Administração do SGBD PostgreSQL

Camila Freitas Sarmento

Unidade 4

Unidade 4| Introdução

O *backup* e a restauração são tarefas essenciais na indústria, por isso, os *backups* periódicos são tão importantes para uma pronta recuperação devido a falhas e catástrofes.

Unidade 4| Objetivos

1. Ajustar a performance do PostgreSQL por meio do processo de tuning.
2. Configurar as preferências e parâmetros de ajustes do PostgreSQL.
3. Realizar cópias de segurança de banco de dados e recuperar essas cópias quando necessário, salvaguardando a segurança e integridade dos dados no PostgreSQL.
4. Efetuar o procedimento de replicação de dados que já vem disponível nativamente no PostgreSQL.

1. Performance *tuning*

Em uma base de dados PostgreSQL, os valores de configuração padrão não são apropriados em um ambiente de produção, pois tais valores geralmente são subdimensionados. Ainda assim, em modo produção, o impacto da informação da observação pode não ser preciso o suficiente para aumentar o desempenho.

Fonte: Pixabay

Um dos princípios de ajuste de desempenho é que, em geral, não há como supor qual o próximo gargalo até a remoção do atual. E tentar conjeturar pode ser um trabalho sem eficácia. Diante desse contexto, torna-se responsabilidade do administrador do banco de dados realizar os ajustes do SGBD PostgreSQL necessários conforme a carga de trabalho e, ainda, realizar a correta configuração do banco de dados é apenas a primeira etapa no ajuste de desempenho.

Dessa forma, o primeiro passo para o aprendizado de como realizar a configuração de *tuning* no PostgreSQL é compreender o ciclo de vida de uma consulta, cujas etapas são:

Transmissão de string de consulta para o *backend* do banco de dados: esta é a etapa do comando SQL digitado por meio do aplicativo e envio para o *backend*.

Análise de *string* de consulta: nesse passo do ciclo de vida da consulta, quando a *string* está armazenada no servidor PostgreSQL, ela é analisada em *tokens*.

Planejamento de consulta para otimizar a recuperação de dados: esta etapa é na qual o servidor inicializa o trabalho e verifica se a *query* já está pronta e se a biblioteca do cliente suporta esse recurso e analisa o SQL para determinar qual a abordagem mais eficiente na recuperação dos dados.

Recuperação de dados do *hardware*: após a tomada de decisão do PostgreSQL sobre a melhor abordagem na recuperação dos dados (tais como: uso de índice, *hash join* etc.), então irá realizar a obtenção, cuja etapa depende da configuração de *hardware*.

Transmissão de resultados para o cliente: nesta última etapa do ciclo de vida da consulta ocorre a transmissão dos resultados ao cliente. Nesta etapa, não há opções de ajustes para o administrador do banco de dados.

Performane de *tuning* para leitura

O PostgreSQL analisa tabelas, coleta estatísticas dessas tabelas e constrói histogramas usando auto-vacuuming. Auto-vacuuming é normalmente usada para recuperar espaços em disco, atualizar estatísticas de tabela e executar outras tarefas de manutenção, por exemplo, evitar o retorno de ID da transação.

Fonte: Pixabay

Dessa forma, as estatísticas da tabela realizadas pelo PostgreSQL permitem que seja escolhido um plano de execução com o menor custo possível. O custo é calculado levando em consideração as entradas e saídas, além do custo da CPU. Além disso, o PostgreSQL permite que os usuários.

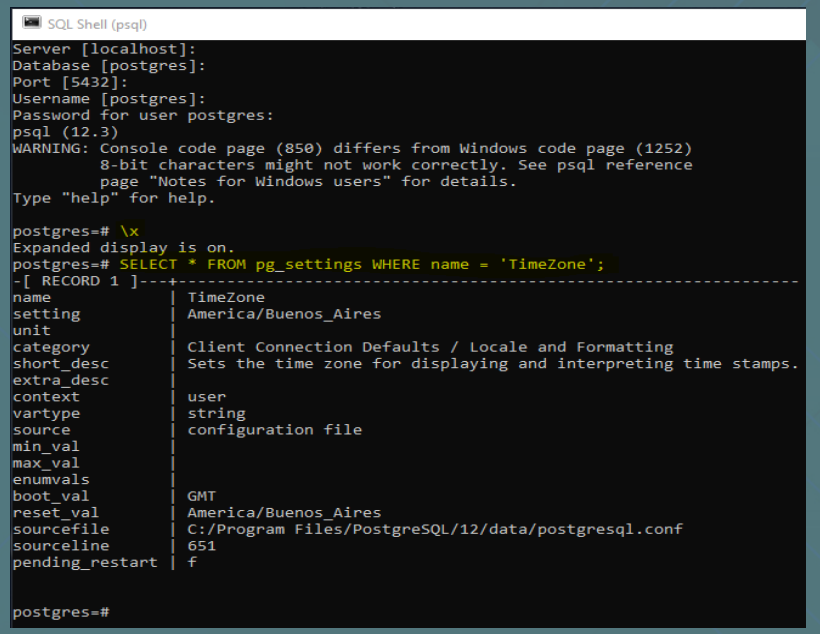
De acordo com os autores Juba e Volkov (2019, p. 410), existem algumas dicas que permitem com que o administrador do banco de dados possa decidir se o plano de execução é bom ou não. Essas dicas são as seguintes: o número estimado da linha em comparação com o número de linhas reais, operação de ordenação em memória ou em disco, *buffer cache*.

Em geral, existem vários problemas que podem levar a um mau desempenho: estatísticas incorretas, recuperação de dados desnecessária e manipulação de dados desnecessária.

A visualização *pg_stat_activity* deve sempre ser verificada primeiro porque possibilitará um *overview* do sistema, apesar de o monitoramento gráfico fornecer uma primeira impressão do sistema, mas tudo se resume às consultas que estão sendo executadas no servidor.

2. Configuração do PostgreSQL

O arquivo postgresql.conf controla as configurações da manutenção do servidor PostgreSQL e, por meio dele, é possível substituir muitas configurações no banco de dados, função, sessão e até mesmo níveis de função. Você encontrará muitos detalhes sobre como ajustar seu servidor modificando as configurações no arquivo e o seu servidor PostgreSQL.



Fonte: Elaborado pela autora (2021).

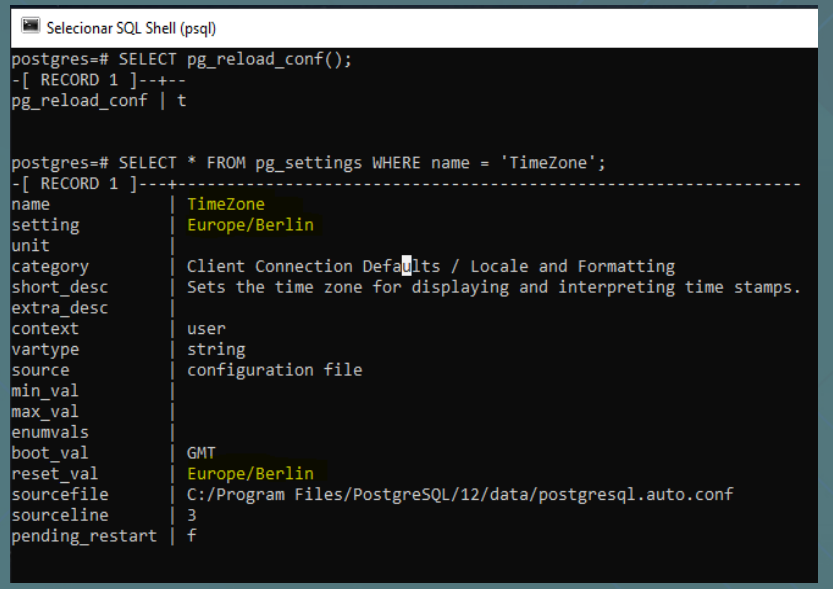
Um dos métodos que possibilita ao administrador realizar a leitura das configurações atuais do servidor PostgreSQL sem abrir o arquivo *postgresql.conf* é consultar a visão chamada *pg_settings*.

A primeira coisa que um administrador precisa aprender é como descobrir de onde vêm os parâmetros de configuração.

As configurações de superusuário podem ser alteradas apenas por um superusuário e serão aplicadas a todos os usuários que se conectarem após uma recarga. Os usuários não podem substituir individualmente a configuração. Agora que você já sabe selecionar o fuso horário, vamos supor que você precise trocar o fuso horário do seu servidor PostgreSQL.

Como trocar o fuso horário no Postgresql

Veja que temos um TimeZone definido como “America/Buenos Aires”. Então, suponhamos que você deseje trocar para “Europe/Berlin”. Para isso, você precisa executar o seguinte comando: postgres=# ALTER SYSTEM SET timezone to ‘Europe/Berlin’.



Fonte: Elaborado pela autora (2021).

Caso você execute novamente o comando para verificar o fuso horário atual, verá que o TimeZone definido está o mesmo. Então, para que as alterações sejam aplicadas, é fundamental reiniciar o serviço do PostgreSQL: postgres=# SELECT pg_reload_conf();

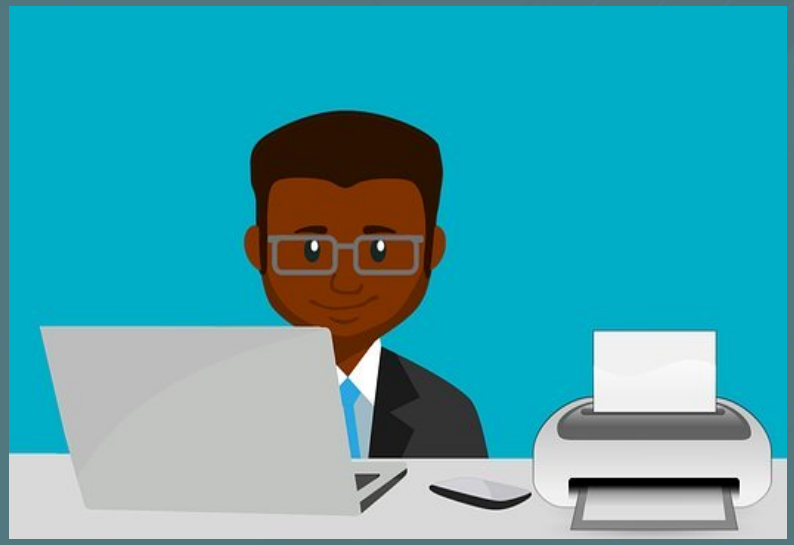
Após a execução, podemos listar o fuso horário e verificar a troca.

O comando ALTER SYSTEM do PostgreSQL e *postgresql.auto.conf* fornecem uma maneira conveniente de alterar a configuração de todo o *cluster* de banco de dados sem que o administrador precise editar o arquivo *postgresql.conf* manualmente.

No PostgreSQL é possível usar o *postgresql.auto.conf* para sobrescrever os parâmetros de configuração em postgresql.conf. Dessa forma, o *postgresql.auto.conf* possui uma prioridade mais alta em relação ao postgresql.conf.

3. Backup e recuperação

PostgreSQL suporta *backups* físicos e lógicos. O *backup* físico é executado copiando os arquivos do banco de dados e os arquivos WAL acumulados durante a cópia dos arquivos. Já o *backup* lógico é usado para fazer a cópia de segurança do banco de dados na forma de instruções SQL.



Fonte: Pixabay

Frequentemente, um banco de dados pode conter vários objetos não utilizados ou dados muito antigos. Limpar esses objetos ajuda os administradores a fazerem *backup* de imagens mais rapidamente. Do ponto de vista do desenvolvimento, os objetos não utilizados são semelhantes ao ruído silencioso porque eles afetam o processo de refatoração.

As estatísticas da tabela (tais como: o número de linhas ativas, varreduras de índice e varreduras sequenciais) podem ajudar a identificar tabelas vazias e não utilizadas. Para isso, a tabela *pg_stat_user_tables* fornece essas informações e é possível verificar tabelas vazias por meio do número de *tuplas*: SELECT relname FROM pg_stat_user_tables WHERE n_live_tup= 0;

No caso das colunas vazias ou não utilizadas, para localizar, é preciso verificar o atributo *null_fraction* da tabela *pg_stats*. Caso o *null_fraction* seja igual a 1, então significa que a coluna está completamente vazia.

O PostgreSQL vem com três utilitários para *backup* dentro da pasta *bin*: **pg_dump**, **pg_dumpall** e **pg_basebackup**. Use o *pg_dump* para fazer *backup* de bancos de dados específicos. No caso do *backup* de todos os bancos de dados em texto simples e junto aos globais do servidor, use *pg_dumpall*. Já para o *backup* de disco em nível de sistema de todos os bancos de dados, use o *pg_basebackup*.

Restauração no PostgreSQL

Há duas formas de restaurar os dados do PostgreSQL: (i) usando o psql para restaurar *backups* de texto simples que foram gerados pelos pg_dump e pg_dumpall; e (ii) usando o utilitário pg_restore para restaurar *backups* TAR, comprimidos e do diretório criado pelo pg_dump.

Fonte: Pixabay

De acordo com Obe e Hsu (2017, pg. 79), um *backup* simples de SQL é um arquivo de texto que contém um script SQL robusto. Com o *backup* SQL, você deve executar o *script* inteiro. Você não pode escolher objetos a menos que queira editar o arquivo manualmente.

Um *backup* simples de SQL é o *backup* menos conveniente de se ter, mas é o mais versátil.

Com o *pg_restore*, é possível realizar algumas operações em paralelo para melhorar a velocidade de carregamento em sistemas que são limitados pela velocidade da CPU em vez do rendimento do disco. Para realizar as operações de restauração em paralelo, você pode executar a opção -j para controlar o número de *threads* que serão usados e, com isso, diminuir o tempo de execução do processo.

Além de permitir realizar restaurações seletivas, o *pg_restore* também permite que você possa criar um arquivo de índice de seu arquivo de *backup* para confirmar o que foi feito de fato e, ainda, pode editar esse índice e usar o arquivo revisado para controlar quais objetos devem ser restaurados.

4. Replicação nativa

O PostgreSQL nos fornece métodos para construir e manter uma cópia totalmente *on-line* do banco de dados primário. Além disso, existem utilitários para duplicar tabelas quando não precisamos de uma cópia de todo o banco de dados.

Thomas (2020, p. 332) afirma que a partir da versão 10, o PostgreSQL oferece suporte nativo à replicação lógica usando estruturas de publicação e assinatura.

Fonte: Pixabay

Eles são gerenciados diretamente pelo analisador de sintaxe PostgreSQL como SQL nativo e criam objetos no catálogo do sistema que são gerenciados como todo o resto.

Dessa forma, facilita a criação de listas de tabelas para transmitir aos servidores PostgreSQL destinatários localizados em outro lugar. Isso também significa que as ferramentas PostgreSQL padrão podem interagir com os conjuntos de assinaturas e tabelas de maneira que as extensões não podem reproduzir.

Replicação de *streaming*

Lembre-se de que a replicação de *streaming* requer apenas conexões no nível do banco de dados PostgreSQL entre o mestre e os escravos (standby).

Os mestres podem enviar dados, enquanto os *standbys* (ou escravos) são sempre receptores de dados replicados. Vejamos o passo a passo para a configuração do **servidor master**:

Fonte: Pixabay

Na máquina master, iremos realizar as seguintes configurações: vamos alterar as seguintes configurações em postgresql.conf. Isso pode ser feito usando ALTER SYSTEM set variable = value seguido por SELECT pg_reload_conf (); alternativamente, também podemos abrir o arquivo *postgresql.conf* e realizar os ajustes necessários.

O próximo passo é criar uma conta de superusuário PostgreSQL dedicada e um banco de dados para os dados do repmgr. O próximo passo da configuração é a atribuição das permissões necessárias no arquivo pg_hba.conf para poder realizar a conexão em modo replicação. Ao final do documento *pg_hba.conf*, a seguinte linha deverá ser adicionada ao final do arquivo para que seja concedida a permissão de acesso à replicação. Após isso, deve-se reiniciar o servidor para que as configurações sejam aplicadas.

Depois é só criar o arquivo de configuração do repmgr. Criado o arquivo, adiciona as configurações, a partir de então, o próximo passo será a criação de um caminho de pesquisa do usuário do **repmgr** para incluir o nome de *schema*, agora é só inicializar o servidor master para que cada servidor no *cluster* de replicação tenha sua própria gravação, cuja atualização ocorrerá quando o *status* ou a função forem alterados.