CENTRO UNIVERSITÁRIO UNIVATES
CURSO DE ENGENHARIA DA COMPUTAÇÃO

FERRAMENTA DE COLETA E ANÁLISE DE ESTATÍSTICAS PARA SISTEMA GERENCIADOR DE BANCO DE DADOS POSTGRESQL

Fabiano Tomasini

Lajeado, novembro de 2015.

Fabiano Tomasini

FERRAMENTA DE COLETA E ANÁLISE DE ESTATÍSTICAS PARA SISTEMA GERENCIADOR DE BANCO DE DADOS POSTGRESQL

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia da Computação do Centro Universitário Univates, para obtenção do título de Bacharel em Engenharia da Computação.

Orientador: Prof. Evandro Franzen

Lajeado, novembro de 2015.

RESUMO

O crescimento das empresas e organizações tem evidenciado a importância da informação para a tomada de decisões, sendo que o volume, cada vez maior de dados, têm exigido atenção com a velocidade com que as informações são obtidas e gera preocupações quanto a forma com que os dados são armazenados. É fundamental garantir a integridade das informações e um bom desempenho em seu acesso. Algumas dessas preocupações podem ser amenizadas, através de algumas ações que podem ser tomadas em relação ao SGBD (Sistema de Gerenciamento de Banco de Dados) escolhido. O trabalho em questão tem como objetivo disponibilizar uma ferramenta que possibilite identificar, de forma visual, possíveis problemas de desempenho em relação ao SGBD. Através da ferramenta desenvolvida são coletadas estatísticas do banco de dados, permitindo detectar impasses que estejam relacionados a administração do banco de dados, podendo assim, tornar visível a necessidade de criação de índices e alterações em configurações do sistema utilizado. A ferramenta facilita a otimização e o monitoramento do sistema gerenciador de banco de dados PostgreSQL.

Palavras-chave: PostgreSQL, banco de dados, desempenho e otimização.

ABSTRACT

The growth of enterprises and organizations has shown the importance of information for decision making, the increasing amount of data has required attention to the speed with which information is obtained and concerns about the way the data is stored. It is essential to ensure the integrity of information and a good performance in access. Some of these concerns can be alleviated with some actions that can be taken in relation to the DBMS (Database Management System) chosen. The work in question is intended to provide a tool that enables to identify, visually, potential performance issues in relation to the DBMS. Through the tool developed database statistics are collected which can detect deadlocks that are related to database administration and can thus make visible the need to create indexes and changes in system settings used. The tool facilitates the optimization and monitoring of Database Management System PostgreSQL.

key words: PostgreSQL, database, performance and optimization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

HTML:	HyperText Markup Language
HTTP:	Hypertext Transfer Protocol
PHP:	PHP Hypertext Preprocessor
SGBD:	Sistema de Gerenciamento de Banco de Dados
SQL:	Structured Query Language
BSD:	Berkeley Software Distribution
DBA:	Database Adminstrator
SaaS:	Software as a Service
GNU:	General Public License
SOAP:	Simple Object Access Protocol

SUMÁRIO

1 INTRODUÇÃO

Com o desenvolvimento tecnológico os softwares exigem mais dos computadores, ou seja, maior poder de processamento, de modo que informações importantes sejam obtidas de maneira fácil e rápida. Sendo essas informações a chave para uma boa tomada de decisão, podemos considerar que a mesma garante a sobrevivência de uma organização no mercado global, motivando diversos estudos na área de banco de dados (ROB; CORONEL, 2011).

Segundo Silberschatz, Korth e Sudarshan (2012) um banco de dados é considerado uma coleção de dados relacionados, que são fornecidos ao usuário final de maneira conveniente, eficiente e segura. Visto que a manipulação de informações é tão importante para a gestão de uma organização, cientistas da computação têm desenvolvido um grande conjunto de conceitos e técnicas de gerenciamento de dados visando a otimização da obtenção destes dados.

Para garantir o armazenamento correto e manter a integridade dos dados é indispensável monitorar o uso do sistema gerenciador de banco de dados, verificando por exemplo a ocorrências de impasses (*deadlocks*) que podem afetar a disponibilidade do software. Além disso, é fundamental analisar a performance das consultas realizadas e isso requer o conhecimento das estatísticas do banco de dados, tais como, número de consultas por tabela, que utilizaram ou não utilizaram índices, e percentual de consultas por tabela, que utilizaram ou não utilizaram cache.

O presente trabalho apresenta o desenvolvimento de uma ferramenta que contribui para manutenção de grandes bancos de dados, com ênfase especial na performance das aplicações que dependem desse serviço. Para obter o resultado esperado, foram abordados gargalos de performance relacionados à administração de banco de dados através de

estatísticas extraídas do SGBD (Sistema de Gerenciamento de Banco de Dados). As informações coletadas servem para medir a performance do banco e alertar o DBA (*Database administrator*) de possíveis pontos de melhorias e problemas.

A ferramenta conta com um módulo coletor de estatísticas que deve ter acesso ao banco de dados a ser monitorado. Nesse trabalho, foi desenvolvido o módulo coletor para o SGBD PostgreSQL, porém a arquitetura desenvolvida permite que seja implementado o monitoramento de outros SGBDs em trabalhos futuros, através de coletores específicos.

1.1 Problema

Com base na importância do monitoramento do SGBD para manter a disponibilidade do serviço, e da análise de estatísticas para garantir a otimização de consultas, é evidenciada a necessidade de ter ferramentas que auxiliam na execução desse trabalho.

Atualmente, existem várias ferramentas que tem como objetivo gerenciar tabelas, acessos, porém, quando se deseja monitorar estatísticas do banco de dados, são encontrados poucos softwares que fornecem essas informações de maneira gráfica. Algumas ferramentas atingem o objetivo, mas acabam sendo descartadas por serem pagas, descontinuadas, não suportarem mais de um SGBD ou simplesmente não proporcionam o que se deseja de maneira clara.

Tendo como base essa realidade, o presente trabalho, proporciona ao DBA (*Database Adminstrator*) uma ferramenta que possibilita o monitoramento e análise de estatísticas de maneira gráfica e de fácil manuseio.

1.2 Objetivos

O objetivo geral do presente trabalho foi disponibilizar uma ferramenta para auxiliar na administração, monitoramento e otimização de um banco de dados que utilize qualquer SGBD e que tenha um grande volume de informações armazenadas. Inicialmente, o software tem um enfoque maior no sistema PostgreSQL, porém, possibilitará que seja implementado o monitoramento de outros SGBDs em trabalhos futuros.

1.2.1 Objetivos específicos

Para atingir o objetivo principal definido, os seguintes objetivos específicos também foram cumpridos:

a) Compreender os princípios de coleta e utilização de estatísticas de um banco de dados;

b) Permitir a coleta de estatísticas através de rotinas automatizadas para vários SGBDs;

c) Contribuir para uma melhor administração de dados através da ferramenta proposta.

1.3 Organização do trabalho

A fim de melhorar a compreensão do presente trabalho, os capítulos serão apresentados na seguinte ordem.

O capítulo 2 explica os conceitos que foram necessários para o desenvolvimento da proposta, tais como, conceitos sobre bancos de dados relacionais, administração de bancos de dados, consultas, concorrência, transação e etc.

No capítulo 3 foi realizado um estudo comparativo de ferramentas já existentes para monitoramento e otimização de SGBDs.

O capítulo 4 apresenta os materiais e métodos utilizados no desenvolvimento da ferramenta, onde foi abordada a metodologia utilizada para a realização do trabalho.

No capítulo 5 é descrito em detalhes a ferramenta desenvolvida e os resultados obtidos com ela.

Por fim, o capítulo 6 apresenta as considerações finais do presente trabalho.

2 REFERENCIAL TEÓRICO

Neste capítulo, são abordados os fundamentos teóricos que foram necessários para o desenvolvimento da ferramenta. Foram realizados estudos em artigos, livros e documentos digitais. Dentre as informações levantadas, destacam-se, conceitos sobre bancos de dados, administração de sistema de gerenciamento de bancos de dados, processamento de consultas e controle de concorrência.

2.1 Banco de dados

Segundo Silberschatz, Korth e Sudarshan (2012), um banco de dados, é uma coleção de dados relacionados, ou seja, sempre que tenho informações que se relacionam e que tratam do mesmo assunto, pode ser considerado um banco de dados. Um banco de dados de uma universidade, por exemplo, poderia conter informações sobre os alunos, professores e suas turmas, e poderiam estar relacionados às matrículas dos alunos nos cursos ministrados pelos professores e ao uso de salas por curso (RAMAKRISHNAN; GEHRKE, 2008).

O banco de dados tem como principal objetivo, armazenar e permitir aos usuários buscar e atualizar informações quando necessário. Essas informações podem ser qualquer dado que tenha algum significado para um usuário ou organização, em outras palavras, as informações são dados necessários para auxiliar no processo geral de tomada de decisões (DATE, 2004).

Na ferramenta desenvolvida bancos de dados foram utilizados como fonte de dados e para o armazenamento informações coletadas, esse conceito é considerado fundamental para o SMBD.

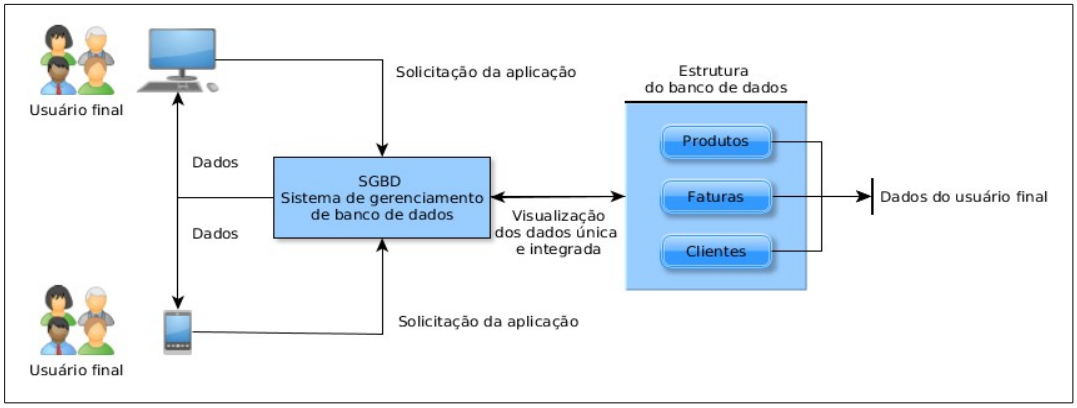
2.2 Sistema de gerenciamento de banco de dados

Segundo Ramakrishnan e Gehrke (2008) um sistema de gerenciamento de banco de dados, ou SGBD, é um software que foi desenvolvido para auxiliar na manipulação e organização de vastos conjuntos de dados. A necessidade de um software com esse fim tem crescido rapidamente, pois antes de surgirem, os dados eram armazenados em arquivos e era necessário escrever códigos específicos na própria aplicação para gerenciá-los. Siberschartz, Korth e Sudarshan (2012) também consideram este sistema como uma coleção de dados inter-relacionados e um compilado de programas para manipular esses dados.

Os sistemas gerenciadores de base dados proporcionam produtividade na manipulação de informações, ou seja, obter de maneira rápida os dados desejados para uma decisão. Utilizando uma linguagem de alto nível, estes sistemas permitem que seus usuários escrevam consultas de maneira simples sem definir detalhes relacionados ao seu processamento, tarefa a qual é atribuída ao próprio SGBD, que irá escolher através de um processo de planejamento e otimização, a forma mais eficaz de obter os dados desejados (DATE, 2004).

Um sistema gerenciador de base de dados serve como um intermediário entre o banco de dados e o usuário final. Em sua estrutura interna os dados são armazenados em um conjunto de arquivos e a única forma de acessar as informações é por meio desse sistema. A figura 1 ilustra que para o usuário final é mostrado uma visualização única e integrada das informações armazenadas. O sistema gerenciador recebe diversos comandos enviados a partir de aplicações e os traduz em uma linguagem compreensiva para atendê-los, ocultando dos usuários boa parte da complexidade interna do banco de dados (ROB; CORONEL, 2011).

Figura 1 - O SGBD gerencia a interação entre o usuário final e o banco de dados.



Fonte: Modificado de Rob, Coronel (2011, p. 7).

2.2.1 Modelo relacional

Segundo Ramakrishnan e Gehrke (2008) um modelo de dados é considerado uma seleção de construtores de alto nível que ocultam detalhes de baixo nível do armazenamento de informações. O SGBD permite que o usuário final defina as informações a serem armazenadas em relação ao modelo de dados. Grande parte dos SGBDs atuais baseia-se no modelo relacional.

O modelo relacional devido sua compreensibilidade é considerado o principal modelo de dados utilizado atualmente, facilitando o trabalho dos desenvolvedores de software, diferentemente dos modelos mais antigos, como o modelo de rede ou o modelo hierárquico (SILBERSCHATZ; KORTH; SUDARSHAN, 2012).

Segundo Siberschartz, Korth e Sudarshan (2012) um banco de dados que utiliza o modelo relacional contém um conjunto de tabelas (conjunto de entidades), onde cada tabela recebe uma denominação única que irá representá-la. Cada tabela contém campos (atributos) que também recebem uma denominação única e um tipo especifico de dados.

2.2.2 Entidades

De acordo com Siberschartz, Korth e Sudarshan (1999) uma entidade é uma “coisa” ou um “objeto” do mundo real que pode ser determinado de forma homogênea em relação a todos os outros objetos. Por exemplo, em uma empresa, cada pessoa é uma entidade. Uma entidade é composta por uma ou mais propriedades, sendo que algumas dessas propriedades devem ser únicas. Uma entidade pode ser concreta, como uma pessoa ou um livro, ou pode ser abstrata como um empréstimo.

2.2.3 Atributos

Os atributos são características descritivas de cada membro de um conjunto de entidades (tabelas). A escolha de um atributo para uma tabela mostra que o banco de dados mantém informações equivalentes de cada entidade, entretanto cada entidade pode ter seu próprio valor em cada atributo (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

2.2.4 Chaves

Segundo Siberschartz, Korth e Sudarshan (1999) uma chave em um banco de dados tem o objetivo de identificar e estabelecer ligações entre entidades. Em um banco de dados relacional existem dois tipos de chave, chave primária e chave estrangeira.

2.2.4.1 Chave primária

A chave primária de uma determinada entidade é representada por uma coluna ou uma combinação de colunas, onde os valores irão diferenciar de forma única uma entidade das demais dentro de uma tabela (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

2.2.4.2 Chave estrangeira

Segundo Siberschartz, Korth e Sudarshan (1999) uma chave estrangeira de uma tabela é um atributo ou um conjunto de atributos que referenciam uma chave primária, ou seja, os atributos de uma chave estrangeira possui o mesmo valor de um atributo de uma chave primária de uma outra tabela. A chave estrangeira permite relacionamento entre os conjuntos de entidades contidas em um banco de dados relacional.

2.2.5 Administração de um banco de dados

Segundo Milani (2008) a administração de um banco de dados é uma tarefa que, dependendo das ferramentas utilizadas e da complexidade dos bancos de dados em questão, pode ser fácil ou extremamente difícil.

De acordo com Siberschartz, Korth e Sudarshan (1999) a tarefa de administração de um banco de dados é executada por uma pessoa denominada DBA (*Database Adminstrator*) que centraliza o controle do sistema. Dentre as funções do DBA podem ser destacadas as seguintes:

a) Definição da estrutura de dados e método de acesso: O DBA é responsável por criar estruturas de dados e métodos de acessos definidos através de um conjunto de instruções, as quais são traduzidas pelo compilador do SGBD;

b) Esquema e modificação na organização física: O DBA é responsável por alterações no esquema do banco de dados por meio de um conjunto de definições que serão utilizadas pelo SGBD gerando modificações nas tabelas apropriadas;

c) Fornecer autorização de acesso ao sistema: O DBA é responsável por regular o acesso de diferentes usuários às diferentes partes do sistema. Os dados referente aos acessos são armazenados em uma estrutura especial do SGBD e é consultada sempre que o acesso a determinado dado for solicitado;

d) Especificação de regra de integridade: O DBA é responsável por definir regras que devem garantir a integridade dos dados armazenados. As regras são tratadas por uma estrutura especial do SGBD e é consultada sempre que uma atualização está em curso no sistema;

e) Definição de esquema: O DBA é responsável pela criação do esquema do banco de dados, escrevendo um conjunto de definições que são transformadas pelo compilador do SBGD em um conjunto de tabelas armazenadas permanentemente no dicionário de dados.

2.2.6 Catálogo de um SGBD

Segundo Elmasri e Ramez (2011) um SGBD é um sistema genérico projetado para atender diversas aplicações de banco de dados, desta maneira, sempre que necessário conhecer a estrutura de um banco de dados específico deve se recorrer ao catálogo, pois é lá que o SGBD armazena os metadados do esquema, tais como informações sobre tabelas, colunas e informações de controle interno.

No catálogo, além de informações sobre a estrutura do banco de dados, são armazenadas também informações estatísticas que são usadas pelo SGBD para otimizar o plano de execução de uma consulta. Além disso, as estatísticas podem ser usadas pelo administrador de banco de dados para identificar problemas de performance, tais como

índices não utilizados e a ocorrência de *deadlocks* (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

Uma consulta convencional retorna informações existentes em tabelas, já uma consulta no catálogo retorna informações sobre os bancos, os objetos dos bancos, os campos de tabelas, seus tipos de dados, seus atributos e etc. Na figura 2 podemos visualizar uma consulta que obtém do catálogo do PostgreSQL informações como o nome e o schema de dados de todas as tabelas de um banco de dados (POSTGRESQL, 2015).

Figura 2 - Consulta no catálogo do PostgreSQL.

```
1      SELECT schemaname AS esquema,
2             tablename AS tabela
3        FROM pg_catalog.pg_tables
4    ORDER BY schemaname,  tablename;
```

Fonte: Elaborado pelo autor.

2.2.7 Processamento de consultas

Segundo Siberschartz, Korth e Sudarshan (1999) o processamento de consultas refere-se ao conjunto de tarefas incluídas na extração de informações de um banco de dados. As tarefas incluem, tradução das consultas em uma linguagem que seja entendida pelo SGBD e que permitem ser usadas no nível físico do sistema de arquivos e uma série de transformações de otimização das consultas.

Antes de qualquer consulta iniciar o processamento, o SGBD precisa transcrever a consulta para uma linguagem interna. A linguagem de banco de dados SQL utilizada na maioria dos SGBDs relacionais, é ideal para a ação humana, porém essa representação não é a ideal para o sistema interno de consultas. A representação interna adequada é baseada na álgebra relacional.

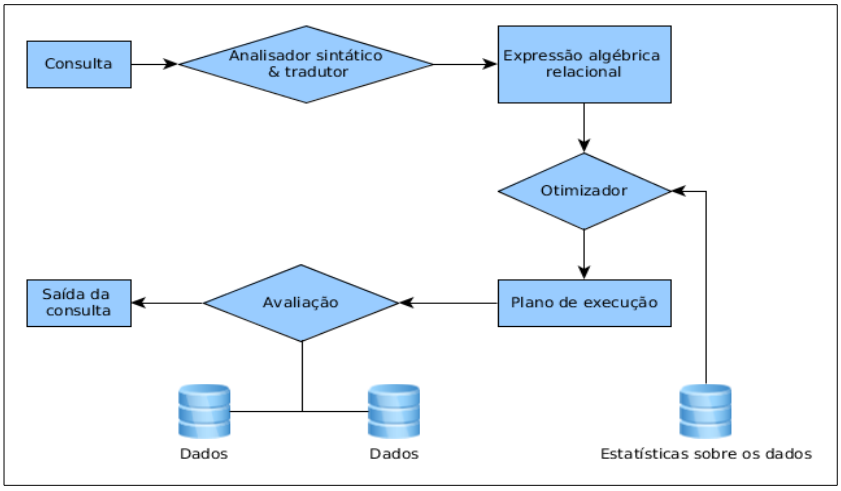
Na Figura 3 podemos visualizar as etapas envolvidas no processamento de uma consulta, onde podemos destacar as seguintes:

a) Analisador sintático e tradutor: Convertem a consulta para interpretação interna do SGBD;

b) Otimizador: Transforma a expressão em um modelo de álgebra relacional;

c) Avaliação: É o mecanismo responsável pela execução da consulta, executar o plano e retorna a resposta.

Figura 3 - Etapas no processamento de consultas no banco de dados.



Fonte: Adaptado de Silberschatz, Korth, Sudarshan(1999).

Segundo Blumm e Fornari (2006) no processamento de consultas o otimizador desempenha o papel principal. Há dois tipos de otimizações:

a) Otimização baseada em regras heurísticas: Estas regras estão inclusas ao SGBD Oracle e produzem bons resultados, porém não há comprovação que garanta sua correção em todas as consultas (BLUMM; FORNARI, 2006);

b) Otimização baseada em estatísticas: Nesta opção o otimizador de consultas aplica algumas fórmulas para calcular o custo (tempo de processamento + tempo de acesso aos dados em disco) de várias alternativas possíveis e opta a apresentar o menor custo estimado. Os dados estatísticos como o número de linhas de uma determinada tabela são mantidos no dicionário de dados do SGBD (BLUMM; FORNARI, 2006). Para estimar os custos de diversas estratégias de execução, o SGBD registra qualquer informação necessárias para as funções de custo, essas informações são acessadas pelo otimizador sempre que necessário (ELMASRI; RAMEZ, 2011).

Segundo Blumm e Fornari (2006) alguns SGBDs utilizam apenas um dos métodos apresentados e outros permitem a escolha do método que apresentar melhores resultados.

2.2.7.1 Álgebra relacional

Segundo Borello e Kneipp (2008) a álgebra relacional é considerada uma linguagem de consulta procedural, onde o usuário especifica as etapas a serem executadas através de um conjunto de operações. Esta sequencia de operações forma uma expressão em álgebra relacional, do qual o resultado originará também uma consulta.

A álgebra relacional faz parte da manipulação de dados do modelo relacional e consiste em um conjunto de operações que fornece uma nova ligação a partir de uma ou mais relações existentes no banco de dados, dessa maneira proporciona os procedimentos essenciais para a execução de uma sequência de operações, de tal forma a adquirir o resultado desejado independente da forma de tratamento do banco de dados (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

As operações da álgebra relacional pode ser decomposto em dois grupos:

a) Conjunto de operações da teoria dos conjuntos: UNION (união), INTERSECTION (interseção), DIFFERENCE (diferença) e CARTESIAN PRODUCT (produto cartesiano) (DATE,2004);

b) Conjunto de operações projetadas para banco de dados relacionais: SELECT (seleção), PROJECT (projeção) e JOIN (junção) (DATE,2004).

2.2.7.2 Heurística na otimização

Segundo Siberschartz, Korth e Sudarshan (1999) a heurística trabalha diretamente com a álgebra relacional com objetivo de converter as consultas para realizar as operações de seleção o mais breve possível.

O otimizador baseado em heurística realiza seleções antecedendo a projeção, pois nesse caso há grandes chances de obter-se a redução das relações (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

Em resumo, as heurísticas reestruturam uma representação preliminar de uma árvore de consulta, dessa forma as operações que reduzem o tamanho dos resultados intermediários são executadas primeiro (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

2.2.8 Transações

Segundo Ramakrishnan e Gehrke (2008) uma transação é vista pelo SGBD como uma série ou lista de ações. As ações que podem ser executadas por uma transação incluem leituras e gravações de objetos de banco de dados.

Cada transação além de ler e gravar deve especificar ação final ou a efetivação (incluir com sucesso) ou o cancelamento (desfazer todas as ações executadas até o momento). As transações interagem umas com as outras apenas por meio de operações de leitura e gravação do banco de dados, por exemplo, elas não podem trocar mensagens (RAMAKRISHNAN; GEHRKE, 2008).

De acordo com Siberschartz, Korth e Sudarshan (1999) um conjunto de várias operações no banco de dados é vista pelo usuário como uma única operação. Por exemplo, a transferência de fundos de uma conta corrente para uma poupança é uma operação única sob o ponto de vista do cliente, porém internamente no banco de dados ela envolve várias operações.

Uma transação atualiza vários itens de dados e geralmente é o resultado da execução de um programa de usuário escrito em uma linguagem de programação. Cada transação é delimitada por declarações que marcam seu início e fim, as operações que serão executados nessa transação são todas as invocadas ali entre o começo e o fim (SILBERSCHATZ; KORTH; SUDARSHAN, 1999).

De acordo com Siberschartz, Korth e Sudarshan (1999) para assegurar a integridade dos dados é exigido que o sistema de banco de dados mantenha as seguintes propriedades das transações:

a) Atomicidade: Ou todas as operações da transação são executadas com sucesso no banco de dados ou nenhuma o será executada;

b) Consistência: A execução de uma transação isolada, ou seja, sem a execução concorrente de uma outra transação, preserva a consistência do banco de dados;

c) Isolamento: Sabendo que diversas transações podem ser executadas de forma concorrente o sistema deve garantir que, para cada par de transações T_i e T_j, o T_j tenha terminado sua execução antes de T_i começar, ou que T_j tenha começado sua execução após T_i terminar. Assim, cada transação não toma conhecimento de outras transações concorrentes no sistema;

d) Durabilidade: Depois da transação ser finalizada com sucesso, as mudanças que ela fez no banco de dados persistem até mesmo se houver falhas no sistema.

2.2.9 Controle de concorrência

Uma das propriedades fundamentais de uma transação é o isolamento. Entretanto, quando várias transações são executadas concorrentemente no banco de dados, essa propriedade pode não ser mantida. Para garantir que essa propriedade seja preservada o SGBD precisa controlar a interação entre as transações concorrentes. Esse controle é atingido por uma série de procedimentos chamados de esquemas de controle de concorrência (SILBERSCHATZ; KORTH; SUDARSHAN, 2012).

De acordo com Siberschartz, Korth e Sudarshan (2012) existem uma variedade de esquemas (protocolos) de controle de concorrência, entre eles estão, protocolo baseados em bloqueios, protocolo de bloqueios baseados em duas fases e protocolo baseado em grafo.

2.2.9.1 Protocolo baseados em bloqueio

Nesse esquema é exigido que os dados sejam acessados de maneira mutuamente exclusiva, ou seja, enquanto uma transação está acessando uma informação ou dado, nenhuma outra transação pode modificar esse dado. O método mais comum para atender esse requisito é permitir que uma transação acesse esse dado somente se estiver atualmente mantendo um bloqueio sobre o mesmo (SILBERSCHATZ; KORTH; SUDARSHAN, 2012).

2.2.9.2 Protocolo de bloqueio em duas fases

Segundo Siberschartz, Korth e Sudarshan (2012) esse esquema requer que cada transação emita solicitações de bloqueio e desbloqueio em duas fases:

a) Fase de crescimento: Uma transação pode obter bloqueios, mas não pode liberar qualquer bloqueio;

b) Fase de encolhimento: Uma transação pode liberar bloqueios mas não pode liberar novos bloqueios.

Inicialmente, uma transação encontra-se na fase de crescimento e adquire bloqueios conforme a necessidade. Quando a transação libera um bloqueio ela entra na fase de encolhimento e não pode emitir mais solicitações de bloqueio (SILBERSCHATZ; KORTH; SUDARSHAN, 2012).

O bloqueio em duas fases não garante o surgimento de impasse (*deadlock*).

2.2.9.3 Protocolo baseado em grafo

Nesse protocolo precisamos de informações adicionais sobre como cada transação acessará o banco de dados. Existem vários modelos que nos dão as informações adicionais. O modelo mais simples exige conhecimento prévio sobre a ordem em que os itens do banco de dados serão acessados (SILBERSCHATZ; KORTH; SUDARSHAN, 2012).

No protocolo baseado em grafo a única instrução de bloqueio permitida é o bloqueio de modo exclusivo. Cada transação T_i pode bloquear um item de dados no máximo uma vez e precisa observar as seguintes regras:

a) O primeiro bloqueio por T_i pode ser sobre qualquer item de dados;

b) Subsequentemente, um item de dados A pode ser bloqueado por T_i somente se o pai de A estiver atualmente bloqueado por T_i;

c) Os itens de dados podem ser desbloqueados a qualquer momento;

d) Um item de dados que foi bloqueado e desbloqueado por T_i não pode ser bloqueado novamente por T_i.

2.2.10 Impasses

Segundo Elmasri e Ramez (2011) um impasse (*deadlock*) ocorre quando cada transação em um conjunto de duas ou mais transações está esperando por algum item que está bloqueado por alguma outra transação.

Considere o exemplo a seguir: A transação T_i define um bloqueio exclusivo sobre o dado A, T_j define um bloqueio exclusivo sobre B, T_i requisita um bloqueio exclusivo sobre B e é enfileirada, e T_j requisita um bloqueio exclusivo sobre A e é enfileirada. Agora, T_i está esperando que T_j libere seu bloqueio e T_j está esperando que T_i libere seu bloqueio. Esse ciclo de transações esperando que os bloqueios sejam liberados é chamado de impasse (*deadlock*). Podemos deduzir que essas duas transações não terão nenhum progresso. O SGBD deve evitar ou detectar as situações de impasse (RAMAKRISHNAN; GEHRKE, 2008).

Se um impasse não for identificado pode ocorrer o travamento do SGBD onde uma transação fica esperando pela outra eternamente, até o serviço ficar indisponível. Os impasses podem também ser camuflados por um *timeout* de transação, que pode ser configurado na maioria dos SGBDs, nesse caso não ocorre o travamento, porém, nenhuma das transações concorrentes são executadas e podem passar despercebidas, caso não haja um monitoramento do banco de dados (ELMASRI; RAMEZ, 2011).

2.2.11 Índices

Segundo Rob e Coronel (2011) um índice pode ser considerado uma estrutura ou arquivo associado a uma tabela, e tem como objetivo melhorar o desempenho de acesso à dados de uma ou mais linhas de uma tabela, dessa maneira o índice cria ponteiros para os dados armazenados nas colunas que devem ser especificas em sua criação.

Os índices podem ser utilizados de diferentes maneiras e levar a planos de execução mais rápidos do que qualquer outro que não utiliza índices (RAMAKRISHNAN; GEHRKE, 2008).

Um índice é uma disposição ordenada utilizada para acessar logicamente uma tabela. Considerando de um ponto de vista conceitual, é formado de uma chave de índice e de um conjunto de ponteiros, onde, a chave de índice é o ponto de referência do mesmo. Cada chave aponta para a localização dos dados identificados por ela (ROB; CORONEL, 2011).

Os índices executam um papel importante na estrutura de um SGBD, pois ao definir a chave primária de uma tabela, o SGBD cria automaticamente um índice exclusivo para a(s) coluna(s) dessa chave (ROB; CORONEL, 2011).

Uma tabela pode ter vários índices, porém, cada um deles está associado a apenas uma tabela (ROB; CORONEL, 2011).

Por exemplo, suponha que você queira procurar todas as pinturas criadas por um determinado pintor no banco de dados. Sem um índice, é necessário ler todas as linhas da tabela pintura e ver se o atributo correspondente ao código do pintor é do pintor solicitado. No entanto, se for criado um índice utilizando-se o código do pintor, basta procurar o valor adequado desse atributo no índice e encontrar os ponteiros correspondentes. Em termos conceituais, o índice se assemelha à representação ilustrada na figura 4 (ROB; CORONEL, 2011).

Figura 4 - Etapas no processamento de consultas no banco de dados.

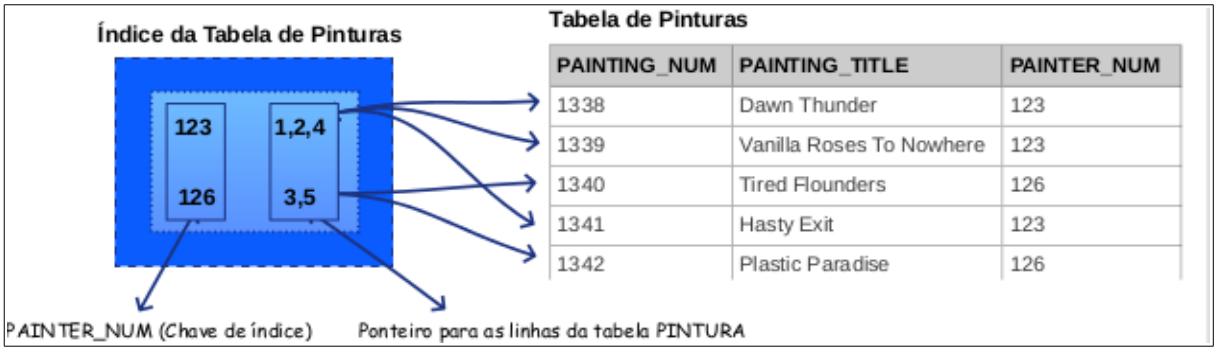


Tabela de Pinturas

PAINTING_NUM	PAINTING_TITLE	PAINTER_NUM
1338	Dawn Thunder	123
1339	Vanilla Roses To Nowhere	123
1340	Tired Flounders	126
1341	Hasty Exit	123
1342	Plastic Paradise	126

Fonte: Modificado de Rob e Coronel (2011).

2.2.12 PostgreSQL

Nessa seção são abordados os principais conceitos envolvendo o PostgreSQL que é o sistema gerenciador de banco de dados inicialmente monitorado pela ferramenta proposta. São abordadas as limitações da ferramenta, características e informações sobre a coleta e visualização de estatísticas.

O PostgreSQL é um SGBD relacional *open source* com mais de 15 anos de desenvolvimento. É extremamente robusto e confiável, além de ser extremamente flexível e rico em recursos (BIAZUS, 2003).

O PostgreSQL derivou do projeto POSTGRES da Universidade de Berkley, cuja última versão foi a 4.2, originalmente patrocinado pelo DARPA (Agência de Projetos de Pesquisa Avançada para Defesa), ARO (Departamento de Pesquisa Militar) e NSF (Fundação Científica Nacional) (BIAZUS, 2003).

O desenvolvimento do projeto POSTGRES teve inicio em 1986 e em 1987 já estava operacional. A primeira versão lançada para o público externo foi em 1989 (BIAZUS, 2003).

Segundo Biazus (2003) o PostgreSQL pode ser considerado um SGBD objeto-relacional, por conter algumas características de orientação a objetos, como herança e tipos personalizados.

O PostgreSQL é compatível com diversos sistemas operacionais entre eles podemos citar: Windows, Linux, MacOS e Solaris. Além disso, o SGBD fornece suporte a diversas plataformas e linguagens de programação como: Java, C, Python, PHP e Ruby (MILANI, 2008).

Segundo Milani (2008) o PostgreSQL encontra-se em uma versão estável e confiável, e disponibiliza os principais recursos existentes nos sistemas gerenciadores de banco de dados pagos do mercado e com capacidade para suprir pequenas, médias e grandes aplicações.

2.2.12.1 Características do PostgreSQL

A seguir, algumas características existentes no PostgreSQL:

a) Suporte a transações: O PostgreSQL possui suporte a operações ACID (Atomicidade, Consistência, Isolamento e Durabilidade). Cada uma destas propriedades garante a qualidade dos serviços disponibilizados pelo SGBD (MILANI, 2008);

b) Alta disponibilidade: Visando expansão da capacidade de processamento o PostgreSQL possibilita que atue como um *cluster* de informações (MILANI, 2008);

c) Multithreads: O PostgreSQL possibilita mais de uma conexão com o banco de dados, por meio de recurso de *multithereads*, ou seja, permite que mais de uma pessoa possa acessar a mesma informação simultaneamente sem ocasionar atrasos ou filas de acesso (MILANI, 2008);

d) Segurança e criptografia: O PostgreSQL possui suporte nativo a SSL, possibilitando conexões seguras através destes canais para trafegar informações consideradas sigilosas (MILANI, 2008);

e) Sql: Adota os padrões ANSI SQL na implementação de suas funcionalidades (MILANI, 2008);

f) Incorporável em aplicações gratuitamente: Por utilizar a licença BSD (*Berkeley Software Distribution*) pode ser livremente incorporado em aplicações pessoais e/ou comerciais (MILANI, 2008);

g) Capacidade de armazenamento: O PostgreSQL suporta de maneira eficiente e confiável grandes tamanhos de informações não tendo limite máximo para um banco de dados.

2.2.12.2 Limitações do PostgreSQL

Segundo Smanioto (2007) o PostgreSQL é bem flexível no que diz respeito a processador e plataforma. O SGBD pode ser instalado de forma instável em qualquer tipo de hardware com diversos tipos de sistemas operacionais entre eles Linux, UNIX (AIX, BSD, HP-UX, SGI ARIX, MAC OS X, Solaris, Tru64) e Windows. A tabela 1 apresenta algumas limitações.

Tabela 1 - Limitações do PostgreSQL.

Limite	Valor
Tamanho máximo do banco de dados	Ilimitado
Tamanho máximo de uma tabela	32 TB
Tamanho máximo de uma linha de tabela	1.6 TB
Tamanho máximo de um registro	1 GB
Quantidade de linhas por tabela	Ilimitado
Quantidade de colunas por tabela	250 à 1600. Depende do tipo de coluna
Quantidade de index por tabela	Ilimitado

Fonte: Modificado de Smanioto (2007).

2.2.12.3 Catálogo do PostgreSQL

Os catálogos do SGBD PostgreSQL são tabelas como qualquer outra do banco de dados. Estas tabelas podem ser removidas e recriadas, podem ser adicionadas novas colunas, podem ser inseridos e atualizados valores, porém nada disso é recomendável, pois normalmente esses dados são inseridos e atualizados automaticamente pelo SGBD (POSTGRESQL, 2015).

2.2.12.4 Coletor de estatísticas do PostgreSQL

O coletor de estatísticas do PostgreSQL é um subsistema do SGBD que coleta informações sobre as atividades do servidor que estejam relacionadas aos dados armazenados no banco de dados. Com as estatísticas coletadas é possível analisar, por exemplo, acessos em tabelas e índices (POSTGRESQL, 2015).

O PostgreSQL armazena apenas o último status de cada estatísticas, não permitindo que se faça uma análise em uma linha do tempo. Através da ferramenta desenvolvida o usuário DBA tem uma base histórica de cada estatística coletada, podendo consultá-la de maneira gráfica para o melhor entendimento.

2.2.12.4.1 Configuração para coleta de estatísticas

A coleta de estatísticas no PostgreSQL pode deixar o servidor sobrecarregado, dessa forma, o SGBD pode ser configurado para coletar ou não coletar informações. Isto é controlado por parâmetros de configuração que são normalmente definidos no arquivo postgresql.conf (POSTGRESQL, 2015).

Conforme PostgreSQL (2015) segue-se alguns parâmetros de configuração que controlam a coleta de estatísticas:

a) *track_activities:* Este parâmetro quando ativado permite o monitoramento do comando atual que está sendo executado por qualquer processo no banco de dados;

b) *track_counts:* Este parâmetro quando ativado coleta estatísticas sobre acessos em tabelas e índices;

c) *track_functions:* Este parâmetro quando ativado permite o acompanhamento do uso de funções definidas pelo usuário;

d) *track_io_timing:* Este parâmetro quando ativado permite o monitoramento de tempo de leitura e escrita em um bloco de transação.

Geralmente, esses parâmetros são definidos no arquivo postgresql.conf para que tenham efeito em todos os processos do servidor, mas é possível ativá-los ou desativá-los em sessões individuais usando o comando *SET*. Porém, por motivos de segurança somente usuários com privilégios administrativos podem alterar essas configurações utilizando o comando *SET* (POSTGRESQL, 2015).

Para que a base de dados da ferramenta desenvolvida seja alimentada é importante que os parâmetros identificados acima estejam ativados, pois é através dos dados coletados pelo PostgreSQL que a base do SMBD (Sistema de Monitoramento de Base de Dados) é populada.

2.2.12.4.2 Visualizando as estatísticas coletadas

Segundo PostgreSQL (2015) existem várias visões que permitem o resultado das estatísticas coletadas, dessa forma pode-se construir visualizações customizadas usando os recursos nativos do SGBD PostgreSQL.

Conforme PostgreSQL para construir visualizações customizadas podem ser utilizadas as visões nativas listadas na figura 5, através dessas visões que a ferramenta desenvolvida extrai os dados para que sejam exibidos de maneira gráfica para o melhor entendimento do usuário DBA.

Figura 5 - Figura de visões de estatísticas nativas do PostgreSQL.

Nome	**Descrição**
pg_stat_activity	Retorna uma linha por processo que está sendo executado no servidor, mostrando informações relacionadas a atividade atual desse processo.
pg_stat_bgwriter	Retorna uma linha única, mostrando estatísticas sobre a atividade do processo em *background* de escrita.
pg_stat_database	Retorna uma linha por banco de dados, mostrando as estatísticas de todo o banco de dados.
pg_stat_all_tables	Retorna uma linha para cada tabela do banco de dados atual, mostrando estatísticas sobre acessos a essa tabela em específico.
pg_stat_sys_tables	Retorna o mesmo que a *pg_stat_all_tables*, exceto que somente são mostradas as tabelas do sistema.
pg_stat_user_tables	Retorna o mesmo que a *pg_stat_all_tables*, exceto que somente são mostradas as tabelas de usuário.
pg_stat_xact_all_tables	Tem um resultado semelhantes a *pg_stat_all_tables*, mas conta as medidas tomadas até agora dentro da transação corrente (que ainda não estão incluídas no *pg_stat_all_tables*).
pg_stat_xact_sys_tables	Tem o mesmo resultado que a *pg_stat_xact_all_tables*, exceto que somente são mostradas as tabelas do sistema.
pg_stat_xact_user_tables	Tem o mesmo resultado que a *pg_stat_xact_all_tables*, exceto que somente são mostradas as tabelas de usuário.
pg_stat_all_indexes	Retorna uma linha para cada índice do banco de dados atual, mostrando estatísticas sobre acessos a esse índice em específico.
pg_stat_sys_indexes	Tem o mesmo resultado que a *pg_stat_all_indexes*, exceto que somente os índices das tabelas do sistema são mostradas.

pg_stat_user_indexes	Tem o mesmo resultado que a *pg_stat_all_indexes*, exceto que somente índices das tabelas do usuário são mostrados.
pg_statio_all_tables	Retorna uma linha para cada tabela no banco de dados atual, mostrando estatísticas sobre I / O da tabela específica.
pg_statio_sys_tables	Tem o mesmo resultado que a *pg_statio_all_tables*, exceto que somente são mostradas as tabelas do sistema.
pg_statio_user_tables	Tem o mesmo resultado que a *pg_statio_all_tables*, exceto que somente são mostradas as tabelas de usuário.
pg_statio_all_indexes	Retorna uma linha para cada índice no banco de dados atual, mostrando estatísticas sobre I / O do índice em específico.
pg_statio_sys_indexes	Tem o mesmo resultado que *pg_statio_all_indexes*, exceto que somente os índices das tabelas do sistema são mostradas.
pg_statio_user_indexes	Tem o mesmo resultado que *pg_statio_all_indexes*, exceto que somente índices das tabelas do usuário são mostrados.
pg_statio_all_sequences	Retorna uma linha para cada sequência no banco de dados atual, mostrando estatísticas sobre I/O da sequência em específico.
pg_statio_sys_sequences	Retorna o mesmo que *pg_statio_all_sequences*, exceto que somente sequências de sistema são mostradas.
pg_statio_user_sequences	Retorna o mesmo que *pg_statio_all_sequences*, exceto que somente sequências de usuários são mostrados.
pg_stat_user_functions	Retorna uma linha pra cada função, mostrando estatísticas sobre suas execuções.
pg_stat_xact_user_functions	Resultado semelhante a *pg_stat_user_functions*, mas conta apenas chamadas enquanto a transação etá em andamento.
pg_stat_replication	Retorna uma linha por processo remetente WAL, mostrando estatísticas sobre a replicação para o servidor de espera.
pg_stat_database_conflicts	Retorna linha por banco de dados, mostrando as estatísticas de todo o banco de dados sobre as consultas cancelas devido à conflitos.

Fonte: Modificado de PostgreSQL (2015).

Se forem usadas as estatísticas para monitorar os processos que estão sendo executados, é importante perceber que os dados não são atualizados instantaneamente, as informações estatísticas são atualizadas após o término das transações, dessa forma, as transações que ainda estão em processo não afetam os totais exibidos. Quando um processo do banco de dados solicita as estatísticas coletadas, primeiramente ele busca o relatório mais recente emitido pelo coletor e em seguida será utilizado até o final da transação corrente, este recurso permite que você execute várias consultas sobre as estatísticas sem se preocupar com números mudando continuamente com transações que ainda não terminaram (POSTGRESQL, 2015).

A transação também tem suas próprias estatísticas e podem ser visualizadas utilizando as visões *pg_stat_xact_all_tables*, *pg_stat_xact_sys_tables*, *pg_stat_xact_user_tables* e *pg_stat_xact_user_functions* (POSTGRESQL, 2015).

2.2.12.5 O Comando EXPLAIN

O comando EXPLAIN mostra o plano de execução gerado pelo planejador do PostgreSQL para o comando fornecido. O plano de execução mostra como as tabelas referenciadas pelo comando serão varridas, por uma varredura sequencial simples ou uma varredura por um índice e etc. Do que é mostrado, a parte mais importante é o custo estimado de execução do comando, que é a estimativa feita pelo planejador de quanto tempo vai demorar para executar o comando (POSTGRESQL, 2015).

Para visualizar o plano que o planejador cria para a consulta pode ser utilizado o comando EXPLAIN + consulta (POSTGRESQL, 2015).

2.2.12.6 O comando VACUUM

O comando VACUUM recupera o armazenamento ocupado por tuplas mortas. Em condições normais o PostgreSQL mantém tuplas obsoletas para ter mais agilidade na execução de um comando que atualizam ou excluem informações do banco, essas tuplas só são fisicamente removidas quando o comando VACCUM é executado. Portanto, o mesmo deve ser periodicamente executado, especialmente em tabelas frequentemente atualizadas (POSTGRESQL, 2015).

Segundo PostgreSQL (2015) o comando VACUUM pode ser executado agrupado com alguns parâmetros, que serão listado abaixo:

a) *FULL:* Faz uma limpeza completa em tuplas mortas e pode recuperar mais espaço, mas leva muito mais tempo e bloqueia as tabelas. Esse método também requer espaço em disco extra, uma vez que ele faz uma cópia das tabelas e só libera a cópia antiga quando a operação estiver concluída;

b) *VERBOSE* : Imprime um relatório detalhado da atividade de limpeza de cada tabela;

c) *ANALYZE:* Atualiza as estatísticas utilizadas pelo planejador para determinar a maneira mais eficiente de executar uma consulta.

2.2.12.7 Parâmetros de configuração que podem melhorar a performance

Segundo Berkus (2005) alguns parâmetros do SGBD PostgreSQL se ajustados de maneira correta podem refletir em aumento de performance. Segue na figura 6 alguns desses parâmetros.

Figura 6 - Quadro de parâmetros que podem melhorar a performance do PostgreSQL.

Nome	**Descrição**
shared_buffers	Bloco de memória dedicado ao PostgreSQL utilizado para as operações ativas.
work_mem	Bloco de memória não compartilhada, sendo alocada para cada operação, esta configuração coloca um teto na quantidade de memória que uma única operação ocupar antes de ser forçada para o disco.
maintenance_work_mem	Bloco de memória utilizada pelo PostgreSQL para executar os comandos VACUUM, ANALYZE, CREATE INDEX, e adição de chaves estrangeiras.
checkpoint_segments	Bloco de memória que define o tamanho do cache do log de transações para operações de escrita.
effective_cache_size	Bloco de memória que define ao planejador de consultas o maior objeto do banco de dados que pode se esperar ser cacheado.

Fonte: Modificado de Berkus (2005).

Através da ferramenta desenvolvida é possível consultar as configurações do SGBD, visualizando os valores setados e os valores *default* de cada configuração, além disso é mantida uma base histórica do valor da mesma, para que seja consultada caso haja alguma dúvida do valor de uma configuração em determinada data.

3 TRABALHOS RELACIONADOS

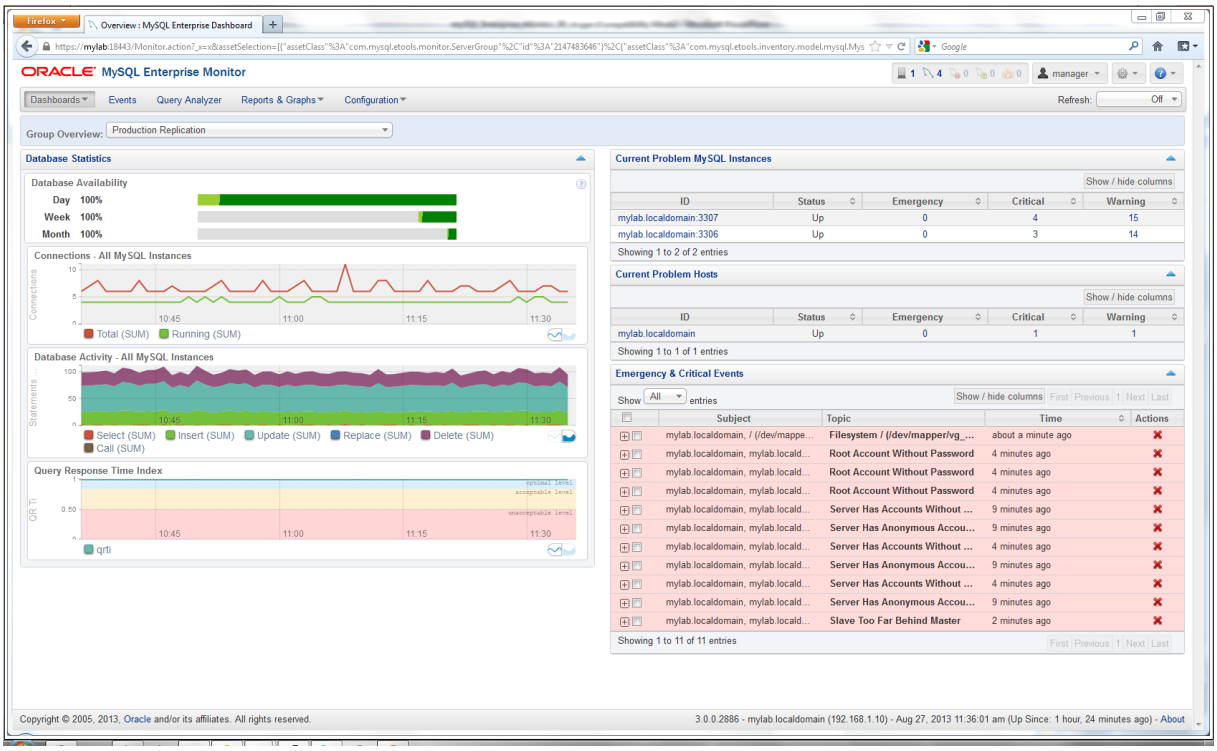
Nessa sessão são apresentadas algumas ferramentas já existentes para análise e monitoramento de estatísticas para bases de dados que utilizam SGBDs.

3.1 Ferramenta MySQL Enterprise Monitor

Segundo Oracle (2015) o MySQL Enterprise Monitor é um sistema de monitoramento para base de dados que utilizam o SGBD MySQL, o sistema notifica sobre problemas e aconselha como corrigi-los.

O MySQL Enterprise Monitor monitora continuamente consultas e métricas de desempenho de uma ou mais base de dados que utiliza o SGBD MySQL, o sistema alerta o DBA sobre desvios significativo das métricas e recomenda alterações em configurações para manter o desempenho ORACLE (2015). Segue na figura 7 um *screenshot* da ferramenta MySQL Enterprise Monitor.

Figura 7 - Screenshot da ferramenta MySQL Enterprise Monitor.



Fonte: Oracle (2015).

O programa MySQL possui licença dupla. Os usuários podem optar por usar o software MySQL como um produto *open source* sob os termos da GNU (*General Public License*) ou podem comprar uma licença comercial padrão da Oracle.

3.2 Ferramenta Pgwatch

O Pgwatch é uma ferramenta para monitorar bancos de dados PostgreSQL que a partir de sua versão 2.0 se tornou uma ferramenta *open source*. A versão 2.0 encontra-se em desenvolvimento e pode ser adquirida através do site da mantenedora (CYBERTEC, 2015).

A ferramenta contém um *daemon,* ou seja, um programa de computador que roda de forma independente sem ser controlado diretamente por um usuário, esse programa coleta periodicamente estatísticas do sistema de banco de dados, as estatísticas são armazenadas em uma base de dados central do Pgwatch para uma posterior análise (CYBERTEC, 2015).

Atualmente o Pgwatch em sua versão de desenvolvimento suporta as versões 9.0 e 9.1 do PostgreSQL, não contemplando as versões mais atuais como a 9.3. (CYBERTEC, 2015). Segue na figura 8 um *screenshot* da ferramenta Pgwatch.

Figura 8 - Screenshot da ferramenta Pgwatch.

Fonte: Cybertec (2015).

O Pgwatch está licenciado sob a licença Creative Commons e pode ser baixado gratuitamente (CYBERTEC, 2015).

3.3 Ferramenta Pganalytics

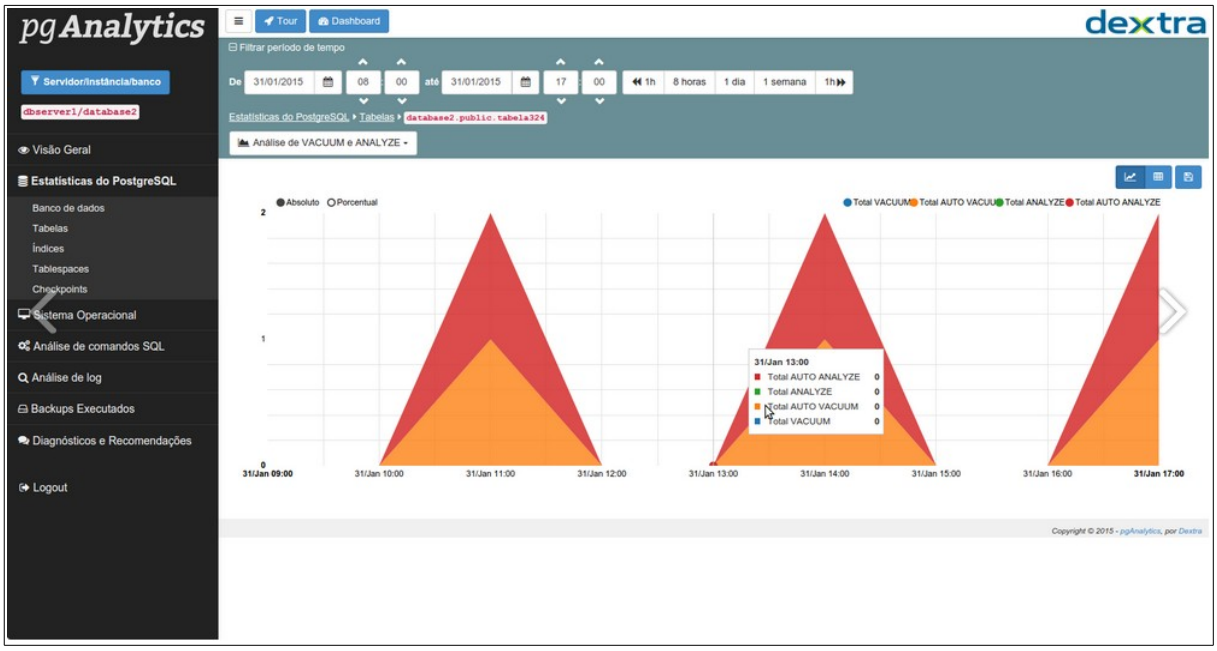
O Pganalytics é uma ferramenta SaaS (Software as a Service) desenvolvida pela Dextra com o objetivo de coleta e exibição de dados estatísticos de bases de dados que utilizam o SGBD PostgreSQL (DEXTRA, 2015).

A ferramenta disponibiliza diagnósticos e recomendações que auxiliam na administração do banco de dados, a ferramenta conta também com com um recurso baseado

em alertas avisando o usuário sobre incidentes que podem afetar o funcionamento do SGBD (DEXTRA, 2015).

O Pganalytics disponibiliza para seus clientes um agente não intrusivo que deve ser instalado no servidor de banco de dados que pode ser Linux ou Windows. O agente coleta estatísticas e sincroniza as informações com um servidor mantido pela Dextra. Para utilizar a ferramenta é necessário criar uma conta e pagar uma mensalidade que varia de acordo com o plano de monitoramento escolhido (DEXTRA, 2015). Segue na figura 9 um *screenshot* da ferramenta Pganalytics.

Figura 9 - Screenshot da ferramenta Pganalytics.



Fonte: Dextra (2015).

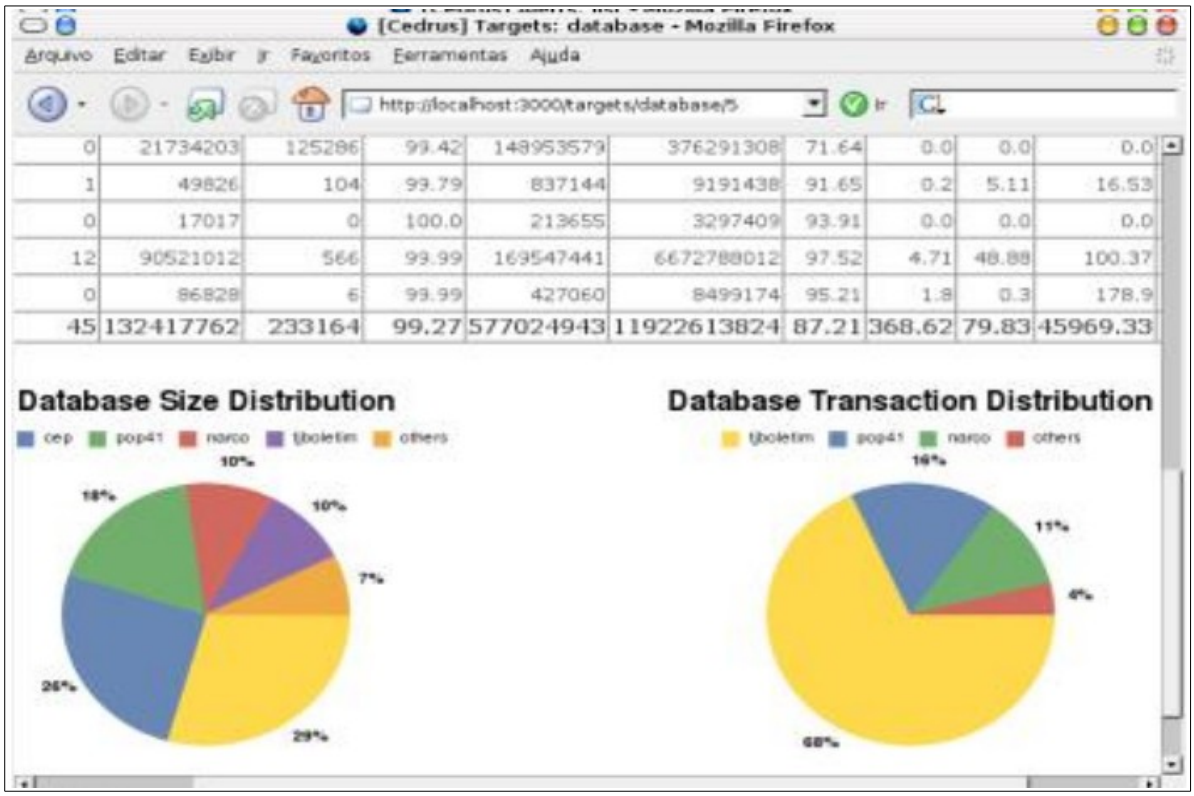
3.4 Ferramenta Cedrus

O Cedrus é uma ferramenta de administração e monitoramento gráfica para PostgreSQL inspirada no Enterprise Manager da Oracle. A ferramenta coleta informações úteis sobre uma ou mais instâncias do PostgreSQL e do sistema operacional onde a base de dados está instalada. A coleta de dados é feita através de um agente instalado no servidor de banco de dados, o coletor envia para a base de dados do Cedrus todas as informações obtidas, sendo assim, é possível ter uma base histórica de todas as informações coletadas (CEDRUS, 2006).

Através da interface gráfica do sistema é possível emitir relatórios e visualizar gráficos sobre informações do sistema operacional e estatísticas coletadas do banco de dados (CEDRUS, 2006).

O Cedrus foi desenvolvido em 2006 e não teve continuidade, não suportando versões mais atuais do PostgreSQL (CEDRUS, 2006). Segue na figura 10 um *screenshot* da ferramenta Cedrus.

Figura 10 - Screenshot da ferramenta Cedrus.



Fonte: Cedrus (2006).

A ferramenta Cedrus possui licença GNU General Public License version 2.0 (GPLv2) (CEDRUS, 2006).

Na sessão seguinte foi realizada uma análise comparativa entre as ferramentas estudas, com a finalidade de ressaltar os pontos fortes e fracos de cada uma delas.

3.5 Análise comparativa entre as ferramentas estudadas

Ao analisar as ferramentas descritas acima conclui-se que há soluções que fornecem informações para auxiliar na administração, monitoramento e otimização de um banco de dados. No entanto, algumas são proprietárias ou estão descontinuadas tornando seu uso mais restrito.

A figura 11 apresenta um comparativo entre as ferramentas estudadas, levando em consideração algumas características consideradas relevantes como:

a) Administração de múltiplos servidores: Identifica se a ferramenta possibilita monitorar mais de um servidor de banco de dados;

b) Open source: Identifica se a ferramenta é de livre distribuição;

c) Compatibilidade com versões atuais de SGBDs: Identifica se a ferramenta pode ser utilizada com as versões mais atuais de SGBDs;

d) Alarmísticas: Identifica se a ferramenta possuí funcionalidades que avisam ao usuário de possíveis problemas com o SGBD;

e) Gráficos pré definidos: Identifica se a ferramenta já possui algum gráfico de monitoramento pré definido;

f) Multiplataforma: Identifica se a ferramenta pode ser utilizada quando o SGBD está instalado em diferentes sistemas operacionais;

g) Geração de relatórios: Informa se a ferramenta possibilita a geração de relatórios das informações coletadas.

Figura 11 - Quadro comparativo entre ferramentas de análise e monitoramento para PostgreSQL.

Características	**MySQL Enterprise Monitor**	**Pgwatch**	**Pganalytics**	**Cedrus**
Administração de múltiplos servidores	Sim	Sim	Sim	Sim
Open source	Sim	Sim	Não	Sim
Compatibilidade com versões atuais de SGBDs	Sim	Não	Sim	Não
Alarmísticas	Sim	Não	Sim	Sim
Gráficos pre definidos	Sim	Sim	Sim	Sim
Multiplataforma	Sim	Sim	Sim	Não
Geração de relatórios	Sim	Sim	Sim	Sim

Fonte: Elaborado pelo autor.

Após a análise da arquitetura e funcionamento das ferramentas estudadas, nota-se que a maioria conta com um agente coletor que deve ser instalado no servidor de banco de dados a ser monitorado. Este coleta as informações locais e as concentram em um servidor para análise futura.

Além de centralizar as informações em um servidor único, as ferramentas disponibilizam uma interface que permite ao DBA visualizar diversos gráficos que evidencia a necessidade de alterar configurações do SGBD ou efetuar alguma manutenção na estrutura do banco de dados.

Algumas das ferramentas estudadas contam também com um recurso de alertas, que avisa o administrado do bando de dados sobre problemas automaticamente identificados ou manualmente redigidos por uma equipe interna, que é o cado do Pganalytics.

De acordo com o estudo feito sobre as ferramentas foi possível verificar que algumas já estão defasadas e não receberam novas funcionalidades nos últimos anos, existe ainda o caso do Cedrus que de acordo com a documentação está descontinuado desde 2006.

Nos capítulos seguintes é explicado os materiais e métodos utilizados na implementação do SMBD e o detalhamento do desenvolvimento da ferramenta que segue a ideia explanada nos parágrafos anteriores onde os sistemas possuem um agente local para captura de informações e um servidor único para armazená-las.

4 MATERIAIS E MÉTODOS

Este capítulo tem como objetivo observar os artefatos, documentos e técnicas que foram utilizados para o desenvolvimento da ferramenta proposta. Nas próximas seções pode-se visualizar detalhadamente cada um deles.

4.1 Metodologia

Nesta seção é apresentado o enquadramento metodológico utilizado neste estudo, com finalidade de atender aos objetivos propostos.

De acordo com Gil (2002) a pesquisa tem um caráter pragmático, e é um "processo formal e sistemático de desenvolvimento do método científico".

Uma pesquisa exploratória tem como objetivo aumentar o conhecimento do pesquisador sobre um determinado problema. Ela pode ser composta por uma revisão de literatura, entrevistas, entre outros (GIL, 2002).

O presente trabalho teve como objetivo geral disponibilizar uma ferramenta web de coleta e análise de estatísticas para o sistema gerenciador de banco de dados PostgreSQL. Para isso, foram realizados estudos sobre os temas relacionados, com a finalidade de desenvolver uma ferramenta em conformidade com os conceitos já existentes. Desta forma, este trabalho se caracteriza conforme Santos (1999), de acordo com o seu objetivo, como pesquisa exploratória.

A ferramenta desenvolvida foi utilizada e testada em um ambiente de produção, tendo como base, um local com volume considerável de informações armazenadas e um grande número de acessos simultâneos. Sendo assim, a sua utilização se da em um ambiente real.

Após análise geral dos resultados obtidos com a ferramenta, foi avaliado o ganho que se teve com o seu uso e foram realizadas as devidas conclusões.

4.2 Visão geral

A ferramenta desenvolvida teve como objetivo a criação de uma ferramenta web, *open source* que possibilita ao usuário DBA extrair e visualizar estatísticas do software PostgreSQL e futuramente de qualquer SGBD, possibilitando o monitoramento e a tomada de ações para garantir a performance da base de dados. Através da ferramenta desenvolvida são coletadas estatísticas do banco de dados que permitem detectar impasses que estejam relacionados a administração do banco de dados, podendo assim, tornar visível a necessidade de criação de índices e alterações em configurações do sistema utilizado.

O SMBD (Sistema de Monitoramento de Base de Dados) possibilita inicialmente apenas o monitoramento do SGBD PostgreSQL como um módulo da ferramenta, dessa maneira, oportuniza a expansão para atender outros SGBDs em trabalhos futuros.

4.3 Tecnologias utilizadas

Com o objetivo de desenvolver uma ferramenta em conformidade com tendências do desenvolvimento web, conceitos como design responsivo foram adotadas, com a finalidade de melhorar a usabilidade, para que isso fosse possível foi necessário utilizar algumas tecnologias que serão explicadas abaixo:

a) PHP: (*Hypertext Preprocessor*) é uma linguagem de script *open source* de uso geral, muito utilizada, e adequada para o desenvolvimento web. O php[1] foi utilizado para desenvolver o agente coletor e os processamentos relacionados a ferramenta web (PHP, 2015);

[1]http://www.php.net

b) Apache HTTP Server: É um software de código aberto que tem como objetivo disponibilizar páginas HTML à web. O Apache[2] foi utilizado como servidor web para os usuários terem acesso a ferramenta desenvolvida (Apache, 2015);

c) PostgreSQL: É um SGBD (Sistema Gerenciador de Banco de Dados) objeto relacional de código aberto extremamente robusto e confiável, O PostgreSQL[3] foi utilizado para armazenar as estatísticas e informações coletadas, usuários que irão acessar a ferramenta e configurações do sistema (PostgreSQL, 2015);

d) HTML: (*HyperText Markup Language*) é uma linguagem de marcação da internet, utilizada para desenvolver as páginas na web. O HTML[4] foi utilizado para criar a interface da ferramenta (HTML, 2015);

e) JavaScript: É uma linguagem de programação do lado cliente processada pelo próprio navegador. O JavaScript[5] foi utilizado para criar efeitos nas interfaces da ferramenta para proporcionar uma maior interatividade (JAVASCRIPT, 2015);

f) Bootstrap: É um *framework* para facilitar a criação de sites ou páginas web (responsivo). O Bootstrap[6] foi utilizado para deixar a ferramenta mais intuitiva e utilizável em qualquer dispositivo que tenha acessoa a web. (BOOTSTRAP, 2015);

g) Morris: É uma biblioteca leve que usa jQuery e para tornar fácil desenhar gráficos simples. O Morris[7] foi utilizado para representar graficamente as estatísticas coletadas.

4.4 Levantamento de requisitos

Para que a ferramenta pudesse atingir os objetivos desejados foi necessário que atendessem determinados requisitos. Os requisitos foram divididos em funcionais e não funcionais e são visualizados nas sessões seguintes.

[2]http://httpd.apache.org/
[3]http://www.postgresql.org/
[4]http://www.w3.org/html/
[5]https://developer.mozilla.org/en-US/docs/Web/JavaScript
[6]http://getbootstrap.com/
[7]http://morrisjs.github.io/morris.js/

4.4.1 Requisitos funcionais

A seguir, são listados os principais requisitos funcionais atendidos pela ferramenta desenvolvida:

a) Manter perfil: Através desse requisito a ferramenta disponibiliza uma interface que permite o usuário DBA editar os dados de seu cadastro;

b) Visualizar estatísticas: Através desse requisito o usuário DBA pode visualizar todas as estatísticas coletadas pelo agente de forma gráfica, permitindo uma melhor análise e facilitando a decisão da ação a ser tomada para melhorar o desempenho;

c) Gerenciar estatísticas: Através desse requisito o usuário DBA pode definir as estatísticas que serão monitoradas e inseri-lá ou removê-lá do painel principal;

d) Visualizar alertas: Através desse requisito o usuário DBA pode visualizar os alertas que a ferramenta gerou para que tome as devidas providencias;

e) Monitorar processos em execução: Através desse requisito o usuário DBA pode monitorar os processos que estão em execução no banco de dados. A partir dessa funcionalidade é possível identificar possíveis problemas de performance de um determinado comando que esteja sendo executado.

4.4.2 Requisitos não funcionais

A seguir, são listados os principais requisitos não funcionais atendidos pela ferramenta proposta:

a) Ser desenvolvido na linguagem PHP: Este requisito define qual a linguagem utilizada na escrita do programa. Como a ferramenta desenvolvida é web foi optado por utilizar PHP por ser um linguagem de desenvolvimento web *open source*;

b) Utilizar SGBD PostgreSQL para armazenar as estatísticas coletadas: Este requisito define o sistema gerenciador de banco de dados utilizado para armazenar

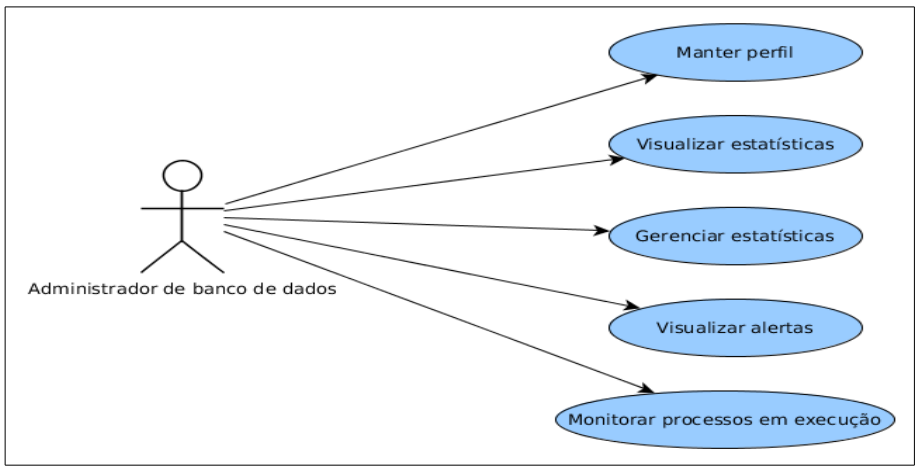
os dados de configurações, usuários e estatísticas coletadas. Foi utilizado PostgreSQL por ser um SGBD robusto *open source*, porém caso o DBA queira utilizar um outro sistema gerenciador de banco de dados deve-se somente reescrever as funções de conexão e adaptar as instruções SQL da ferramenta.

4.4.3 Casos de uso

Os casos de uso de um sistema são a definição de qual requisito cada papel deve executar para que o sistema alcance seus objetivos.

Como pode ser visto na figura 12 no sistema proposto, existe apenas o papel do administrador de banco de dados, que executa todas as tarefas.

Figura 12 - Casos de uso.

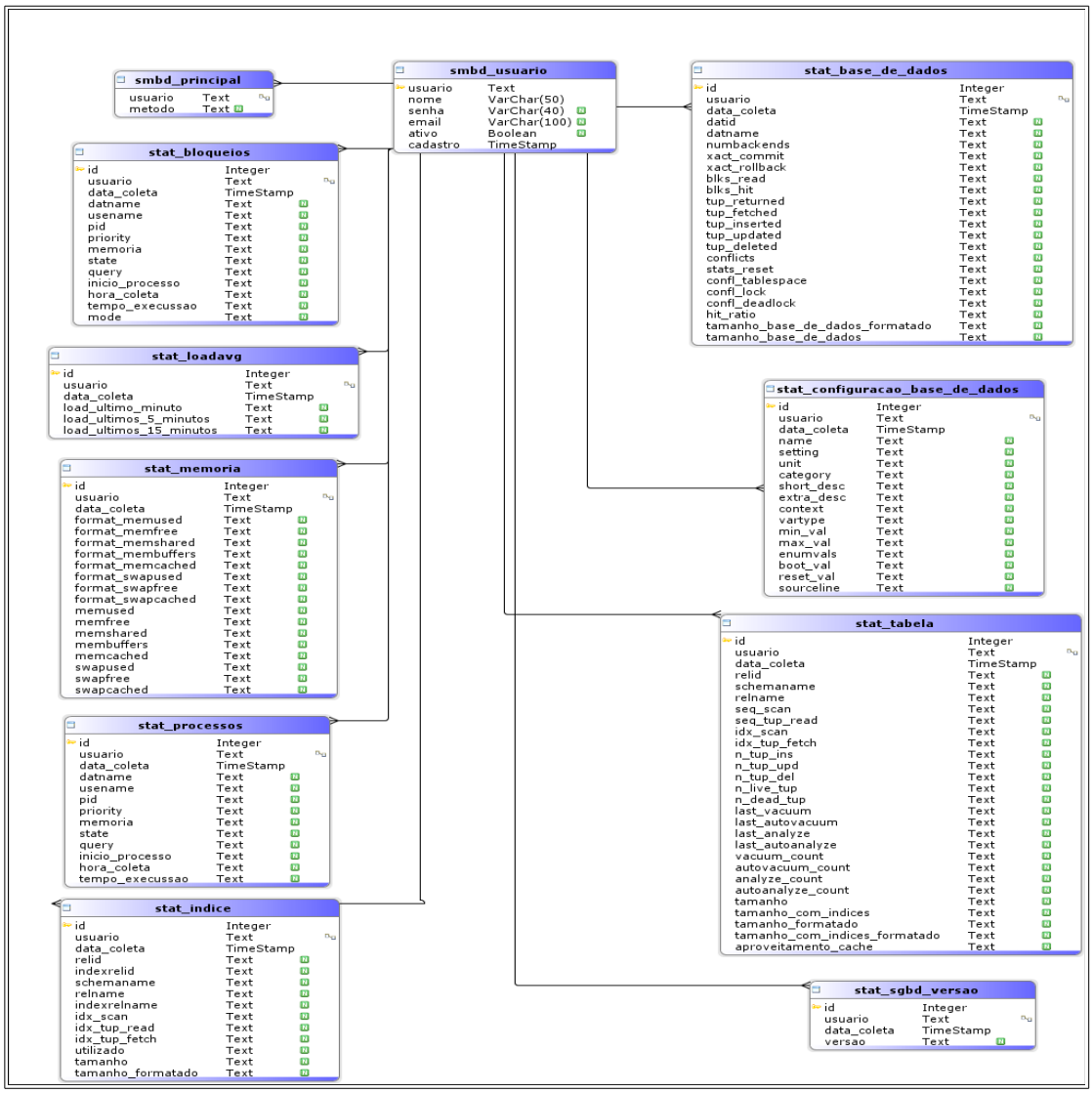


Fonte: Elaborado pelo autor.

4.5 Modelagem do banco de dados

Através dos requisitos levantados foram identificados os elementos necessários para definir o modelo do banco de dados da ferramenta proposta. Este modelo é apresentado na figura 13.

Figura 13 - Modelo do banco de dados.



Fonte: Elaborado pelo autor.

Como mostrado na Figura 13, a estrutura do banco de dados foi modelada em onze tabelas, algumas delas estão detalhadas abaixo.

A tabela *stat_base_de_dados* armazena as estatísticas coletadas relacionadas a informações gerais da base de dados. Nessa tabela pode ser encontrada dados como a data em que o sistema gerenciador de base de dados iniciou a coleta interna de estatísticas. Abaixo podem ser consultadas algumas colunas dessa tabela:

a) *numbackends*: Número de processos em execução nesse banco de dados no momento da coleta;

b) *xact_commit*: Número de transações efetuadas nesse banco de dados, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

c) *xact_rollback*: Número de transações nesse banco de dados que foram revertidas, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

d) *blks_read*: Número de blocos lidos nesse banco de dados, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

e) *blks_hit*: Número de vezes que os blocos lidos foram encontrados no *cache*, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

f) *tup_returned*: Número de linhas retornados por consultas neste banco de dados, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

g) *tup_fetched*: Número de linhas buscadas por consultas neste banco de dados, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

h) *tup_inserted*: Número de linhas inseridas por consultas neste banco de dados, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

i) *tup_updated*: Número de linhas atualizadas por consultas neste banco de dados, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

j) *tup_deleted*: Número de linhas excluídas por consultas neste banco de dados, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

k) *conflicts*: Número de consultas canceladas devido a conflitos neste banco de dados, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

l) *stats_reset*: Data e hora em que as estatísticas internas do PostgreSQL foram ativadas ou resetadas;

m) *confl_tablespace*: Número de consultas neste banco de dados que foram cancelados devido a problemas com *tablespaces,* desde a ativação da coleta de estatísticas nativa do PostgreSQL;

n) *confl_lock*: Número de consultas neste banco de dados que tenham sido cancelada devido a tempo limite para bloqueio, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

o) *confl_deadlock*: Número de consultas neste banco de dados que foram cancelados devido a impasses, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

p) *hit_ratio*: Valor do uso do *cache* formatado em percentual.

Na tabela *stat_processos* são armazenadas informações relacionadas a processos que estavam em execução no momento em que a coleta foi feita. A seguir são detalhadas algumas colunas dessa tabela.

a) *datname*: Nome da base de dados onde o processo está sendo executada;

b) *usename*: Usuário que está executando o processo na base de dados;

c) *pid*: Código identificador do processo no sistema operacional que está sendo executado na base de dados;

d) *priority*: Prioridade do processo no sistema operacional;

e) *memoria*: Memória que o processo está utilizando do sistema operacional;

f) *state*: Estado do processo no sistema operacional;

g) *query*: Comando SQL que o processo está executando no banco de dados;

h) *inicio_processo*: Data e hora que o processo iniciou no banco de dados;

i) *hora_coleta*: Hora em que a estatística foi coletada;

j) *tempo_excussão*: Tempo de execução do processo até o momento da coleta.

A tabela *stat_bloqueios* armazena dados relacionados a bloqueios que estão ocorrendo na base de dados. Essa tabela contém todas as colunas da *stat_processos* com a adição da coluna *mode*, que define o modo de bloqueio diretamente relacionado ao processo em execução. Os modos definidos por PostgreSQL (2015) podem ser consultados abaixo:

a) *ACCESS SHARE*: O comando *SELECT* e o comando *ANALYZE* obtêm um bloqueio neste modo nas tabelas referenciadas. Em geral, qualquer comando que apenas lê a tabela sem modificá-la obtém este modo de bloqueio;

b) *ROW SHARE*: O comando *SELECT FOR UPDATE* obtém o bloqueio neste modo na(s) tabela(s) de destino, além do bloqueio no modo *ACCESS SHARE* para as demais tabelas referenciadas mas não selecionadas através de *UPDATE*;

c) *ROW EXCLUSIVE*: Os comandos U*PDATE, DELETE e INSERT* obtêm este modo de bloqueio na tabela de destino, além do modo de bloqueio *ACCESS SHARE* nas outras tabelas referenciadas. Em geral, este modo de bloqueio é obtido por todos os comandos que alteram os dados da tabela;

d) *SHARE UPDATE EXCLUSIVE* Este modo protege a tabela contra alterações simultâneas no esquema e a execução do comando *VACUUM*;

e) *SHARE*: Este modo protege a tabela contra alterações simultâneas nos dados;

f) *SHARE ROW EXCLUSIVE*: Este modo protege a tabela contra alterações de dados concorrentes e é auto-exclusiva, de modo que apenas uma sessão pode prendê-lo em um momento;

g) *EXCLUSIVE*: Este modo permite apenas bloqueios *ACCESS SHARE* simultâneos, ou seja, somente leituras da tabela podem prosseguir em paralelo com uma transação que obteve este modo de bloqueio;

h) *ACCESS EXCLUSIVE*: Este modo garante que a transação que o obteve é a única acessando a tabela.

Na tabela *stat_configuracao_base_de_dados* são armazenadas as configurações setadas no SGBD da base de dados analisada. Com essas informação é possível manter uma base histórico de cada configuração e consultar facilmente os valores *default* de cada uma delas. A seguir são especificadas algumas colunas dessa tabela:

a) *name*: Nome do parâmetro de configuração;

b) *setting*: Valor do parâmetro no momento em que a coleta foi efetuada;

c) *short_des*c: Uma breve descrição do parâmetro de configuração;

d) *extra_desc*: Descrição adicional mais detalhada do parâmetro de configuração;

e) *context*: Contexto necessário para definir o valor do parâmetro;

f) *vartype*: Tipo do parâmetro *boolean* ou *integer*;

g) *min_val*: Valor mínimo permitido do parâmetro (nulo para valores não numéricos);

h) *max_val*: Valor máximo permitido do parâmetro (nulo para valores não numéricos);

i) *enumvals*: Os valores permitidos para o parâmetro;

j) *boot_val*: O valor do parâmetro assumido na inicialização do servidor;

k) *reset_val*: O valor que o parâmetro irá assumir se for resetado.

A tabela *stat_indice* armazena as estatísticas coletadas relacionadas a índices das tabelas da base de dados, informações como número de varreduras no índice e o seu tamanho podem ser encontradas nessa tabela. Abaixo são especificadas outras informações nela contida:

a) *idx_scan*: Número de varreduras nesse índice, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

b) *idx_tup_read*: Número de linhas retornadas por consultas nesse índice, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

c) *idx_tup_fetch*: Número de linhas ativas retornadas por consultas nesse índice, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

d) *utilizado*: Indica se o índice está sendo utilizado;

e) *tamanho*: Tamanho em *bytes* do índice;

f) *tamanho_formatad*o: Tamanho do índice formatado.

Na tabela *stat_loadavg* são armazenadas informações relacionadas a carga do servidor onde o SGBD está armazenado, ou seja, a média de processos em execução no último minuto, nos últimos cinco minutos e nos últimos quinze minutos.

A tabela *stat_memoria* armazena informações relacionadas a memória da máquina onde o SGBD está armazenado. Abaixo são especificadas algumas colunas dessa tabela.

a) *memused*: Memória usada expressada em *bytes* do servidor onde o SGBD está hospedado;

b) *memfree*: Memória livre expressada em *bytes* do servidor onde o SGBD está hospedado;

c) *memshared*: Memória compartilhada expressada em *bytes* do servidor onde o SGBD está hospedado;

d) *membuffers*: Memória em *buffer* expressada em *bytes* do servidor onde o SGBD está hospedado;

e) *memcached*: Memória em *cache* expressada em *bytes* do servidor onde o SGBD está hospedado;

f) *swapused*: Memória *swap* utilizada expressada em *bytes* do servidor onde o SGBD está hospedado;

g) *swapfree*: Memória *swap* disponível expressada em *bytes* do servidor onde o SGBD está hospedado;

h) *swapcached*: Memória *swap* em *cache* expressada em *bytes* do servidor onde o SGBD está hospedado;

Na tabela *stat_tabela* são armazenadas as estatísticas coletadas de todas as tabelas da base de dados monitorada. A seguir são apresentadas algumas colunas da *stat_tabela*.

a) *seq_scan*: Número de varreduras sequenciais iniciado nessa tabela, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

b) *seq_tup_read*: Número de linhas vivas buscadas por varreduras sequenciais, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

c) *idx_scan*: Número de varreduras de índice iniciado em esta tabela, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

d) *idx_tup_fetch*: Número de linhas vivas buscadas por varreduras de índice nessa tabela, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

e) *n_tup_ins*: Número de linhas inseridas da tabela em questão desde da ativação da coleta de estatísticas nativa do PostgreSQL;

f) *n_tup_upd*: Número de linhas que foram atualizadas nessa tabela, com início de contagem no momento em que as estatísticas nativas do PostgreSQL foram ativadas;

g) *n_tup_del*: Número de linhas que já foram excluídas dessa tabela, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

h) *n_live_tup*: Número de linhas vivas da tabela, ou seja, que estão ativas, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

i) *n_dead_tup*: Número de linhas mortas da tabela, ou seja, que estão marcadas como deletadas na tabela, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

j) *last_vacuum*: Último *vacuum* executado manualmente na tabela;

k) *last_autovacuum*: Último *autovacuum* executado na tabela;

l) *last_analyze*: Último *vacuum analyze* executado manualmente na tabela;

m) *last_autoanalyze*: Último *vacuum analyze* executado automaticamente na tabela;

n) *vacuum_count*: Número de vezes que o *vacuum* foi executado manualmente na tabela, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

o) *autovacuum_count*: Número de vezes que o *vacuum* foi executado automaticamente na tabela, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

p) *analyze_count*: Número de vezes que o *vacuum analyze* foi executado manualmente na tabela, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

q) *autoanalyze_count*: Número de vezes que o *vacuum analyze* foi executado automaticamente na tabela, desde a ativação da coleta de estatísticas nativa do PostgreSQL;

r) *tamanho*: Tamanho da tabela na base de dados em bytes;

s) *tamanho_com_indices*: Tamanho da tabela em bytes com a adição do tamanho de seus índices;

t) aproveitamento_cache: Valor do uso do *cache* formatado em percentual.

No próximo capítulo está descrito em detalhes a ferramenta desenvolvida e os resultados obtidos com ela. Por fim, avalia o SMBD em um ambiente em produção.

5 RESULTADOS E DISCUSSÃO

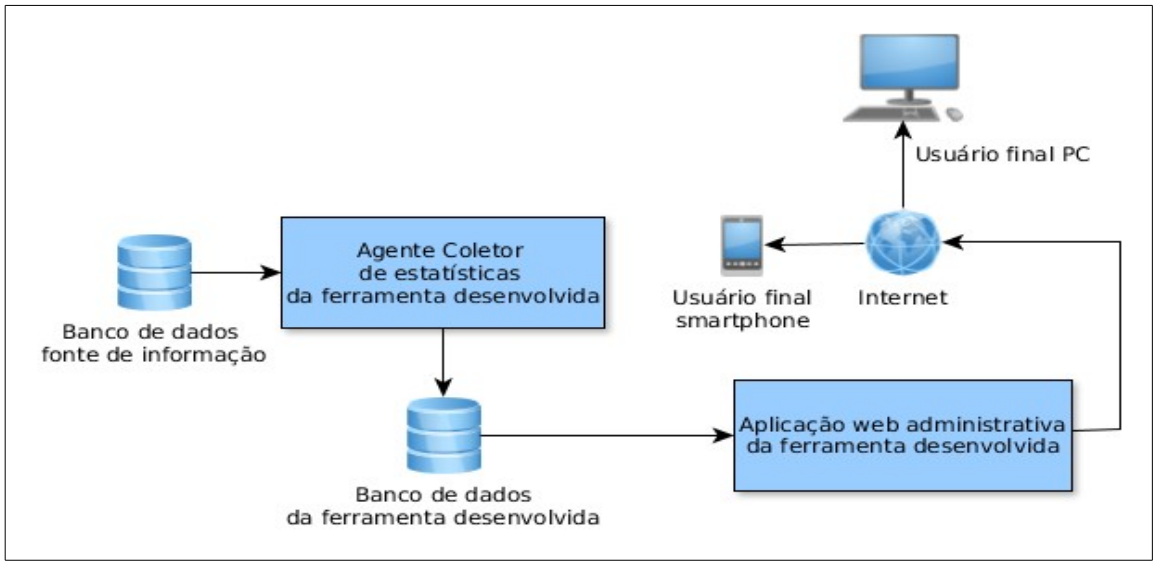
De acordo com a importância do monitoramento de um sistema gerenciador de banco de dados e análise de estatísticas para garantir a otimização de consultas, é evidenciada a real necessidade de se ter ferramentas que auxiliam na execução desse trabalho. Atualmente, são encontrados poucos softwares que oferecem de maneira gráfica esse tipo de informação. Algumas ferramentas atingem o objetivo, mas acabam sendo descartadas por serem pagas, descontinuadas, não suportar mais de um SGBD ou simplesmente não proporcionam o que se deseja de maneira clara.

Tendo como base a realidade apresentada, a ferramenta desenvolvida proporciona ao DBA uma ferramenta gratuita que possibilita o monitoramento e análise de estatísticas de maneira gráfica e de fácil manuseio.

A utilização da mesma é simples e permite identificar facilmente, por exemplo, instruções com execução lenta, índices não utilizados e o percentual que as tabelas representam do tamanho total de uma base de dados.

Através da figura 14 pode ser visualizada a arquitetura da ferramenta desenvolvida, onde a mesma conta com um agente coletor que se comunica com a base de dados monitorada e o banco de dados responsável por armazenar as estatísticas. A estrutura desenvolvida possibilita que seja monitorado um outro SGBD, basta que seja implementado o módulo coletor específico para o sistema gerenciador desejado.

Figura 14 - Arquitetura da ferramenta desenvolvida.



Fonte: Elaborado pelo autor.

Nas próximas seções são descritos em detalhes o funcionamento do agente coletor e da aplicação web desenvolvida.

5.1.1 Agente coletor para SGBD PostgreSQL

O agente é a parte da ferramenta responsável pela coleta das estatísticas da base de dados que está sendo monitorada, através dessa rotina que a base de dados do SMBD vai ser populada.

A rotina desenvolvida utiliza a linguagem de programação PHP e se comunica com o SMBD através do protocolo SOAP (*Simple Object Access Protocol*). O agente não precisa necessariamente estar no mesma máquina que o SGBD se encontra, basta haver comunicação entre os dois pontos.

O agente desenvolvido pode ser utilizado apenas para base de dados que utilizam SGBD PostgreSQL e sistema operacional Linux.

Na próxima seção é exemplificada a configuração do agente coletor.

5.1.1.1 Configuração do agente

Para que a coleta seja feita de maneira moderada sem onerar o processamento do servidor onde o SGBD está hospedado, o agente conta com uma série de configurações que podem ser visualizadas na figura 15.

Para cada tipo de coleta pode ser configurado o tempo, em minutos, que a próxima coleta será feita. Dessa maneira o DBA pode avaliar e configurar a ferramenta de acordo com a necessidade.

Figura 15 - Configurações do agente coletor.

```
5    $config = array(
6        //base de dados
7        'host' => 'localhost',
8        'port' => '5432',
9        'user' => 'postgres',
10       'password' => 'postgres',
11       'dbname' => 'smbd',
12       //smbd
13       'url' => 'http://smbd.com.br',
14       'usuario' => 'ftomasini.rs@gmail.com',
15       //intervalo de coleta
16       'tempo_coleta_sgbd_versao' => '200 minutes',
17       'tempo_coleta_base_de_dados' => '200 minutes',
18       'tempo_coleta_tabela' => '200 minutes',
19       'tempo_coleta_indice' => '200 minutes',
20       'tempo_coleta_configuracoes' => '200 minutes',
21       'tempo_coleta_loadavg' => '1 minute',
22       'tempo_coleta_memoria' => '1 minute',
23       'tempo_coleta_processos' => '1 minute',
24       'tempo_coleta_bloqueios' => '200 minutes',
25       );
```

Fonte: Elaborado pelo autor.

Na próxima seção, pode ser visualizado os métodos utilizados pelo coletor para obter os dados que serão enviados para a aplicação web desenvolvida. Os demais métodos e classes utilizados pelo coletor podem ser encontradas no apêndice A do presente trabalho.

5.1.1.2 Consultas executadas pelo agente

Através das consultas, a seguir, o agente coletor obtém os dados que serão enviados para o SMBD. Para cada função detalhada abaixo é configurado o tempo de coleta conforme descrito no seção anterior.

Mediante a consulta exibida na figura 16 o agente coletor obtém informações gerais da base de dados monitorada, como o tamanho da base de dados e o percentual do uso do *cache*.

Para obter as informações gerais da base de dados foi utilizada a visão *pg_stat_database,* disponibilizada pelo PostgreSQL para extrair estatísticas de todo o banco de dados.

Figura 16 - Consulta que obtém informações da base de dados.

```
26  SELECT '{$coletaAtual}' as data_coleta,
27         '{$this->usuario}' as usuario,
28         pg_stat_database.datid,
29         pg_stat_database.datname,
30         pg_stat_database.numbackends,
31         pg_stat_database.xact_commit,
32         pg_stat_database.xact_rollback,
33         pg_stat_database.blks_read,
34         pg_stat_database.blks_hit,
35         pg_stat_database.tup_returned,
36         pg_stat_database.tup_fetched,
37         pg_stat_database.tup_inserted,
38         pg_stat_database.tup_updated,
39         pg_stat_database.tup_deleted,
40         pg_stat_database.conflicts,
41         date_trunc('seconds', pg_stat_database.stats_reset) AS stats_reset,
42         --conflitos
43         pg_stat_database_conflicts.confl_tablespace,
44         pg_stat_database_conflicts.confl_lock,
45         pg_stat_database_conflicts.confl_snapshot,
46         pg_stat_database_conflicts.confl_bufferpin,
47         pg_stat_database_conflicts.confl_deadlock,
48         --cache
49         (SELECT round(AVG(
50                 (heap_blks_hit + COALESCE(toast_blks_hit, 0))::numeric /
51                 (heap_blks_read + COALESCE(toast_blks_read, 0) +
52                  heap_blks_hit + COALESCE(toast_blks_hit,0))::numeric * 100), 3) as hit_ratio
53            FROM pg_statio_user_tables
54           WHERE (heap_blks_read + COALESCE(toast_blks_read, 0)) + heap_blks_hit +
    COALESCE(toast_blks_hit) > 0) as hit_ratio,
55         pg_size_pretty(pg_database_size(pg_stat_database.datname)) as
    tamanho_base_de_dados_formatado,
56         pg_database_size(pg_stat_database.datname) as tamanho_base_de_dados
57       FROM pg_stat_database
58  INNER JOIN pg_stat_database_conflicts
59          ON pg_stat_database.datname = pg_stat_database_conflicts.datname
60       WHERE pg_stat_database.datname= '$bdNome'
```

Fonte: Elaborado pelo autor.

Através da consulta exibida na figura 17 o agente coletor obtém informações das tabelas da base de dados que está sendo monitorada, como pode ser visto foi utilizada a visão *pg_stat_user_tables* que fornece informações como a quantidade de varreduras e a data do último *vacuum* executado, de todas as tabelas criadas pelo usuário, desconsiderando as tabelas do catálogo do PostgreSQL.

Figura 17 - Consulta que obtém informações das tabelas da base de dados.

```
61   SELECT '{$coletaAtual}' as data_coleta,
62          '{$this->usuario}' as usuario,
63          relid,
64          schemaname,
65          relname,
66          seq_scan,
67          seq_tup_read,
68          idx_scan,
69          idx_tup_fetch,
70          n_tup_ins,
71          n_tup_upd,
72          n_tup_del,
73          n_tup_hot_upd,
74          n_live_tup,
75          n_dead_tup,
76          last_vacuum,
77          last_autovacuum,
78          last_analyze,
79          last_autoanalyze,
80          vacuum_count,
81          autovacuum_count,
82          analyze_count,
83          autoanalyze_count,
84          pg_relation_size(relid) as tamanho,
85          pg_total_relation_size(relid)as tamanho_com_indices,
86          pg_size_pretty(pg_relation_size(relid)) as tamanho_formatado,
87          pg_size_pretty(pg_total_relation_size(relid)) as tamanho_com_indices_formatado,
88          (SELECT round((heap_blks_hit + COALESCE(toast_blks_hit, 0))::numeric /
89                        (heap_blks_read + COALESCE(toast_blks_read, 0) +
90                         heap_blks_hit + COALESCE(toast_blks_hit,0))::numeric * 100, 3) as hit_ratio
91             FROM pg_statio_user_tables
92            WHERE pg_stat_user_tables.relname = pg_statio_user_tables.relname
93              AND pg_stat_user_tables.schemaname = pg_statio_user_tables.schemaname
94              AND pg_statio_user_tables.schemaname = 'public'
95              AND (heap_blks_read + COALESCE(toast_blks_read, 0)) + heap_blks_hit +
     COALESCE(toast_blks_hit) > 0) as aproveitamento_cache
96     FROM pg_stat_user_tables
```

Fonte: Elaborado pelo autor.

Por meio da consulta da figura 18 o agente coletor extrai informações sobre os índices criados pelo usuário nas tabelas da base de dados que está sendo monitorada. Como pode ser visualizado, foi utilizada a visão p*g_stat_user_indexes* que fornece todas as informações relacionadas aos índices criados pelo usuário.

Figura 18 - Consulta que obtém informações sobre índices das tabelas da base de dados.

```
97   SELECT '{$coletaAtual}' as data_coleta,
98          '{$this->usuario}' as usuario,
99          relid,
100         indexrelid,
101         schemaname,
102         relname,
103         indexrelname,
104         idx_scan,
105         idx_tup_read,
106         idx_tup_fetch,
107         (idx_scan > 0) as utilizado,
108         pg_relation_size(indexrelid) as tamanho,
109         pg_total_relation_size(relid)as tamanho_com_indices,
110         pg_size_pretty(pg_relation_size(indexrelid)) as tamanho_formatado,
111         pg_size_pretty(pg_total_relation_size(relid)) as tamanho_com_indices_formatado
112    FROM pg_stat_user_indexes
```

Fonte: Elaborado pelo autor.

Através da consulta exibida na figura 19 o agente coletor obtém as configurações do SGBD da base de dados que está sendo monitorada. Por meio da visão *pg_setting*, foi

possível obter todas informações disponíveis no arquivo *postgresql.conf*, responsável pela configuração do sistema gerenciador de banco de dados.

Figura 19 - Consulta que obtém as configurações do SGBD da base de dados monitorada.

```
113  SELECT '{$coletaAtual}' as data_coleta,
114         '{$this->usuario}' as usuario,
115         name,
116         setting,
117         unit,
118         category,
119         short_desc,
120         extra_desc,
121         context,
122         vartype,
123         source,
124         min_val,
125         max_val,
126         enumvals,
127         boot_val,
128         reset_val,
129         sourcefile,
130         sourceline
131    FROM pg_settings
```

Fonte: Elaborado pelo autor.

Mediante a consulta exibida na figura 20 o agente coletor obtém dados relacionados a informações da carga do servidor onde o SGBD da base monitorada está hospedado. Para extrair essa informação foi utilizada a visão *pg_loadavg* disponibilizada pela extensão *pg_proctab.*

Figura 20 - Consulta que obtém informações sobre a carga do servidor do SGBD.

```
132  SELECT '{$coletaAtual}' as data_coleta,
133         '{$this->usuario}' as usuario,
134         load1 as load_ultimo_minuto,
135         load5 as load_ultimos_5_minutos,
136         load15 as load_ultimos_15_minutos
137    FROM pg_loadavg();
```

Fonte: Elaborado pelo autor.

Por meio da consulta exibida na figura 21 o agente coletor obtém informações da memória utilizada pelo servidor do SGBD da base de dados que está sendo monitorada. Para extrair essas informações foi utilizada a visão *pg_memusage* disponibilizada pela extensão *pg_proctab.*

Figura 21 - Consulta que obtém informações sobre a memória do servidor do SGBD.

```
138  SELECT '{$coletaAtual}' as data_coleta,
139         '{$this->usuario}' as usuario,
140         pg_size_pretty(memused * 1024) as format_memused,
141         pg_size_pretty(memfree * 1024) as format_memfree,
142         pg_size_pretty(memshared * 1024) as format_memshared,
143         pg_size_pretty(membuffers * 1024) as format_membuffers,
144         pg_size_pretty(memcached * 1024) as format_memcached,
145         pg_size_pretty(swapused * 1024) as format_swapused,
146         pg_size_pretty(swapfree * 1024) as format_swapfree,
147         pg_size_pretty(swapcached * 1024) as format_swapcached,
148         memused,
149         memfree,
150         memshared,
151         membuffers,
152         memcached,
153         swapused,
154         swapfree,
155         swapcached
156    FROM pg_memusage();
157
```

Fonte: Elaborado pelo autor.

Através da consulta da figura 22 o agente coletor obtém informações sobre os processos que estão em execução na base de dados que está sendo monitorada. Por meio da visão *pg_stat_activity* e da extensão *pg_proctab* foi possível obter informações como o código do processo em execução e sua *query*.

Figura 22 - Consulta que obtém os processos que estão em execução na base de dados.

```
158     SELECT '{$coletaAtual}' as data_coleta,
159            '{$this->usuario}' as usuario,
160             pg_stat_activity.datname,
161             pg_stat_activity.usename,
162             pg_stat_activity.pid,
163             priority,
164             pg_size_pretty(B.rss * 1024) as memoria,
165             B.state,
166             pg_stat_activity.query,
167             'PROC_SMBD_COLETOR' as identificador,
168             date_trunc('seconds', pg_stat_activity.backend_start) AS inicio_processo,
169             date_trunc('seconds', now()) AS hora_coleta,
170             date_trunc('seconds',SUM(now() - pg_stat_activity.backend_start)) as tempo_execussao
171       FROM pg_stat_activity
172  INNER JOIN pg_proctab() B
173          ON pg_stat_activity.pid = B.pid
174         AND pg_stat_activity.query not ilike '%PROC_SMBD_COLETOR%'
175       WHERE datname = '$bdNome'
176    GROUP BY 1,2,3,4,5,6,7,8,9,10,11,12;
177
```

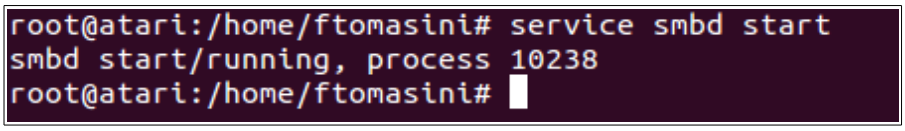
Fonte: Elaborado pelo autor.

5.1.1.3 Ativando o coletor

Para maior controle e garantia da coleta de dados o módulo coletor é inserido no sistema operacional como um serviço. Na figura 23 é possível visualizar a ativação da coleta em um servidor.

Dessa maneira se o servidor for reiniciado, o serviço de coleta irá se reestabelecer automaticamente, evitando possíveis distorções na análise decorrente de intervalos de tempo onde os dados não foram coletados.

Figura 23 - Ativação do módulo coletor.



Fonte: Elaborado pelo autor.

Na próxima seção são abordado a aplicação web administrativa da ferramenta desenvolvida.

5.1.2 Aplicação web administrativa da ferramenta desenvolvida

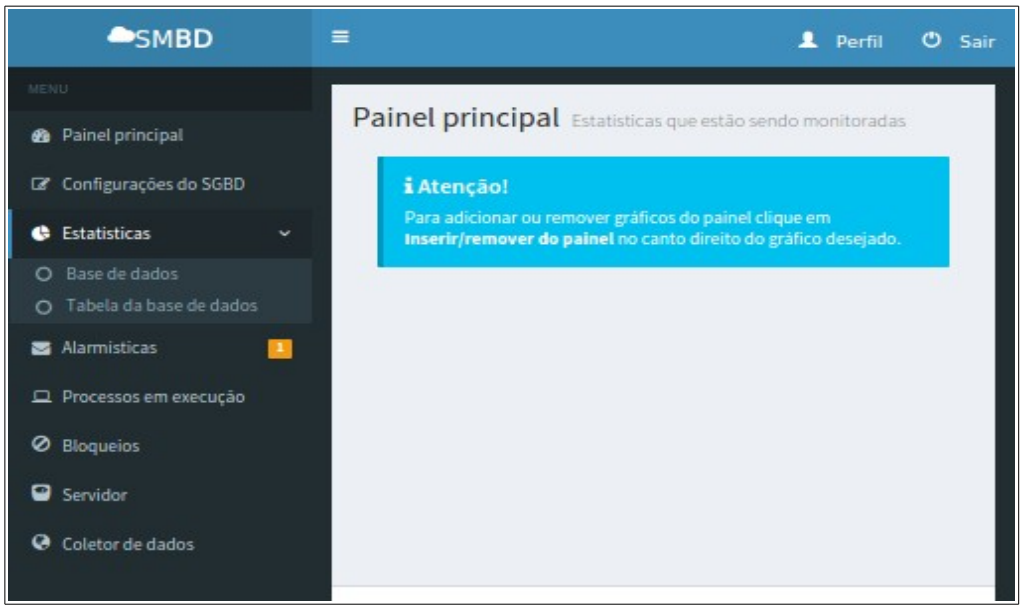
Depois ou durante a extração de dados o resultado pode ser consultado através de uma ferramenta onde as informações são integradas. Ainda, por meio dela o DBA pode identificar, de forma visual, possíveis problemas de desempenho em relação ao SGBD, podendo assim, tornar visível a necessidade de criação de índices e alterações em configurações do sistema utilizado. A ferramenta facilita a otimização e o monitoramento do sistema gerenciador de banco de dados.

É importante ressaltar que a mesma ferramenta pode ser utilizada para o monitoramento de outros SGBDs, basta que seja implementado o módulo coletor específico para o sistema gerenciador escolhido. Este recurso não é encontrado nas ferramentas estudadas.

Na ferramenta as informações estão agrupadas em itens de menu e podem ser visualizadas em conjunto através do painel principal, basta clicar no botão *inserir/remover do painel* no *dashboard* escolhido.

Para um melhor entendimento é apresentada a tela inicial da ferramenta e uma breve explicação de cada item do menu disponibilizado. Os painéis implementados são visualizados na próxima seção, onde a ferramenta é avaliada em um ambiente de produção.

Figura 24 - Tela principal do SMBD.



Fonte: Elaborado pelo autor.

Com base na figura 24 segue abaixo uma breve explicação de cada item de menu da ferramenta:

a) Painel principal: A partir dessa interface o DBA pode visualizar as estatísticas que estão sendo monitoradas, para monitorar uma estatística o usuário deve clicar em publicar no canto superior direito do *dashbord* desejado;

b) Configurações do SGBD: A partir dessa interface o DBA pode visualizar as configurações do SGBD da base de dados monitorada;

c) Estatísticas: Esse item de menu possui duas opções, base de dados e tabela da base de dados. Para visualizar os *dashbords* relacionados a base de dados deve ser selecionada a primeira opção, se o DBA desejar visualizar alguma informação específica de uma tabela deve ser selecionada a segunda opção;

d) Alarmísticas: Por meio dessa interface o usuário pode visualizar os alertas identificados pela ferramenta;

e) Menu processos em execução: Interface que possibilita visualizar todas as tarefas que estão sendo executadas no banco de dados monitorado, para uma melhor eficácia dessa funcionalidade o tempo de coleta dessa estatística deve ser pequena;

f) Bloqueios: Através dessa funcionalidade o DBA pode visualizar todos os processos que estão efetuando bloqueios na base de dados;

g) Servidor: Por meio dessa interface o usuário poderá visualizar os gráficos de uso de memória e carga do servidor do SGBD do banco de dados monitorado;

h) Coletor de dados: Através dessa ação o DBA pode baixar o módulo coletor para instalar em um servidor de bando de dados que deseja monitorar.

5.2 Avaliação da ferramenta em um ambiente de produção

Esta avaliação tem por objetivo verificar o comportamento do SMBD em um ambiente real e detalhar os recursos da ferramenta. O ambiente é composto por um servidor de base de dados que utiliza o sistema operacional Linux (Ubuntu Server 12.04 LTS) e SGBD PostgreSQL 9.4.

A base de dados monitorada armazena informações de um software para gestão de ensino superior que contém cerca de 806 tabelas. A instituição utiliza o sistema desde 2007, portanto, contém um volume considerável de dados e já sofre com alguns problemas de performance em processos que envolvem tabelas com maior quantidade de registros.

Com o objetivo de facilitar a configuração de acesso a base de dados monitorada o agente coletor foi instalado na mesma máquina que o SGBD se encontra, e após a instalação o mesmo foi colocado em execução em um tempo total de 24 horas onde foram feitas coletas de todas estatísticas previstas pelo coletor.

Para se ter um bom acompanhamento dos processos executados na base de dados, bloqueios, carga e uso de memória do servidor, foram feitas coletas no intervalo de um minuto, já para informações que não variam tanto como informações da base de dados, estatísticas de índices, tabelas, versão do servidor e configurações do SGBD as coletas tiveram um intervalo configurado de duzentos minutos.

Com a configuração apresentada foi possível obter um bom volume de dados e identificar alguns problemas da base monitorada. Nas próximas seções são apresentados dados relativos a estas coletas e os recursos disponibilizados pela ferramenta através de um

ambiente real. Dentre os recursos são vistos os *dashboards* disponibilizados em cada item de menu da ferramenta.

5.2.1 Informações da base de dados

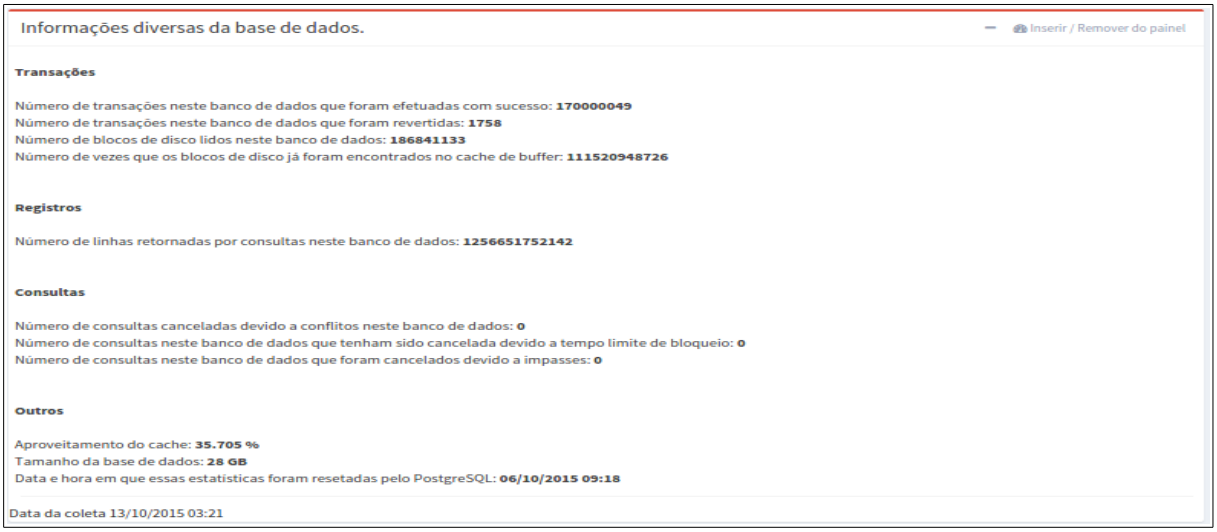
A interface apresentado na figura 25 apresenta ao usuário DBA informações gerais da base de dados monitorada e está disponível na ferramenta através do menu estatísticas da base de dados.

Com base nas informações coletadas com essa funcionalidade, foi possível identificar que as coletas de estatísticas foram iniciadas pelo SGBD na data 06/10/2015, consequentemente todas as informações disponíveis nesse painel e nos demais que são apresentados nas próximas seções foram iniciadas nessa data.

Nesse painel foi possível também identificar que a base de dados monitorada tem um tamanho total de 28 GB.

Por meio desse mesmo painel o usuário DBA pode consultar informações relacionada ao número de transações efetivadas no banco de dados, número de conflitos e o percentual do uso do *cache*.

Figura 25 - Informações da base de dados.



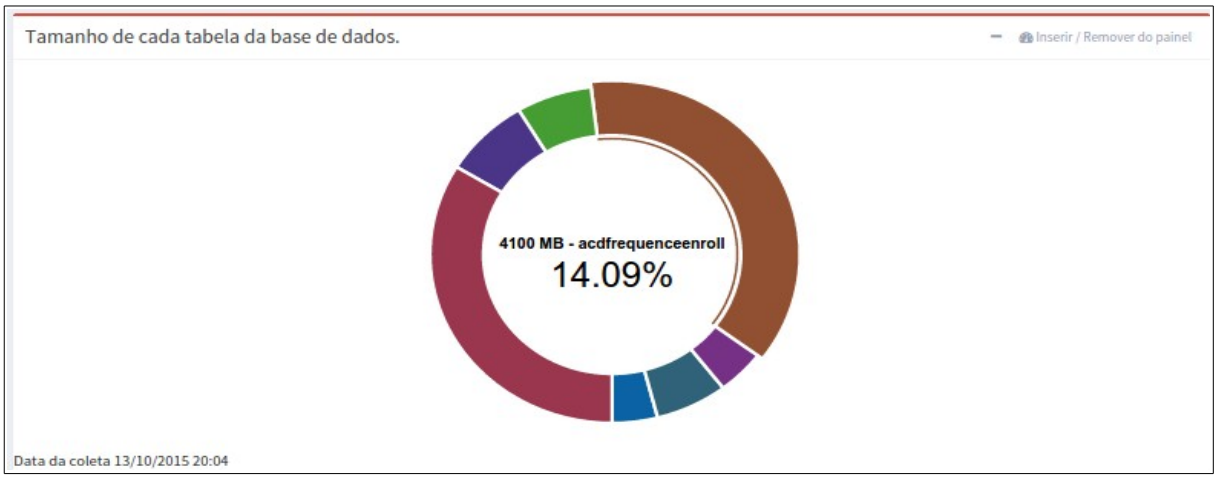
Fonte: Elaborado pelo autor.

5.2.2 Tamanho da base de dados

No painel que pode ser visualizado na figura 26 é exibida a distribuição do tamanho da base de dados em tabelas sem considerar seus índices. Com base nessa informação foi possível identificar que 14.09% da base de dados é ocupada por dados de apenas uma tabela *acdfrequenceenroll* que é responsável por armazenar a frequência dos alunos nas disciplinas oferecidas pela instituição. A outra grande fatia representa 12.78% da base de dados e armazena um acumulado de dados de tabelas com menos de 1% do espaço total.

Mediante essa informação o DBA pode acompanhar o aumento do volume de dados e medir a proporção de uma determinada tabela em relação as demais, podendo assim tomar algumas providencias antes que isso se torne um problema de desempenho ou disponibilidade do serviço.

Figura 26 - Tamanho da base de dados.



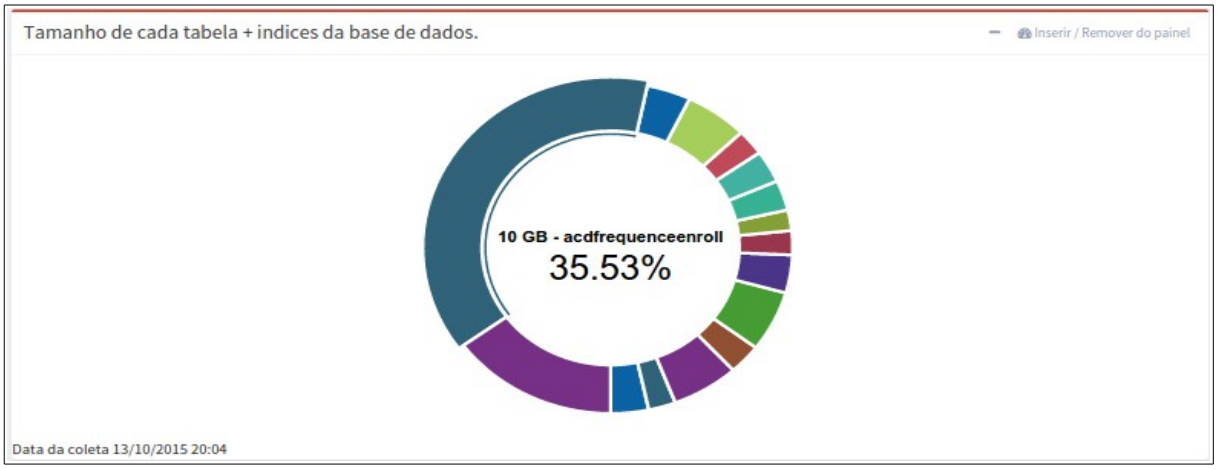
Fonte: Elaborado pelo autor.

5.2.3 Tamanho da base de dados com índices

No *dashboard* apresentado na figura 27 é exibida a distribuição do tamanho da base de dados em tabelas considerando seus índices, e da mesma forma que o gráfico apresentado anteriormente é visível que a maior fatia do espaço continua sendo ocupado pela mesma tabela *acdfrequenceenroll*, porém agora representa 35% do tamanho total e 10GB do total de 28GB.

Por meio dessa informação o DBA pode visualmente identificar tabelas que os índices estão ocupando um elevado volume do espaço e avaliar a remoção dos que não estão sendo utilizado.

Figura 27 - Tamanho da base de dados com índices.



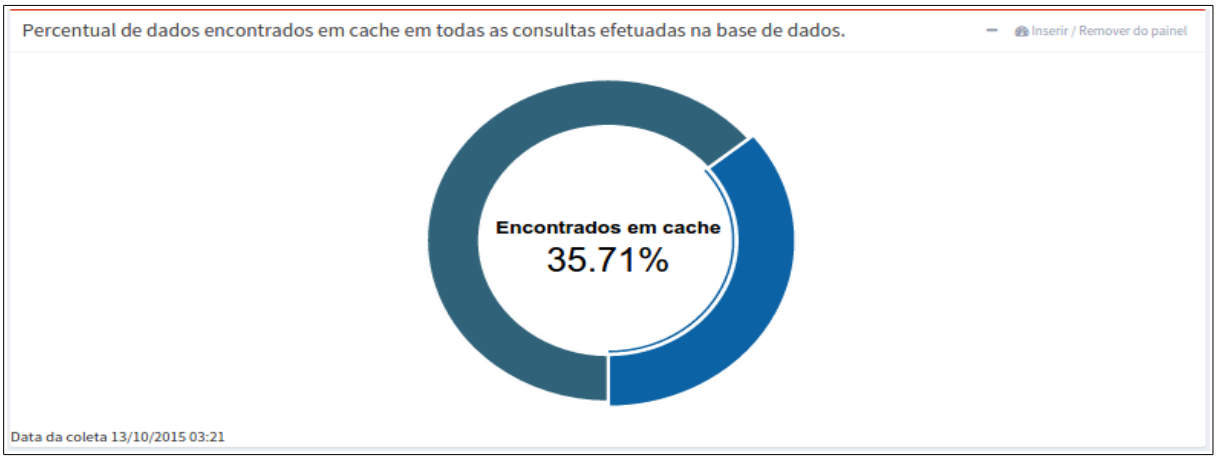
Fonte: Elaborado pelo autor.

5.2.4 Uso do cache

Com base no gráfico apresentado na figura 28 é possível visualizar que 35.71% dos dados buscados por consultas no banco de dados analisado, foram encontradas diretamente no cache de memória reservado para o SGBD.

Com essa informação o administrador do banco de dados pode calibrar o parâmetro de configuração *shared_buffers* responsável por definir o tamanho em *bytes* da memória disponibilizada especificamente para as operações ativas no banco de dados.

Figura 28 -Uso do cache.



Fonte: Elaborado pelo autor.

5.2.5 Configurações da base de dados

O SMBD permite que o DBA consulte as configurações setadas no sistema gerenciador de base de dados. Na figura 29 é possível verificar que a memória reservada para o banco de dados foi alterada de 1024 para 262144. Dessa maneira o PostgreSQL consegue armazenar mais dados em cache e economizar tempo com consultas. Este parâmetro influência diretamente no gráfico apresentado na seção anterior figura 28, pois quanto maior a memória reservada mais dados poderão ficar em cache.

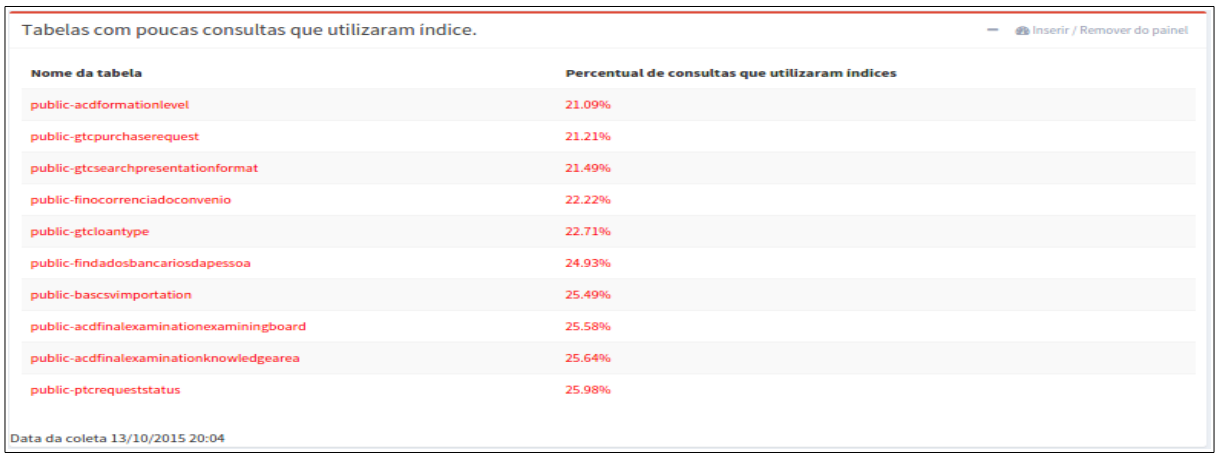
Figura 29 - Configurações da base de dados

Fonte: Elaborado pelo autor.

5.2.6 Tabelas com poucas pesquisas que utilizaram índices

Com base nas informações apresentadas na figura 30 o usuário DBA pode visualizar tabelas que são fortes candidatas a receberem algum índice em alguma de suas colunas, na situação das tabelas da base de dados analisada é possível verificar que na melhor situação apenas 25,98% das instruções executadas consultaram algum índice da tabela.

Figura 30 - Tabelas com poucas pesquisas que utilizaram índices.



Tabelas com poucas consultas que utilizaram índice.

Inserir / Remover do painel

Nome da tabela	Percentual de consultas que utilizaram índices
public-acdformationlevel	21.09%
public-gtcpurchaserequest	21.21%
public-gtcsearchpresentationformat	21.49%
public-finocorrenciadoconvenio	22.22%
public-gtcloantype	22.71%
public-findadosbancariosdapessoa	24.93%
public-bascsvimportation	25.49%
public-acdfinalexaminationexaminingboard	25.58%
public-acdfinalexaminationknowledgearea	25.64%
public-ptcrequeststatus	25.98%

Data da coleta 13/10/2015 20:04

Fonte: Elaborado pelo autor.

5.2.7 Instruções mais lentas

De acordo com o *dashboard* apresentado na figura 31 é possível identificar as instruções mais lentas executadas no último dia de coleta. Na situação da base de dados analisada foi possível perceber que foi executado VACUUM FULL ANALYZE e teve duração de 46:17 minutos, além desse comando foi possível identificar outros que tiveram uma duração elevada.

Essa informação é importante para o administrador do banco de dados dar uma atenção especial a estes comandos, ou seja, tentar otimizá-los ou refatorá-los, tomando essa ação pode se ter um ganho considerável no desempenho do sistema.

Figura 31 - Comandos mais lentos.

Comandos mais lentos do dia.

Inserir / Remover do painel

Comando	Tempo de execução
VACUUM FULL VERBOSE ANALYZE	00:46:17
SELECT array_agg(UPPER(replace(convert_from(convert_to(replace(replace(replace(replace(replace(X.complementarybibliography, 'Â', ''), 'Â', ''), 'Â§ÃM', 'XX'), 'A \u0080\u0093', '!')), 'ÃM', 'ÔE'), 'LATIN1'), 'UTF8'), 'XX', 'ÇÔ'))) AS complementarybibliography FROM (SELECT acdGroup.groupId, UNNEST(complementarybibliography) AS complementarybibliography FROM acdGroup INNER JOIN acdLearningPeriod ON acdLearningPeriod.learningPeriodId = acdGroup.learningPeriodId WHERE acdLearningPeriod.periodId = '2015/2' AND array_to_string(complementarybibliography, ',') <> '' AND groupid = 27269 ORDER BY groupId) X;	00:29:28
SELECT acdGroup.groupId, UNNEST(complementarybibliography) AS complementarybibliography FROM acdGroup INNER JOIN acdLearningPeriod ON acdLearningPeriod.learningPeriodId = acdGroup.learningPeriodId WHERE acdLearningPeriod.periodId = '2015/2' AND array_to_string(complementarybibliography, ',') <> '' AND groupid = 27269 ORDER BY groupId ;	00:27:22
SELECT acdGroup.groupId, UNNEST(complementarybibliography) AS complementarybibliography FROM acdGroup INNER JOIN acdLearningPeriod ON acdLearningPeriod.learningPeriodId = acdGroup.learningPeriodId WHERE acdLearningPeriod.periodId = '2015/2' AND array_to_string(complementarybibliography, ',') <> '' AND groupid = 27269 ORDER BY groupId limit 1 offset 28;	00:24:13
INSERT INTO bi_rematricula SELECT COALESCE(SUM(X.matriculados), 0) AS matriculados, COALESCE(SUM(X.renovacoes), 0) AS renovacoes , COALESCE(SUM(X.reingressos), 0) AS reingressos, COALESCE(SUM(X.transferencia_de), 0) AS transferencia_de, COALESCE(SUM(X.ndas_matriculados), 0) AS ndas_matriculados, COALESCE(SUM(X.portadores_diploma), 0) AS portadores_diploma, COALESCE(SUM(X.nao_renovados), 0) AS nao_renovados, COALESCE(SUM(X.prematriculados), 0) AS pre_matriculados FROM (SELECT codigo_curso AS courseid, nome_curso AS NAME, codigo_turno AS turnid, descricao_turno AS description, (CASE WHEN situacao_no_periodo = 'Pré-matriculado' THEN COUNT(codigo_contrato) ELSE 0 END) AS prematriculados, (CASE WHEN situacao_no_periodo = 'Portador de diploma' THEN COUNT(codigo_contrato) ELSE 0 END) AS portadores_diploma, (CASE WHEN situacao_no_periodo = 'Vestibulando' THEN COUNT(codigo_contrato) ELSE 0 END) AS matriculados, (CASE WHEN situacao_no_periodo = 'Renovado' THEN COUNT(codigo_contrato)	00:21:31

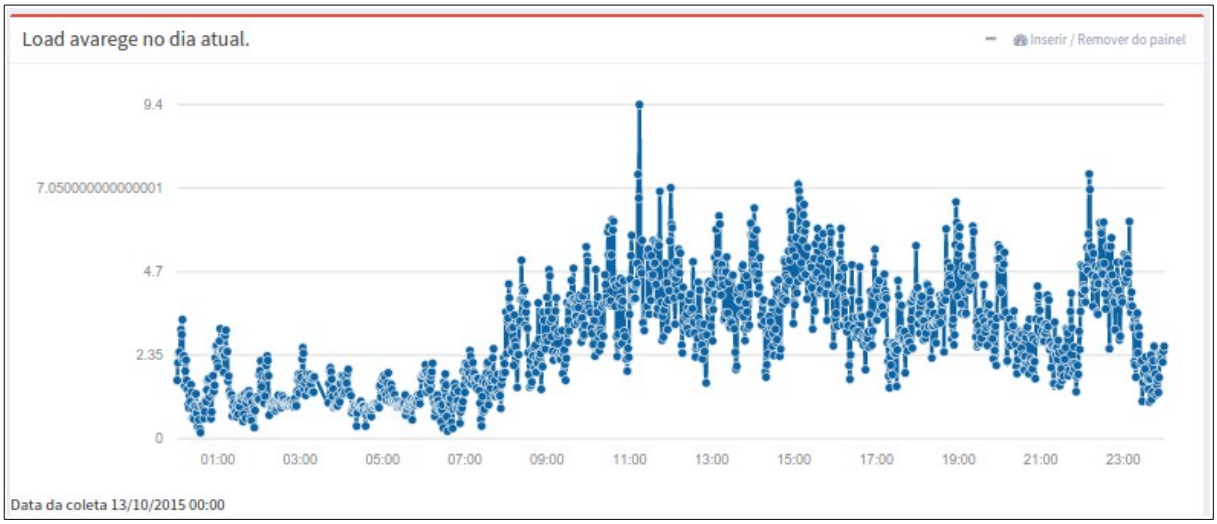
Data da coleta 13/10/2015

Fonte: Elaborado pelo autor.

5.2.8 Carga do servidor

Na interface apresentada na figura 32 pode ser visualizada a carga do servidor onde o SGBD está hospedado. A informação representa a média de processos em execução no instante da coleta. Na situação do servidor monitorado a coleta teve um intervalo configurado de um minuto, e pode ser visualizado alguns picos em alguns horários. Por meio dessa informação o DBA pode identificar o melhor horário para efetuar uma manutenção no SGBD ou no banco de dados.

Figura 32 - Carga do servidor.



Fonte: Elaborado pelo autor.

5.2.9 Memória do servidor

Outra atividade relevante para se ter um monitoramento eficiente é o acompanhamento da utilização da memória do servidor onde o banco de dados está hospedado, através dessa informação é possível identificar se a maquina está fazendo cache em disco tornando o serviço lento.

No painel que pode ser visualizado na figura 33 o usuário DBA pode acompanhar o uso da memória do servidor onde o SGBD está hospedado. No servidor da base de dados monitorada foram feitas coletas a cada um minuto e foi possível perceber que o uso da memória se manteve constante por boa parte do tempo, e é importante salientar que em nenhum momento foi utilizado 100% da memória dispensando o uso do disco (memória swap).

Figura 33 - Memória do servidor.

Fonte: Elaborado pelo autor.

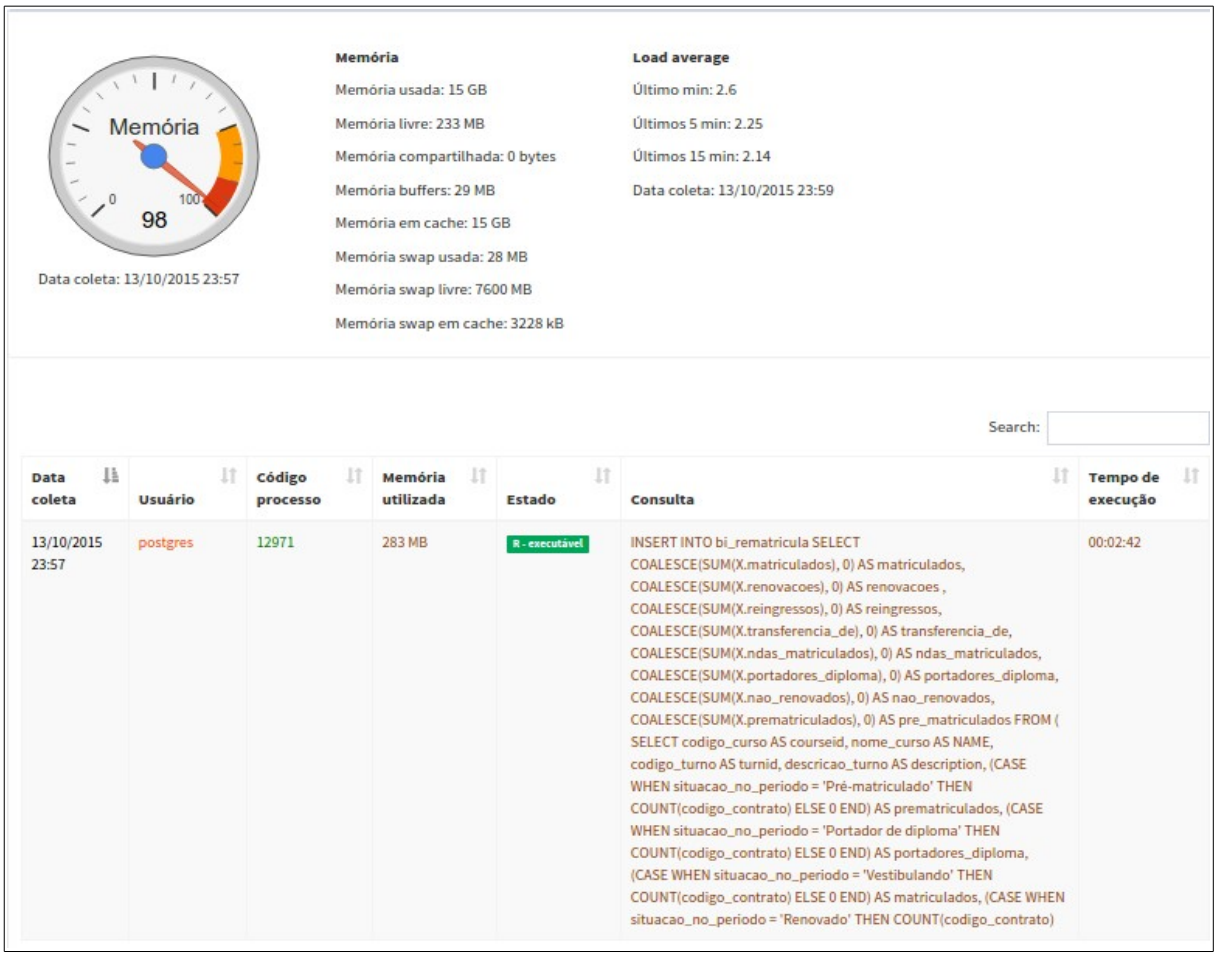
5.2.10 Processos em execução

Em um sistema de banco de dados vários processos podem estar em execução ao mesmo tempo, dessa maneira, é possível que um em específico possa estar onerando a execução de uma porção de outros comandos. É importante que o DBA visualize todos esses comandos e o tempo que os mesmos estão sendo executados, para que possa tomar alguma ação caso a carga do servidor estiver muito elevada.

Através do painel apresentado na figura 34 o usuário DBA pode visualizar os comandos que estão em execução na base de dados monitorada. Na mesma interface é possível acompanhar o uso de memória e a carga da máquina na ultima coleta. O painel apresentado é recarregado automaticamente a cada minuto para que as informações que estão sendo exibidas estejam sempre sincronizadas com a ultima coleta.

No instante da captura da imagem foi possível identificar que um comando do tipo INSERT estava em execução à dois minutos e quarenta e dois segundos e estava utilizando 283 MB de memória do servidor.

Figura 34 - Processos em execução.



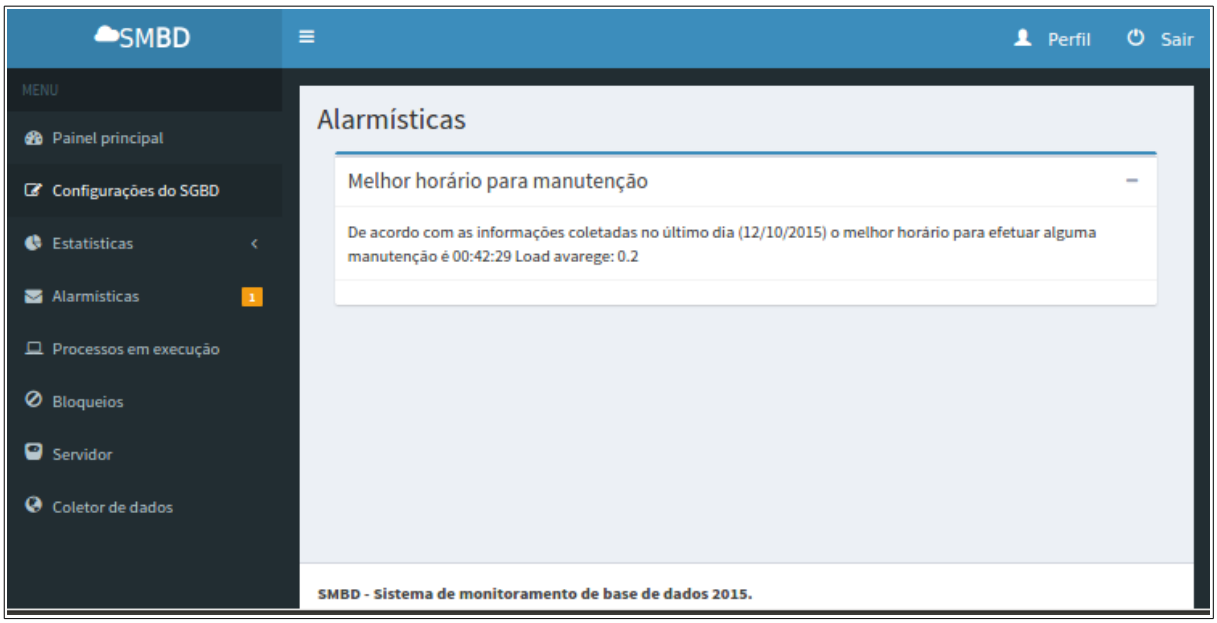
Fonte: Elaborado pelo autor.

5.2.11 Alarmísticas

A ferramenta desenvolvida além de exibir informações úteis que facilitam o monitoramento de uma base de dados, conta com uma funcionalidade baseada em alertas com o objetivo de informar o usuário DBA de possíveis problemas e avisos diversos.

Atualmente a ferramenta conta apenas com um tipo de alerta que informa o usuário DBA do melhor horário para efetuar uma manutenção. Na figura 35 é possível verificar o alerta na base de dados analisada, onde o melhor horário para efetuar manutenção registrado foi a meia noite e quarenta e dois minutos, horário onde a carga da maquina teve o menor valor registrado 0,2.

Figura 35 - Alarmísticas.



Fonte: Elaborado pelo autor.

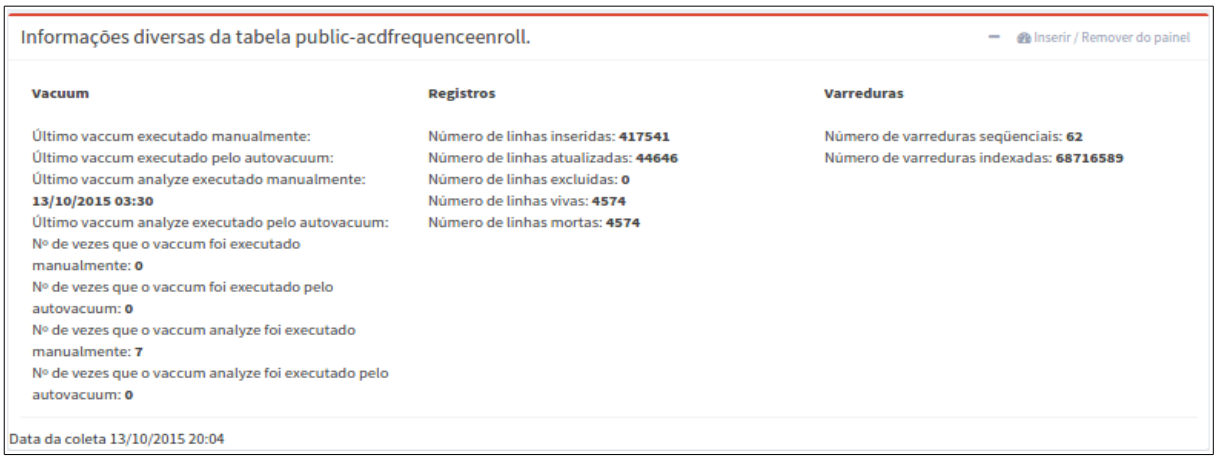
5.2.12 Informações de uma tabela da base de dados

Além de informações gerais da base de dados, como pode ser visto na figura 35 a ferramenta permite que sejam consultadas estatísticas de uma tabela em especifica. Para exemplificar a funcionalidade foi escolhida a tabela *acdfrequenceenroll,* que de acordo com os gráficos apresentados nas seções anteriores, apresenta o maior volume de informação armazenada na base de dados analisada.

A partir das informações coletadas e exibidas na figura 36 foi possível verificar que a maioria das consultas feitas nessa tabela utilizaram algum índice da mesma.

Nesse mesmo painel é possível verificar a última vez que o comando *vacuum* foi executado, caso o DBA perceba que isso faz muito tempo e a tabela analisada seja bastante acessada, é evidenciada a importância da execução do comando. Além disso, é possível verificar o número de registros, varreduras indexadas e não indexadas da tabela.

Figura 36 - Informações de uma tabela da base de dados.



Fonte: Elaborado pelo autor.

5.2.13 Índices utilizados de uma tabela

No *dashboard* apresentado na figura 37 o usuário administrador da base de dados pode verificar os índices utilizados de uma tabela do banco de dados, bem como o número de varreduras no mesmo.

Na base de dados monitorada foi possível identificar que boa parte dos índices da tabela *acdfrequenceenroll* são utilizados.

Figura 37 - Índices utilizados de uma tabela.

Fonte: Elaborado pelo autor.

5.2.14 Índices não utilizados de uma tabela

No *dashboard* apresentado na figura 38 o usuário DBA pode verificar os índices não utilizados de uma tabela do banco de dados, com essa informação pode avaliar a remoção dos mesmos caso estejam apenas ocupando espaço.

Na base de dados monitorada foi possível identificar que alguns índices da tabela *acdfrequenceenroll* não são utilizados e cada um deles tem em média 892 MB. Nessa base de dados esses índices poderiam ser excluídos pois estão apenas ocupando espaço.

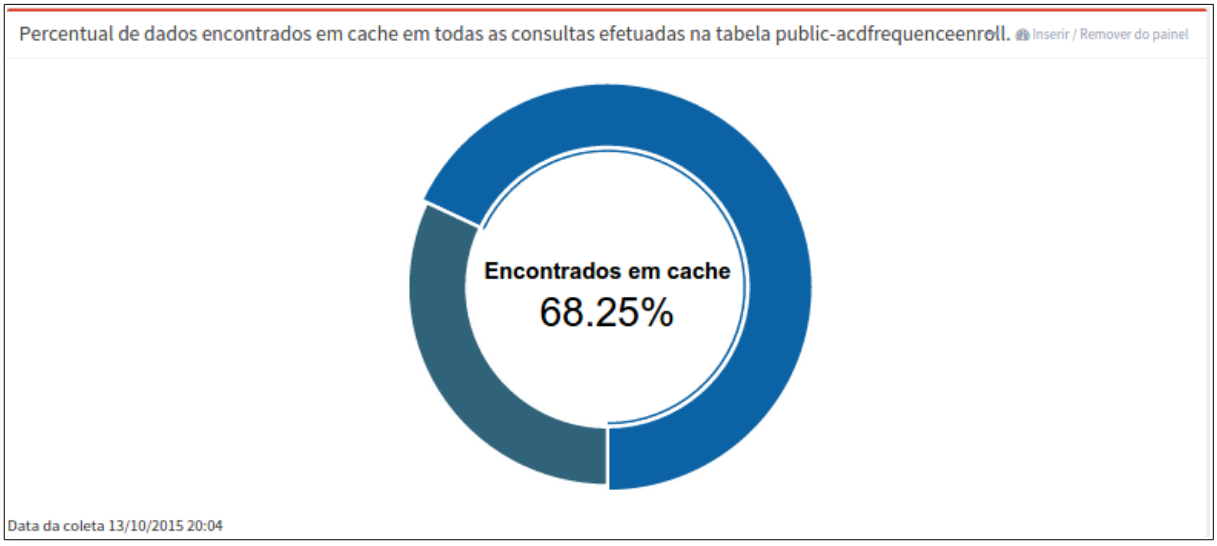
Figura 38 - Índices não utilizados de uma tabela.

Fonte: Elaborado pelo autor.

5.2.15 Uso do cache de uma tabela

O painel exibido na figura 39 exibe ao usuário DBA o aproveitamento do cache para a tabela selecionada. Para a tabela *acdfrequenceenroll* 68.25% dos dados consultados foram encontrados em cache.

Figura 39 - Uso do cache de uma tabela.



Fonte: Elaborado pelo autor.

5.3 Análise geral dos resultados obtidos com a ferramenta

Analisando os resultados obtidos verificou-se que o uso SMBD facilitou o monitoramento de uma base de dados real, visto que foi possível identificar momentos que o servidor esteve sobrecarregado, consultas que tiveram execução lenta, tabelas que tiveram poucas consultas que utilizaram índices e identificação de índices não utilizados.

O uso da ferramenta desenvolvida permitiu identificar as consultas mais lentas executadas no ultimo dia analisado, com essa informação o administrador pode otimizar diversos processos no software que está utilizando a base de dados monitorada.

Além de identificar tabelas que tenham índices que não estão sendo utilizado o software permitiu o administrador detectar tabelas com potencial para receber novos índices.

Com base nos resultados obtidos com a ferramenta no ambiente real foram identificadas ações que podem ser tomadas pelo DBA para melhorar o desempenho e evitar problemas no SGBD monitorado, algumas das ações a seguir não poderiam ter sido identificadas se utilizada uma das ferramentas estudadas no capítulo 3:

a) Efetuar manutenções no banco de dados no horário sugerido pelo SGBD a partir das alarmísticas;

b) Refatorar ou identificar um ponto de melhoria nas consultas que tiveram uma execução lenta;

c) Considerar a possibilidade de criar índices nas tabelas que tiveram poucas consultas que utilizaram índices;

d) Avaliar a possibilidade de deletar os índices que não estão sendo utilizados pela base de dados.

6 CONSIDERAÇÕES FINAIS

Através dos estudos realizados foi possível verificar a importância que o monitoramento e a análise das estatísticas disponibilizadas pelo SGBD empregam no funcionamento ideal e na performance de um banco de dados. Para evitar a perda de informação e garantir que os dados possam ser acessados de maneira eficaz, empresas procuram cada vez mais profissionais que tenham conhecimento em sistemas de banco de dados.

Ao propor o desenvolvimento de uma ferramenta de coleta de estatísticas e monitoramento de um SGBD, foi buscado atender uma demanda comum entre todos os administradores de banco de dados, manter a disponibilidade do serviço, identificar possíveis problemas de performance e prevenir impasses de maneira fácil.

A ferramenta desenvolvida demonstrou-se ser capaz de auxiliar o profissional a perceber diversas situações, tais como, identificar pontos de melhoria em consultas que tiveram execução lenta, possibilidade de criação de índices em tabelas que tiveram poucas consultas utilizaram esse recurso.

Após a conclusão da ferramenta SMBD foi possível avaliá-la em um ambiente em produção. Dessa forma, foi evidenciado que a mesma pode ser usada em cenários reais e que cumpriu com o objetivo de disponibilizar uma ferramenta para auxiliar na administração, monitoramento e otimização de um banco de dados que tenha um grande volume de informação armazenada.

Entende-se também que, além do objetivo principal, os objetivos secundários também foram alcançados, uma vez que foi possível compreender os princípios de coleta e utilização

de estatísticas de um banco de dados e contribuir para uma melhor administração do SGBD através da ferramenta.

O painel gerenciador das informações coletadas permite visualizar o monitoramento de apenas uma base de dados para cada conta de usuário, em trabalhos futuros deseja-se que o mesmo usuário possa acompanhar os dados coletados de mais de uma base e efetuar comparativos entre elas, ainda com o objetivo de tornar o SMBD mais proativo, podem ser implementados novos alarmes que sugerem formas de resolver problemas automaticamente identificados.

A coleta de estatísticas através de rotinas automatizadas suporta apenas o sistema gerenciador de banco de dados PostgreSQL, possibilitando a evolução da ferramenta com desenvolvimento de outros módulos coletores que permitam o monitoramento de outros SGBDs em trabalhos vindouros. Cabe ressaltar aqui que este trabalho é parte de uma iniciativa maior, com o objetivo de criar uma ferramenta que possa monitorar qualquer SGBD.

REFERÊNCIAS

BERKUS, J. Checklist de Performance do PostgreSQL 8.0. **PostgreSQL WIKI,** 2005. Disponível em: <https://wiki.postgresql.org/wiki/Checklist_de_Performance_do_PostgreSQL_8.0>. Acesso em. Acesso em: 06 mai. 2015.

BIAZUS, D.O. PostgreSQL. **PostgreSQL WIKI,** 2003. Disponível em: <https://wiki.postgresql.org/wiki/Introdu%C3%A7%C3%A3o_e_Hist%C3%B3rico>. Acesso em: 03 mai. 2015.

BLUMM, C; FORNARI, M. R. Dúvidas frequentes sobre Banco de Dados. **Revista SQL Magazine,** 28. ed., 2006.

BORELLO, F. ; KNEIPP, R. E. Álgebra Relacional. **Revista SQL Magazine**, p. 1 - 1, 2008.

CEDRUS. Cedrus: PostgreSQL Manager. Disponível em: <http://sourceforge.net/projects/cedrus/> Acesso em: 19 mai. 2015.

CYBERTEC. Pgwatch Cybertec Enterprise PostgreSQL Monitor. Disponível em: <http://www.cybertec.at/postgresql_produkte/pgwatch-cybertec-enterprise-postgresql-monitor/> Acesso em: 19 mai. 2015.

DATE, C. J. **Introdução a sistemas de bancos de dados.** 8.ed. Rio de Janeiro: Campus, 2004.

ELMASRI, R; NAVATHE, S. B. **Sistemas de banco de dados.** 6. ed. Pearson, 2011.

GIL, A. C. **Como elaborar projetos de pesquisa.** 4. ed. São Paulo: Atlas, 2002.

MILANI, A. **PostgreSQL:** Guia do Programador. São Paulo: Novatec Editora, 2008.

ORACLE. MySQL Enterprise Monitor. Disponível em: <https://www.mysql.com/products/enterprise/monitor.html> Acesso em 19 maio. 2015.

POSTGRESQL. Documentation. Disponível em <http://www.postgresql.org/docs/manuals/archive/ > Acesso em: 18 abr. 2015.

SANTOS, A. R. **Metodologia científica: a construção do conhecimento.** 2. ed. Rio de Janeiro: DP&A editora, 1999.

SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistemas de Bancos de Dados.** 3. ed. Makron Books, 1999.

SILBERSCHATZ, A.; KORTH, H.F.; SUDARSHAN, S; **Sistemas de Bancos de Dados.** 6. ed. Makron Books, 2012.

SMANIOTO, C. E. PostgreSQL. **Revista SQL Magazine,** 37. ed., 2007.

RAMAKRISHNAN, R.; GEHRKE, J. **Sistemas de Gerenciamento de Bando de Dados.** 3. ed. Mc Graw Hill, 2008.

ROB, P.; CORONEL, C. **Sistemas de Bando de Dados**: Projeto, implementação e administração. 8. ed. CENGATE Learning, 2011.

APÊNDICES

APÊNDICE A – Código fonte completo do agente coletor de dados.

```
1
2  <?php
3  /**
4   * Módulo coletor de estatísticas
5   * para o SGBD PostgreSQL do
6   * sistema SMBD (Sistema de monitoramento de base de dados)
7   */
8  $config = array(
9      //base de dados
10     'host' => 'localhost',
11     'port' => '5432',
12     'user' => 'postgres',
13     'password' => 'postgres',
14     'dbname' => 'avaliacao',
15     //smbd
16     'url' => 'http://smbd.com.br',
17     'usuario' => 'ftomasini.rs@gmail.com',
18     //intervalo de coleta
19     'tempo_coleta_sgbd_versao' => '200 minutes',
20     'tempo_coleta_base_de_dados' => '200 minutes',
21     'tempo_coleta_tabela' => '200 minutes',
22     'tempo_coleta_indice' => '200 minutes',
23     'tempo_coleta_configuracoes' => '200 minutes',
24     'tempo_coleta_loadavg' => '200 minutes',
25     'tempo_coleta_memoria' => '200 minutes',
26     'tempo_coleta_processos' => '1 minutes',
27     'tempo_coleta_bloqueios' => '200 minutes',
28     );
29
30 //Arquivo de log de erros
31 $log = fopen('/var/log/smbd.log', 'a+');
32
33 //Instancia da classe coletora
34 $smbdColetor = new smbdColetor($config);
35 //Define configurações de coleta para versão do banco de dados
36 $coletaSgbdVersao = new Coleta($config['tempo_coleta_sgbd_versao']);
37 //Define configurações de coleta para estatísticas do banco de dados
38 $coletaBaseDeDados = new Coleta($config['tempo_coleta_base_de_dados']);
39 //Define configurações de coleta para tabelas do banco de dados
40 $coletaTabela = new Coleta($config['tempo_coleta_tabela']);
41 //Define configurações de coleta para índices do banco de dados
42 $coletaIndice = new Coleta($config['tempo_coleta_indice']);
43 //Define configurações de coleta para configurações da base de dados
44 $coletaConfiguracao = new Coleta($config['tempo_coleta_configuracoes']);
45 //Define configurações de coleta para loadavg
```

```
46 $coletaLoadavg = new Coleta($config['tempo_coleta_loadavg']);
47 //Define configurações de coleta para memória do servidor
48 $coletaMemoria = new Coleta($config['tempo_coleta_memoria']);
49 //Define configurações de coleta para processos em execução
50 $coletaProcessos = new Coleta($config['tempo_coleta_processos']);
51 //Define configurações de coleta para bloqueios
52 $coletaBloqueios = new Coleta($config['tempo_coleta_bloqueios']);
53
54
55 echo "[OK] Serviço inicializado com sucesso! \n";
56
57 //Execução
58 while(1)
59 {
60     try
61     {
62         //Coleta da versão do banco de dados
63         $smbdColetor->stat_sgbd_versao($coletaSgbdVersao);
64         //Coleta de estatisticas do banco de dados
65         $smbdColetor->stat_base_de_dados($coletaBaseDeDados);
66         //Coleta de estatísticas das tabelas do banco de dados
67         $smbdColetor->stat_tabela($coletaTabela);
68         //Coleta de estatísticas dos índices do banco de dados
69         $smbdColetor->stat_indice($coletaIndice);
70         //Coleta de configurações do bando de dados
71         $smbdColetor->stat_configuracao_base_de_dados($coletaConfiguracao);
72         //Coleta do load do servidor
73         $smbdColetor->stat_loadavg($coletaLoadavg);
74         //Coleta do uso da memória do servidor
75         $smbdColetor->stat_memoria($coletaMemoria);
76         //Coleta dos processos em execução no servidor
77         $smbdColetor->stat_processos($coletaProcessos);
78         //Coleta dos bloqueios
79         $smbdColetor->stat_bloqueios($coletaBloqueios);
80     }
81     catch ( Exception $e )
82     {
83         fwrite($log, "Erro na coleta de dados! : {$e->getMessage()} \n");
84     }
85 }
86
87 // Fecha arquivo de log do coletor de dados
88 fclose($f);
89
90 /**
91  * Class smbdColetor
92  * Classe responsável por centralizar a logica das coletas.
93  *
94  */
95 class smbdColetor
96 {
97     private $host;
98     private $port;
99     private $user;
100     private $password;
101     private $dbname;
102     private $url;
103     private $clientOptions = array();
104     private $client;
105     private $usuario;
106
107
108     /**
109      * Construtor da classe
110      */
111     public function __construct($config)
112     {
113
```

```
114         $this->host = $config['host'];
115         $this->port = $config['port'];
116         $this->user = $config['user'];
117         $this->password = $config['password'];
118         $this->dbname = $config['dbname'];
119         $this->url = $config['url'];
120         $this->usuario = $config['usuario'];
121         $this->defineClientOptions();
122         $this->client = new SoapClient(NULL, $this->getClientOptions());
123     }
124 
125 
126     /**
127      * Obtém a versão da base de dados
128      * que está sendo analisada
129      *
130      * @throws Exception
131      */
132     public function stat_sgbd_versao($coleta)
133     {
134         if( $coleta->verificaColeta() )
135         {
136             $coleta->proximaColeta = date('d-m-Y H:i:s', strtotime("+$coleta-
    >tempoColeta"));
137             $coletaAtual = date('d-m-Y H:i:s');
138 
139             $this->openDb();
140             $dbres = pg_query("SELECT '{$coletaAtual}' as data_coleta,
141                                       '{$this->usuario}' as usuario,
142                                       version() as versao");
143 
144             $dados = array();
145             if(count($dbres)>0)
146             {
147                 $dados = $this->fetchObject($dbres);
148             }
149             $this->closeDb();
150 
151             $result = $this->client->ws($dados, 'stat_sgbd_versao');
152 
153             if ($result)
154             {
155                 echo "Estatística versão do servidor coletada!! \n";
156             }
157             else
158             {
159                 throw new Exception('Não foi possível obter a versão do banco de
    dados ' . $result);
160             }
161         }
162     }
163 
164 
165     /**
166      * Obtém estatísticas da base de dados.
167      * @throws Exception
168      */
169     public function stat_base_de_dados($coleta)
170     {
171         if( $coleta->verificaColeta() )
172         {
173             $coleta->proximaColeta = date('d-m-Y H:i:s', strtotime("+$coleta-
    >tempoColeta"));
174             $coletaAtual = date('d-m-Y H:i:s');
175             $this->openDb();
176             $bdNome = pg_escape_string($this->dbname);
177 
178             $dbres = pg_query("SELECT '{$coletaAtual}' as data_coleta,
```

```
179                                     '{$this->usuario}' as usuario,
180                                     pg_stat_database.datid,
181                                     pg_stat_database.datname,
182                                     pg_stat_database.numbackends,
183                                     pg_stat_database.xact_commit,
184                                     pg_stat_database.xact_rollback,
185                                     pg_stat_database.blks_read,
186                                     pg_stat_database.blks_hit,
187                                     pg_stat_database.tup_returned,
188                                     pg_stat_database.tup_fetched,
189                                     pg_stat_database.tup_inserted,
190                                     pg_stat_database.tup_updated,
191                                     pg_stat_database.tup_deleted,
192                                     pg_stat_database.conflicts,
193                                     date_trunc('seconds',
    pg_stat_database.stats_reset) AS stats_reset,
194                                     --conflitos
195
    pg_stat_database_conflicts.confl_tablespace,
196                                     pg_stat_database_conflicts.confl_lock,
197                                     pg_stat_database_conflicts.confl_snapshot,
198                                     pg_stat_database_conflicts.confl_bufferpin,
199                                     pg_stat_database_conflicts.confl_deadlock,
200                                     --cache
201                                     (SELECT round(AVG(
202                                             (heap_blks_hit +
    COALESCE(toast_blks_hit, 0))::numeric /
203                                             (heap_blks_read +
    COALESCE(toast_blks_read, 0) +
204                                             heap_blks_hit +
    COALESCE(toast_blks_hit,0))::numeric * 100), 3) as hit_ratio
205                                       FROM pg_statio_user_tables
206                                      WHERE (heap_blks_read +
    COALESCE(toast_blks_read, 0)) + heap_blks_hit + COALESCE(toast_blks_hit) > 0) as
    hit_ratio,
207
    pg_size_pretty(pg_database_size(pg_stat_database.datname)) as
    tamanho_base_de_dados_formatado,
208                                      pg_database_size(pg_stat_database.datname)
    as tamanho_base_de_dados
209                                FROM pg_stat_database
210                           INNER JOIN pg_stat_database_conflicts
211                                   ON pg_stat_database.datname =
    pg_stat_database_conflicts.datname
212                                WHERE pg_stat_database.datname= '$bdNome'");
213
214             $dados = array();
215             if(count($dbres)>0)
216             {
217                 $dados = $this->fetchObject($dbres);
218             }
219             $this->closeDb();
220
221             $result = $this->client->ws($dados, 'stat_base_de_dados');
222
223             if ($result)
224             {
225                 echo "Estatísticas de base de dados coletadas!! \n";
226             }
227             else
228             {
229                 throw new Exception('Não foi possível obter as estatísticas do
    banco de dados');
230             }
231         }
232
233     }
234
```

```
235     /**
236      * Obtém estatísticas das tabelas do
237      * banco de dados que está sendo analisado
238      *
239      * @throws Exception
240      */
241     public function stat_tabela($coleta)
242     {
243         if( $coleta->verificaColeta() )
244         {
245             $coleta->proximaColeta = date('d-m-Y H:i:s', strtotime("+$coleta-
    >tempoColeta"));
246             $coletaAtual = date('d-m-Y H:i:s');
247             $this->openDb();
248 
249             $dbres = pg_query("SELECT '{$coletaAtual}' as data_coleta,
250                                      '{$this->usuario}' as usuario,
251                                      relid,
252                                      schemaname,
253                                      relname,
254                                      seq_scan,
255                                      seq_tup_read,
256                                      idx_scan,
257                                      idx_tup_fetch,
258                                      n_tup_ins,
259                                      n_tup_upd,
260                                      n_tup_del,
261                                      n_tup_hot_upd,
262                                      n_live_tup,
263                                      n_dead_tup,
264                                      last_vacuum,
265                                      last_autovacuum,
266                                      last_analyze,
267                                      last_autoanalyze,
268                                      vacuum_count,
269                                      autovacuum_count,
270                                      analyze_count,
271                                      autoanalyze_count,
272                                      pg_relation_size(relid) as tamanho,
273                                      pg_total_relation_size(relid)as
    tamanho_com_indices,
274                                      pg_size_pretty(pg_relation_size(relid)) as
    tamanho_formatado,
275 
    pg_size_pretty(pg_total_relation_size(relid)) as tamanho_com_indices_formatado,
276                                      (SELECT round((heap_blks_hit +
    COALESCE(toast_blks_hit, 0))::numeric /
277                                                    (heap_blks_read +
    COALESCE(toast_blks_read, 0) +
278                                                     heap_blks_hit +
    COALESCE(toast_blks_hit,0))::numeric * 100, 3) as hit_ratio
279                                        FROM pg_statio_user_tables
280                                       WHERE pg_stat_user_tables.relname =
    pg_statio_user_tables.relname
281                                         AND pg_stat_user_tables.schemaname =
    pg_statio_user_tables.schemaname
282                                         AND pg_statio_user_tables.schemaname =
    'public'
283                                         AND (heap_blks_read +
    COALESCE(toast_blks_read, 0)) + heap_blks_hit + COALESCE(toast_blks_hit) > 0) as
    aproveitamento_cache
284                                  FROM pg_stat_user_tables");
285 
286             $dados = array();
287             if(count($dbres)>0)
288             {
289                 $dados = $this->fetchObject($dbres);
290             }
```

```
291             $this->closeDb();
292             $result = $this->client->ws($dados, 'stat_tabela');
293 
294             if ($result)
295             {
296                 echo "Estatísticas de tabela coletada!! \n";
297             }
298             else
299             {
300                 throw new Exception('Não foi possível obter estatísticas das
    tabelas do banco de dados');
301             }
302         }
303     }
304 
305     /**
306      * Obtém estatísticas dos índices
307      * da base de dados que está sendo analisada
308      *
309      * @throws Exception
310      */
311     public function stat_indice($coleta)
312     {
313         if( $coleta->verificaColeta() )
314         {
315             $coleta->proximaColeta = date('d-m-Y H:i:s', strtotime("+$coleta-
    >tempoColeta"));
316             $coletaAtual = date('d-m-Y H:i:s');
317             $this->openDb();
318 
319             $dbres = pg_query("SELECT '{$coletaAtual}' as data_coleta,
320                                       '{$this->usuario}' as usuario,
321                                       relid,
322                                       indexrelid,
323                                       schemaname,
324                                       relname,
325                                       indexrelname,
326                                       idx_scan,
327                                       idx_tup_read,
328                                       idx_tup_fetch,
329                                       (idx_scan > 0) as utilizado,
330                                       pg_relation_size(indexrelid) as tamanho,
331                                       pg_total_relation_size(relid)as
    tamanho_com_indices,
332 
    pg_size_pretty(pg_relation_size(indexrelid)) as tamanho_formatado,
333 
    pg_size_pretty(pg_total_relation_size(relid)) as tamanho_com_indices_formatado
334                                  FROM pg_stat_user_indexes");
335             $dados = array();
336             if(count($dbres)>0)
337             {
338                 $dados = $this->fetchObject($dbres);
339             }
340             $this->closeDb();
341             $result = $this->client->ws($dados, 'stat_indice');
342 
343             if ($result)
344             {
345                 echo "Estatistica de índice coletada!! \n";
346             }
347             else
348             {
349                 throw new Exception('Não foi possível obter estatísticas de
    índices');
350             }
351         }
352     }
```

```
353
354     /**
355      * Obtém configurações
356      * da base de dados que está sendo analisada
357      *
358      * @throws Exception
359      */
360     public function stat_configuracao_base_de_dados($coleta)
361     {
362         if( $coleta->verificaColeta() )
363         {
364             $coleta->proximaColeta = date('d-m-Y H:i:s', strtotime("+$coleta-
    >tempoColeta"));
365             $coletaAtual = date('d-m-Y H:i:s');
366             $this->openDb();
367
368             $dbres = pg_query(" SELECT '{$coletaAtual}' as data_coleta,
369                                       '{$this->usuario}' as usuario,
370                                       name,
371                                       setting,
372                                       unit,
373                                       category,
374                                       short_desc,
375                                       extra_desc,
376                                       context,
377                                       vartype,
378                                       source,
379                                       min_val,
380                                       max_val,
381                                       enumvals,
382                                       boot_val,
383                                       reset_val,
384                                       sourcefile,
385                                       sourceline
386                                  FROM pg_settings");
387             $dados = array();
388             if(count($dbres)>0)
389             {
390                 $dados = $this->fetchObject($dbres);
391             }
392             $this->closeDb();
393             $result = $this->client->ws($dados,
    'stat_configuracao_base_de_dados');
394
395             if ($result)
396             {
397                 echo "Configurações da base de dados coletadas!! \n";
398             }
399             else
400             {
401                 throw new Exception('Não foi possível as configurações da base de
    dados');
402             }
403         }
404     }
405
406     /**
407      * Obtém loadavg do servidor
408      * da base de dados que está sendo analisada
409      *
410      * @throws Exception
411      */
412     public function stat_loadavg($coleta)
413     {
414         if( $coleta->verificaColeta() )
415         {
416             $coleta->proximaColeta = date('d-m-Y H:i:s', strtotime("+$coleta-
    >tempoColeta"));
```

```
417             $coletaAtual = date('d-m-Y H:i:s');
418
419             $this->openDb();
420
421             $dbres = pg_query("SELECT '{$coletaAtual}' as data_coleta,
422                                       '{$this->usuario}' as usuario,
423                                       load1 as load_ultimo_minuto,
424                                       load5 as load_ultimos_5_minutos,
425                                       load15 as load_ultimos_15_minutos
426                                  FROM pg_loadavg();");
427             $dados = array();
428             if(count($dbres)>0)
429             {
430                 $dados = $this->fetchObject($dbres);
431             }
432             $this->closeDb();
433             $result = $this->client->ws($dados, 'stat_loadavg');
434
435             if ($result && $dados)
436             {
437                 echo "Loadavg do servidor coletado!! \n";
438             }
439             else
440             {
441                 throw new Exception('Não foi possível obter o load do servidor');
442             }
443         }
444     }
445
446     /**
447      * Obtém memoria do servidor
448      * da base de dados que está sendo analisada
449      *
450      * @throws Exception
451      */
452     public function stat_memoria($coleta)
453     {
454         if( $coleta->verificaColeta() )
455         {
456             $coleta->proximaColeta = date('d-m-Y H:i:s', strtotime("+$coleta-
    >tempoColeta"));
457             $coletaAtual = date('d-m-Y H:i:s');
458
459             $this->openDb();
460
461             $dbres = pg_query("SELECT '{$coletaAtual}' as data_coleta,
462                                       '{$this->usuario}' as usuario,
463                                       pg_size_pretty(memused * 1024) as
    format_memused,
464                                       pg_size_pretty(memfree * 1024) as
    format_memfree,
465                                       pg_size_pretty(memshared * 1024) as
    format_memshared,
466                                       pg_size_pretty(membuffers * 1024) as
    format_membuffers,
467                                       pg_size_pretty(memcached * 1024) as
    format_memcached,
468                                       pg_size_pretty(swapused * 1024) as
    format_swapused,
469                                       pg_size_pretty(swapfree * 1024) as
    format_swapfree,
470                                       pg_size_pretty(swapcached * 1024) as
    format_swapcached,
471                                       memused,
472                                       memfree,
473                                       memshared,
474                                       membuffers,
475                                       memcached,
```

```
476                                     swapused,
477                                     swapfree,
478                                     swapcached
479                                FROM pg_memusage();");
480             $dados = array();
481             if(count($dbres)>0)
482             {
483                 $dados = $this->fetchObject($dbres);
484             }
485 
486             $this->closeDb();
487             $result = $this->client->ws($dados, 'stat_memoria');
488 
489             if ($result)
490             {
491                 echo "Memória do servidor coletada coletadas!! \n";
492             }
493             else
494             {
495                 throw new Exception('Não foi possível obter status da memória do
    servidor');
496             }
497         }
498     }
499 
500     /**
501      * Obtém processos em exucução na base de dados
502      * da base de dados que está sendo analisada
503      *
504      * @throws Exception
505      */
506     public function stat_processos($coleta)
507     {
508         if( $coleta->verificaColeta() )
509         {
510             $coleta->proximaColeta = date('d-m-Y H:i:s', strtotime("+$coleta-
    >tempoColeta"));
511             $coletaAtual = date('d-m-Y H:i:s');
512 
513             $this->openDb();
514             $bdNome = pg_escape_string($this->dbname);
515 
516             $dbres = pg_query(" SELECT '{$coletaAtual}' as data_coleta,
517                                     '{$this->usuario}' as usuario,
518                                     pg_stat_activity.datname,
519                                     pg_stat_activity.usename,
520                                     pg_stat_activity.pid,
521                                     priority,
522                                     pg_size_pretty(B.rss * 1024) as memoria,
523                                     B.state,
524                                     pg_stat_activity.query,
525                                     'PROC_SMBD_COLETOR' as identificador,
526                                     date_trunc('seconds',
    pg_stat_activity.backend_start) AS inicio_processo,
527                                     date_trunc('seconds', now()) AS
    hora_coleta,
528                                     date_trunc('seconds',SUM(now() -
    pg_stat_activity.backend_start)) as tempo_execussao
529                                FROM pg_stat_activity
530                           INNER JOIN pg_proctab() B
531                                   ON pg_stat_activity.pid = B.pid
532                                  AND pg_stat_activity.query not ilike
    '%PROC_SMBD_COLETOR%'
533                                WHERE datname = '$bdNome'
534                             GROUP BY 1,2,3,4,5,6,7,8,9,10,11,12; ");
535             $dados = array();
536             if(count($dbres)>0)
537             {
```

```
538                 $dados = $this->fetchObject($dbres);
539             }
540             $this->closeDb();
541             $result = $this->client->ws($dados, 'stat_processos');
542 
543             if ($result)
544             {
545                 echo "Processos em execução coletados \n";
546             }
547             else
548             {
549                 throw new Exception('Não foi possível obter os processos em
    execução');
550             }
551         }
552     }
553 
554 
555     /**
556      * Obtém bloqueios da base de dados
557      *
558      * @throws Exception
559      */
560     public function stat_bloqueios($coleta)
561     {
562         if( $coleta->verificaColeta() )
563         {
564             $coleta->proximaColeta = date('d-m-Y H:i:s', strtotime("+$coleta-
    >tempoColeta"));
565             $coletaAtual = date('d-m-Y H:i:s');
566 
567             $this->openDb();
568 
569             $bdNome = pg_escape_string($this->dbname);
570 
571             $dbres = pg_query(" SELECT '{$coletaAtual}' as data_coleta,
572                                        '{$this->usuario}' as usuario,
573                                        pg_stat_activity.datname,
574                                        pg_stat_activity.usename,
575                                        pg_stat_activity.pid,
576                                        priority,
577                                        pg_size_pretty(B.rss * 1024) as memoria,
578                                        B.state,
579                                        pg_stat_activity.query,
580                                        C.mode,
581                                        C.granted,
582                                        'PROC_SMBD_COLETOR' as identificador,
583                                        date_trunc('seconds',
    pg_stat_activity.backend_start) AS inicio_processo,
584                                        date_trunc('seconds', now()) AS
    hora_coleta,
585                                        date_trunc('seconds',SUM(now() -
    pg_stat_activity.backend_start)) as tempo_execussao
586                                   FROM pg_stat_activity
587                              INNER JOIN pg_proctab() B
588                                      ON pg_stat_activity.pid = B.pid
589                              INNER JOIN pg_locks C
590                                      ON pg_stat_activity.pid = C.pid
591                                     AND pg_stat_activity.query not ilike
    '%PROC_SMBD_COLETOR%'
592                                  WHERE datname = '$bdNome'
593                               GROUP BY 1,2,3,4,5,6,7,8,9,10,11,12,13,14; ");
594             $dados = array();
595             if(count($dbres)>0)
596             {
597                 $dados = $this->fetchObject($dbres);
598             }
599             $this->closeDb();
```

```
600             $result = $this->client->ws($dados, 'stat_bloqueios');
601             $result = true;
602 
603             if ($result)
604             {
605                 echo "Bloqueios coletados \n";
606             }
607             else
608             {
609                 throw new Exception('Não foi possível obter os processos em
    execução');
610             }
611         }
612     }
613 
614     /**
615      * Define parâmetros de conexão
616      */
617     protected function defineClientOptions()
618     {
619         $this->clientOptions["location"] = $this->url . "/webservice.php";
620         $this->clientOptions["uri"] = "$this->url";
621         $this->clientOptions["encoding"] = "UTF-8";
622     }
623 
624     /**
625      * Obtém configurações de conexão com sistema externo
626      * @return array
627      */
628     protected function getClientOptions()
629     {
630         return $this->clientOptions;
631     }
632 
633     /**
634      * Abre conexão com o banco de dados
635      * @throws Exception
636      */
637     protected function openDb()
638     {
639         if (!$con = pg_connect("host={$this->host}
640                                 port={$this->port}
641                                 user={$this->user}
642                                 password={$this->password}
643                                 dbname={$this->dbname}"))
644         {
645             throw new Exception("Conexão com a base de dados falhou!");
646         }
647 
648         if (!pg_dbname($con))
649         {
650             throw new Exception("A base de dados {$this->dbname} não foi
    encontrada no servidor.");
651         }
652 
653         return $con;
654     }
655 
656     /**
657      * Fecha conexão com
658      * o banco de dados
659      */
660     protected function closeDb()
661     {
662         pg_close();
663     }
664 
665     /**
```

```
666      * Transforma o array de dados obtido pelo banco
667      * em um array de objetos stdclass
668      *
669      * @param $data
670      * @return array
671      */
672     protected function fetchObject($data)
673     {
674         $returnData = array();
675         while ( ($obj = pg_fetch_object($data)) != NULL )
676         {
677             $returnData[] = $obj;
678         }
679         return $returnData;
680     }
681
682 }
683
684 class Coleta
685 {
686     public $tempoColeta;
687     public $proximaColeta;
688
689     /**
690      * Construtor da classe
691      */
692     public function __construct($tempoColeta)
693     {
694         $this->tempoColeta = $tempoColeta;
695     }
696
697     /**
698      * Verifica se está na hora de efetuar a coleta
699      * @return bool
700      */
701     public function verificaColeta()
702     {
703         $retorno = false;
704
705         if((!$this->proximaColeta) || strtotime($this->proximaColeta) <
    strtotime(date('d-m-Y H:i:s')) )
706         {
707             $retorno = true;
708         }
709
710         return $retorno;
711     }
712 }
713
714 ?>
```