de classificação.

Identificado para qual especialista o incidente será enviado, a equipe de classificação coloca o incidente na fila do especialista identificado.

- **Receber solicitação de elaboração de requisição de construção e teste para mudança proposta na estrutura de replicação e *cluster* vindo da equipe de mudança**

Responsável: equipe de especialistas em replicação e *cluster*.

A equipe de especialistas recebe a solicitação de mudança na estrutura de replicação e *cluster*, esta mudança deverá ser feita inicialmente em ambiente de teste. O especialista em replicação e *cluster* receberá da equipe de mudança uma solicitação para que execute a mudança em ambiente de teste, verifique o impacto na estrutura de replicação e *cluster*, crie um plano de contingência para o caso de a mudança provocar problemas na estrutura de replicação e *cluster* e crie um detalhamento da execução da mudança. Tudo é documentado em uma requisição de construção e testes que ao ser finalizada será enviada ao cliente para que ele aprove o que o especialista escreveu no documento.

- **Receber da equipe de mudança solicitação de implantação de mudança na replicação e *cluster* aprovada**

Responsável: equipe de especialistas em replicação e cluster.

Após ter sido aprovada a requisição de construção e testes pelo cliente, é criada uma requisição de implantação e a requisição de construção e testes é anexada, pois nela existem o detalhamento da execução e o plano de contingência nos quais o especialista deve fazer consultas quando for fazer a implantação da mudança em produção. O especialista em replicação e *cluster*, então, executa em produção a mudança detalhada na construção e teste, documentando os passos da implantação na requisição de implantação.

> “O especialista em replicação e cluster receberá da equipe de mudança uma solicitação para que execute a mudança...”

- **Implantar mudanças na estrutura de replicação e *cluster***

Responsável: equipe de especialistas em replicação e *cluster*.

Após a aprovação da requisição de construção e teste, esta é enviada ao especialista para que ele implante a mudança seguindo os detalhamentos da execução que foram feitos na fase de construção e testes, documentando quaisquer problemas que por acaso aconteçam durante esta implantação

- **Finalizar mudança na replicação e *cluster* com sucesso**

Responsável: equipe de especialistas em replicação e *cluster*.

O atendimento de uma requisição de implantação, na grande maioria das vezes, é executado com sucesso. Portanto, ela é finalizada e envida ao cliente para a aprovação.

- **Colocar solicitação de implantação de mudança em estado pendente**

Responsável: equipe de especialistas em replicação e *cluster*.

No atendimento de uma requisição de implantação podem ocorrer fatos que impeçam a sua execução. Pode acontecer, por exemplo, que a modificação de um objeto de banco de dados, como uma visão, apresente problemas na execução do *script* de alteração, ou no caso de interrupção de circuitos de comunicação. Nestes casos, a mudança ficará em estado pendente até que os fatores que impediram sua execução sejam favoráveis.

- **Verificar filas de requisições**

Responsável: equipe de especialistas em replicação e *cluster*.

O especialista em replicação e *cluster* deve verificar as filas de requisição, uma vez que as requisições de construção e testes, para as mudanças solicitadas pelo cliente, e as requisições de implantação das mudanças são enviadas para as filas de requisição dos especialistas.

- **Verificar filas de incidentes**

Responsável: equipe de especialistas em replicação e *cluster*.

É uma das atividades principais do especialista em replicação e *cluster* a verificação da fila de incidentes, uma vez que surgindo o incidente, ele é enviado para a fila do especialista e este tem um tempo para ser atendido.

- **Resolver o problema na replicação e *cluster* relatado pelo incidente**

Responsável: equipe de especialistas em replicação e *cluster*.

O especialista, recebendo o incidente, o analise e aplica as técnicas de resolução de incidentes, utilizando as ferramentas disponibilizadas pelos SGBD. Após resolver o incidente, o especialista faz teste de verificação para certificar-se que o incidente realmente foi resolvido e finaliza, para que seja enviado ao cliente para que ele valide.

- **Finalizar atendimento de incidente com sucesso**

Responsável: equipe de especialistas em replicação e *cluster*.

Se foi possível a resolução do incidente, não havendo problemas que impeçam a sua resolução, o especialista finaliza o incidente, documentando a solução.

- **Colocar resolução do incidente como pendente**

Responsável: equipe de especialistas em replicação e *cluster*.

Não havendo possibilidade de resolução do incidente, o especialista coloca o incidente em estado pendente e documenta a razão pela qual o incidente não foi atendido. Como exemplo, um incidente pode deixar de ser atendido por questões de conectividade com um assinante da replicação e *cluster* remoto, ou porque a descrição do incidente não está clara não permitindo ações para resolver.

- **Receber as solicitações de mudanças na estrutura de replicação e *cluster***

Responsável: equipe de mudança.

O cliente detecta uma necessidade de mudança na estrutura de replicação e *cluster*. Por exemplo, houve uma atualização de versões de um sistema, o que resultou na necessidade de modificação de objetos do banco de dados. É gerada uma requisição de mudança pelo cliente e enviada à equipe de mudança para as providências.

- **Solicitar validação da requisição de implantação ao cliente**

Responsável: equipe de mudança.

Quando o especialista atende a uma requisição de implantação com sucesso, esta vai para a equipe de mudança, que faz uma verificação inicial e depois envia ao cliente para que ele faça testes.

- **Gerar requisição de construção e teste para a mudança na estrutura de replicação e *cluster***

Responsável: equipe de mudança.

O cliente solicita uma mudança na estrutura de replicação e *cluster*, com base nesta solicitação, é gerada uma requisição de construção e testes pela equipe de mudança que a envia à equipe de especialistas.

- **Enviar requisição de construção e teste para a fila de requisições do especialista**

Responsável: equipe de mudança.

Tendo sido gerada uma requisição de construção e teste, esta é enviada para a fila de requisições do especialista que a executará.

- **Receber requisição de construção e testes preenchidos**

Responsável: equipe de mudança.

O especialista recebe a requisição de construção e teste, verifica os impactos que a mudança poderá causar na estrutura, determina os horários em que ela deve ser feita, cria um plano de contingência e detalha a execução da mudança. Finalizado tudo, envia para a equipe de mudança.

- **Revisar requisição de construção e testes**

Responsável: equipe de mudança.

Ao receber a requisição de construção e teste do especialista, devidamente preenchida, a equipe revisa a requisição e a envia para a validação pelo cliente.

- **Enviar requisição de construção e testes para aprovação do cliente**

Responsável: equipe de mudança.

A equipe de mudança, recebendo do especialista a requisição de construção e teste, faz verificações iniciais e a envia para o cliente, para a sua validação.

- **Receber requisição de implantação executada e finalizada pelo especialista**

Responsável: equipe de mudança.

Após a validação da construção e testes pelo cliente, é gerada uma requisição de implantação da mudança na estrutura de replicação e cluster e enviada para a fila do especialista, pela equipe de mudança.

- **Receber aprovação da requisição de implantação do cliente**

Responsável: equipe de mudança.

O especialista envia a requisição de implantação finalizada com sucesso para a equipe de mudança, esta por sua vez a envia para o cliente para que ele faça testes de validação, e retorna para a equipe de mudança aceitando ou contestando a implantação.

- **Receber requisição de construção e testes validado pelo cliente**

Responsável: equipe de mudança.

O especialista preenche a requisição de construção e teste, documentando nela os impactos que a mudança poderá causar na estrutura, de replicação ou na estrutura do *cluster*, determina os horários em que ela deve ser feita, cria um plano de contingência e detalha a execução da mudança e envia para a equipe de mudança, que por sua vez faz verificações e envia para o cliente. O cliente aprova ou não a construção e testes e envia de volta para a equipe de mudança, com a aprovação ou colocando questionamentos.

- **Gerar requisição de implantação da mudança na estrutura de replicação e *cluster***

Responsável: equipe de mudança.

Tendo sido aprovada a construção e teste pelo cliente, é gerada uma requisição de implantação da mudança pela equipe de mudança e, posteriormente, enviada ao especialista em replicação ou cluster para a execução da mudança.

- **Enviar requisição de implantação para o especialista**

Responsável: equipe de mudança.

Gerada a requisição de implantação, ela é enviada para o especialista para que ele a execute.

- **Utilizar sistemas que utilizam a estrutura de replicação e *cluster***

Responsável: cliente.

O cliente utiliza os inúmeros sistemas que utilizam a estrutura de replicação e cluster, seja nos servidores centralizados ou nas réplicas. O cliente identifica as necessidades de mudança, as solicita e valida as mudanças feitas.

- **Aprovar requisições de construção e testes feitas pelos especialistas em replicação e *cluster***

Responsável: cliente.

O cliente recebe as requisições de construção e teste feitas pelos especialistas e as aprova.

- **Solicitar mudanças na estrutura de replicação e *cluster***

Ator: cliente.

O cliente identifica uma necessidade de mudança na estrutura de replicação e *cluster*, em função de mudanças ou atualizações de seus sistemas e as solicita.

- **Informar problemas na replicação e cluster à equipe de monitoramento**

Ator: cliente.

O cliente identifica um problema na replicação e *cluster* que não é detectável pelo sistema de monitoramento e informa à equipe de monitoramento.

- **Validar incidentes resolvidos**

Responsável: cliente.

Os incidentes são resolvidos pelos especialistas em replicação e *cluster*, mas são validados pelos clientes. No caso, havendo problemas, o monitoramento irá identificar os problemas e novos incidentes serão gerados e enviados novamente para os especialistas para que sejam revistos.

> “Os incidentes são resolvidos pelos especialistas em replicação e *cluster*...”

4.2. Ferramentas de monitoramento no MySQL

Foi discutido, anteriormente, todos os processos necessários para a gestão de incidentes em uma estrutura que utilize as tecnologias de replicação e também de *cluster*. No caso, verificamos que o processo de monitoramento é de fundamental importância, pois tudo se inicia na detecção de falhas, e o acionamento das equipes de suporte para a resolução do problema, caso isto não aconteça, toda a estrutura ficará comprometida podendo inclusive vir a parar de funcionar.

Via de regra, os sistemas operacionais disponibilizam suas ferramentas para que o monitoramento seja feito. No caso do **SGBD MySQL**, existem várias ferramentas de monitoramento. Selecionamos três delas, que são as mais utilizadas em grandes estruturas do MySQL, para estudar um pouco as suas características. São ferramentas pagas, mas que possuem versão de avaliação que podem ser baixadas nos sites correspondentes. Estas ferramentas são: Monyog, Datadog e Navicat.

- **Ferramenta Monyog**

É uma ferramenta para servidores de banco de dados MySQL, disponibilizando funcionalidades para **monitoramento e assessoria** aos administradores de banco de dados. Atualmente, em grandes instalações de banco de dados com um grande número de servidores torna a tarefa de monitoramento um desafio. A ferramenta executa um serviço que tem a função de fazer **consultas** aos servidores de banco de dados MySQL, instalados em diversas plataformas e que estão configurados na ferramenta. Este serviço envia consultas ao servidores sobre o estado atual dos bancos de dados, gera arquivos de logs que gravam informações de históricos, o Monyog recupera estas informações, as organiza, faz cálculos estatísticos e exibe os resultados do desempenho do servidor MySQL, permitindo a prevenção de problemas ou falhas que estejam ocorrendo e que venham a prejudicar o funcionamento normal dos servidores, permitindo que se tenha uma visão única e consolidada do estados de todos os servidores MySQL, tratando dados de disponibilidade, gerando alertas de desempenho e também relativos à segurança dos servidores através da gravação de dados de vulnerabilidades que estão sendo monitorados.

A ferramenta Monyog se propõe a disponibilizar **recursos** para:

- Mostrar aos administradores de banco de dados, com antecedência, que recursos dos servidores necessitam ser expandidos;
- Permitir o planejamento de atualização ou substituição de hardware;
- Monitorar o uso de CPU memória e disco;
- Oferece vários recursos para localizar problemas de execução de consultas, identificando as consultas lentas, mostrando quais as que precisam ser otimizadas;
- Permite a programação de alertas a serem enviados aos administradores de banco de dados via e-mail ou SNMP;
- Fornece recursos para a monitoração de estrutura de replicação.

> “A ferramenta executa um serviço que tem a função de fazer consultas aos servidores de banco de dados MySQL...”

➢ **Vantagens**

Uma grande vantagem desta ferramenta é que ela não precisa da instalação de **agentes de monitoramento extras** nos servidores de banco de dados que serão monitorados. É comum neste tipo de ferramentas ser necessário, em cada um dos servidores, a instalação de agentes de software que serão responsáveis pela coleta dos dados de monitoramento. No caso do Monyog, ele utiliza uma **API** (*application program interface*) desenvolvida em linguagem C que interage diretamente com o MySQL, facilitando em muito a implantação desta solução, sem a necessidade de instalações extras nos servidores monitorados.

O Monyog utiliza a linguagem **Java script** que faz o tratamento de todos os seus objetos e ainda permite a personalização de acordo com a necessidade do sistema de monitoramento. Pode ser instalado tanto em ambiente Linux quanto em ambiente Windows e possui entre outras funcionalidades gatilhos que disparam de acordo com a programação de eventos e envia notificações via e-mail.

➢ **Desvantagens**

A principal desvantagem do Monyog é o seu **custo**, e associado a isto, o seu **suporte** é considerado fraco em função da escassez de documentação na internet, seja para a para a plataforma Linux ou para a plataforma Windows. Isto faz com que aumente o grau de dificuldade na hora da configuração da ferramenta, pois havendo problemas fica difícil de resolver em função da falta de documentação.

Outra desvantagem é a necessidade de inclusão de **chaves privadas** do tipo SSH para cada um dos servidores que serão monitorados.

A cópia de demonstração pode ser baixada no endereço: https://www.webyog.com/product/monyog.

• **Ferramenta Datadog**

O Datadog possui alguns recursos que podem realizar **análises avançadas** e esta característica tem feito bastante sucesso entre seus usuários. Ele também permite que sejam feitas **personalizações** de sua interface através da utilização de uma interface de programação de aplicativos (API - *application program interface*). Ele não é considerado a melhor ferramenta de monitoramento para MySQL do mercado, mas tem sido bastante elogiado por seus usuários.

➢ **Vantagens**

Sem dúvida alguma a principal vantagem desta ferramenta é o **suporte**, ela tem uma rica documentação na internet tornando a sua instalação e configuração bastante confortável. Outra grande vantagem é que ela permite a criação de **métricas** e a geração de **gráficos** com as informações de monitoramento com um certo grau de facilidade.

Podem ser criados alarmes ou notificações personalizadas que quando ultrapassam métricas definidas estes são disparados, informando aos usuários as ocorrências nos servidores de banco de dados que estão sendo monitorados.

➢ **Desvantagens**

Um dos pontos fortes do Datadog é a sua interface com o usuário, no entanto, dependendo do número dos servidores a serem monitorados, os **painéis de demonstração** podem ficar técnicos demais e de difícil compreensão. Outra situação que pode confundir o usuário é quando há a necessidade de monitorar **muitas aplicações**, que podem deixar os painéis de informações muito poluídos, dificultando encontrar a informação que se deseja.

O preço não é excessivo, mas também não se pode considerar que tem um preço baixo e a versão gratuita tem poucas funcionalidades.

A cópia de demonstração pode ser baixada no endereço: https://www.datadoghq.com/.

- **Ferramenta Navicat**

Esta ferramenta foi projetada para atender amplamente ao **mercado** de monitoramento de banco de dados, sendo pensada para atender a empresas dos mais diversos tamanhos. Está disponível para macOS, Linux e Windows, tem uma excelente **usabilidade**, e é de fácil uso e configuração.

Ela é uma das ferramentas de monitoramento de banco de dados mais antigas do mercado, inclusive tendo ganhado um prêmio em 2009, por permitir a conexão simultaneamente entre SGBDs, não só MySQL, mas também o PostgreSQL e Oracle, permitindo a migração de dados entre eles.

➢ **Vantagens**

A sua facilidade de **uso** e **configuração** é a maior de suas vantagens, permitindo facilmente criar ou adicionar conexões em um único local sem dificuldades. Como as informações de *login* são configuradas apenas uma vez, o processo de conexão aos bancos de dados fica bastante facilitado.

O Navicat também possui **ferramentas integradas** onde podemos escrever consultas em linguagem SQL para obter os dados necessários ao monitoramento.

Possui um poderoso **assistente de exportação** de dados, o que facilita muito o tratamento de dados, principalmente para quem utiliza a ferramenta Excel.

A **capacidade de transferir dados** de um banco de dados para outro, inclusive de SGBDs diferentes, e mesmo de servidores diferentes, pode ser um aliado valioso em um processo de migração de dados. Ele tem uma funcionalidade adicional de permitir que se desenvolvam tarefas e estas tenham a sua execução agendada em determinados dias e horários.

O Navicat também é compatível com **bancos de dados em nuvem**, como Aurora, Amazon RDS, Microsoft Azure, Redshift, Alibaba e Google Cloud, dentre outros.

➢ **Desvantagens**

Em **grandes consultas**, com um certo grau de complexidade, o Navicat pode responder de forma lenta em relação às demais ferramentas testadas.

Até o momento, ainda não existe funcionalidade que permita criar um ***backup*** de um banco de dados na nuvem com um único clique, como no Google Drive ou Dropbox. O Navicat não possui ferramentas que possam ser utilizadas nos **dispositivos móveis**.

A cópia de demonstração pode ser baixada no endereço: https://www.navicat.com/en/download/navicat-for-mysql.

> “...o processo de conexão aos bancos de dados fica bastante facilitado.”

5. Técnicas e dicas para alta performance de banco de dados

- **Evitar o uso de SELECT ***

A utilização da cláusula SELECT não explicitando os nomes das colunas que se deseja verificar, mas ao invés disto utilizar o **asterisco** (*) é muito útil quando se está desenvolvendo as **consultas**, como uma forma de ganhar tempo com a digitação, no entanto, em produção, pode provocar problemas de desempenho. Vamos supor que na nossa tabela tbvendedor, em uma consulta de relatório, tenhamos escrito a consulta "***select * from tbvendedor***". Da forma que se encontra atualmente, não teremos problemas, a tabela é pequena e os campos são do tipo inteiro e varchar(), mas com o passar do tempo, resolvemos colocar mais campos para guardar a foto do vendedor e sua documentação, ou seja, criamos campos do tipo BLOB, que é utilizado para armazenar os dados binários como imagens, arquivos PDF e Word, e vídeos. A busca deste tipo de campo exige mais recursos de processamento e, consequentemente, mais tempo de retorno. Então, uma consulta do tipo "***select * from tbvendedor***", poderia levar o servidor a um alto consumo de recursos de memória, provocando uma parada nas demais consultas que estão sendo executadas no momento.

- **Criar índices para os campos de pesquisa**

Já vimos que a **indexação** é um excelente recurso para a otimização das consultas, desde que utilizado com **planejamento**, pois o exagero na utilização de **índices** pode prejudicar o desempenho do banco ao invés de melhorá-lo. Portanto, tomando como base a tbcliente, caso tenhamos um índice no campo código e fizermos uma busca pelo campo nome do cliente, teremos uma busca registro a registro, como já discutimos antes, ou seja, uma busca que levará um tempo muito grande. Lembrando que os recursos de computação são sempre compartilhados, o consumo de recursos de memória ou CPU por uma determinada consulta, com certeza irá prejudicar o desempenho das demais. Portanto, devemos lembrar sempre de planejar os índices, os construindo como base nas colunas que farão parte das pesquisas. Outro ponto a lembrar são as colunas que fazem parte da cláusula WHERE, que também, dependendo da complexidade desta cláusula, compõe os índices de busca para evitar uma degradação no tempo destas consultas.

- **Tratamento dos dados na junção de tabelas**

Como já vimos anteriormente, em bancos de dados relacionais, é uma necessidade a utilização da **técnica de junção entre tabelas**, a razão já foi explicada em unidades anteriores. Em primeiro lugar, deveremos ter cuidado na utilização dos tipos de dados que farão parte da **cláusula ON** destas junções (cláusula JOIN), vejamos o nosso exemplo de junção no comando SELECT que faz a junção entre as tabelas tbcliente, tbvendedor, tbproduto, tbloja e tbdepartamento. Todas as colunas que estão na cláusula ON do nosso comando são do mesmo tipo de dado, ou seja, são do tipo inteiro. Esta é a prática correta, pois a colocação de colunas de tipos diferentes nesta cláusula, mesmo tipos de dados equivalentes como ***char*** e ***varchar***, podem ter um efeito negativo no tempo de retorno das consultas. É totalmente condenável a utilização de tipos de dados diferentes que tenha que ser convertidos, de forma implícita ou explícita dentro da cláusula ON.

```
select Venda_Codigo,Venda_Data,Venda_Quantidade,tbcliente.
Cliente_Nome,

tbvendedor.Vendedor_Nome,tbproduto.Produto_Nome,tbloja.Loja_
Nome,tbdepartamento.Departamento_Nome

from tbvenda

LEFT JOIN tbcliente

on tbvenda.Venda_cliente=tbcliente.Cliente_Codigo

LEFT JOIN tbvendedor

on tbvenda.Venda_Vendedor=tbvendedor.Vendedor_Matricula

LEFT JOIN tbproduto

n tbvenda.Venda_produto=tbproduto.Produto_Codigo

LEFT JOIN tbloja

on tbvenda.Venda_Loja=tbloja.Loja_Codigo

LEFT JOIN tbdepartamento

on tbvenda.Venda_Departamento=tbdepartamento.Departamento_
Codigo
```

- **Utilização de chaves primárias em uma coluna independente**

Em muitos projetos de banco de dados, poderemos ter tabelas que tenham colunas candidatas a chaves primárias. Estas colunas podem ser uma única, caracterizando uma **chave primária simples**, ou mais de uma coluna formando uma **chave primária composta**. No entanto, é fortemente recomendável que se crie uma coluna independente para a chave primária, do tipo INTEIRO. No nosso banco de dados de exemplo, criamos em todas as tabelas uma coluna independente nomeada como "**código**", do tipo inteiro e com a propriedade AUTO INCREMENTO ativada. Isto, como já vimos anteriormente, muito facilita na inserção dos dados, pois o controle de numeração desta coluna fica por conta do próprio SGBD.

- **Cuidados na utilização dos gatilhos (*triggers*)**

Os gatilhos são objetos úteis de banco de dados, mas o seu uso deve ser bem planejado. Principalmente os **gatilhos de exclusão** que podem causar uma maior degradação no banco de dados. No caso de exclusões, a melhor solução é a **marcação dos registros** a serem excluídos e o seu posterior tratamento em rotinas de manutenção. O mais indicado é sempre que nada seja excluído do banco de dados, havendo apenas uma marcação dos registros que não serão mais utilizados e, posteriormente, rotinas de manutenção os transferirem para outras tabelas de histórico. O desenvolvimento dos gatilhos deve ser feito com bastante cuidado para evitar os "***loops* infinitos**", isto acontece quando os gatilhos ficam sendo executados recursivamente, ou seja, uma ação dispara o gatilho que é novamente disparado, sem uma finalização, causando uma grande degradação no banco de dados.

PRATIQUE

1. Descreva o modelo de replicação mestre-escravo.

2. Descreva o modelo de replicação multimestre.

3. O que é um cluster e quais os tipos mais utilizados?

4. Quais os tipos básicos de replicação do MySQL?

5. Qual a razão para a otimização dos processos de gestão das equipes de tecnologia da informação ?

6. Quais as principais ferramentas de monitoramento do MySQL?

*7. Qual o problema de se utilizar a cláusula SELECT * em uma consulta em produção?*

8. *Por que é importante criar índices nos campos que irão compor as cláusulas de pesquisa de uma consulta?*

9. *O que é importante levar em conta quando projetamos uma consulta que irá utilizar junção de tabelas?*

10. *Por que devo utilizar uma coluna independente como chave primária? Qual a principal vantagem desta prática?*

Relembre

Nesta última unidade, predominantemente teórica, estudamos conceitos avançados das técnicas de replicação e de cluster, técnicas estas que se bem planejadas e aplicadas permitirão a alta disponibilidade e o alto desempenho dos bancos de dados.

Discutimos também a questão do monitoramento, porque de nada adiantaria implantarmos uma estrutura com estas características e não a monitorarmos, porque além da necessidade da identificação de falhas, com a utilização, toda e qualquer estrutura de tecnologia precisa ser modificada. Neste ponto, estudamos processos baseados em ITIL para que a monitoração seja utilizada para a melhoria da estrutura.

Por fim, discutimos algumas práticas que devem ser evitadas quando estamos desenvolvendo um banco de dados, pois estas podem ser bastante prejudiciais para a alta performance de uma estrutura de banco de dados.

Referências

BELL, Charles. **Introducing InnoDB Cluster**. Warsaw Virginia - USA, 2018.

MEHTA, Chintan *et al*. **Database Replication**. M. Tamer Ozsu. University of Wanterloo, 2010.

SCHWARTZ, Baron *et al*. **O'Reilly Hight Performance MySQL**, 2012.

VANIER, Eric; BIRJU, Shah; MALEPATI, Teja. **O'Reilly Advanced MySQL 8**. Birmingham Mumbai, mar., 2019.

XU, Rui C.; WUNSCH II, Donald. **Clustering**. Wiley IEEE Press, 2008.

Anotações

Anotações

Rua Coletor Antônio Gadelha, Nº 621
Messejana, Fortaleza – CE
CEP: 60871-170, Brasil
Telefone (85) 3033.5199