UNIVERSIDADE DE CAXIAS DO SUL
ÁREA DO CONHECIMENTO DE CIÊNCIAS EXATAS E ENGENHARIAS

JOSÉ EDUARDO THUMS

TUNING EM BANCO DE DADOS *POSTGRESQL*: UM ESTUDO DE CASO

CAXIAS DO SUL
2018

JOSÉ EDUARDO THUMS

TUNING EM BANCO DE DADOS *POSTGRESQL*: UM ESTUDO DE CASO

Trabalho de Conclusão de Curso apresentado como requisito para a obtenção do Grau de Bacharel em Ciência da Computação da Universidade de Caxias do Sul, Área do Conhecimento de Ciências Exatas e Engenharias.

Orientadora: Prof.a Dra. Helena Graziottin Ribeiro

CAXIAS DO SUL
2018

AGRADECIMENTOS

Agradeço em primeiro lugar a Deus, por me proporcionar essa conquista.

Aos meus pais, Beno Inacio Thums e Vilma Matilde Thums, que sempre trabalharam muito, colocando a necessidade dos filhos em primeiro lugar, proporcionando a educação necessária e criado uma família exemplar.

À minha esposa Aline Regina Horbach e minha filha Valentina Horbach Thums, que sempre me apoiaram e incentivaram nessa caminhada, compreendendo minhas ausências em casa.

As minhas irmãs, Teresa Anastácia Thums, Janete Maria Ody, Lúcia Teresinha Thums e Marlice Inês Thums, por sempre estarem presentes na minha vida, proporcionando momentos em família e de alegria.

A minha orientadora, Prof.ª Helena, pelas orientações que me guiaram nesse trabalho e estar sempre disposta a ajudar.

Ao Prof. Alexandre e Daniel, pelas dicas e apontamentos realizados, que auxiliaram a refinar a estrutura do trabalho, pela base de dados disponibilizada pelo Prof. Daniel, que foi de extrema importância para o desenvolvimento desse trabalho.

Aos meus colegas Diogo e Lucas pelos ensinamentos durante as aulas e pelos momentos de descontração.

A todos que não foram citados explicitamente, mas que de alguma forma me auxiliaram para que esse trabalho fosse concluído, muito obrigado.

RESUMO

A eficiência de um banco de dados está ligada diretamente com o bom desempenho de determinada aplicação ou sistema. Conseguir obter o máximo de desempenho do banco de dados e seu sistema de gerenciamento pode ser o diferencial de determinado sistema. Existem diversos sistemas de bancos de dados disponíveis, cada qual com suas particularidades. O *PostgreSQL* é um sistema de código aberto, amplamente usado nos mais diversos segmentos. Por esse motivo, este trabalho apresenta um estudo sobre *tuning* em banco de dados *PostgreSQL*, com o objetivo de verificar a melhora de desempenho das operações de consulta após sua aplicação. É realizada uma apresentação das técnicas de *tuning* disponíveis, com um aprofundamento no estudo da técnica de *tuning* de índices, orientando a melhor forma quanto ao uso, pois é imprescindível que se haja com conhecimento e responsabilidade a fim de não causar impactos negativos ao sistema. As técnicas estudadas foram aplicadas em um estudo de caso, no qual foi possível verificar que se corretamente aplicadas, melhoram o desempenho das consultas no *PostgreSQL*.

Palavras-chaves: Banco de dados. Desempenho. Índices. *PostgreSQL*. *Tuning*.

ABSTRACT

The efficiency of a database is directly linked to the good performance of a particular application or system. Getting the maximum performance from the database and your management system can be the differential of a particular system. There are several database systems available, each with its own particularities. PostgreSQL is an open source system, widely used in many different segments. For this reason, this work presents a study on tuning in PostgreSQL database, in order to verify the performance improvement of the query operations after its application. A presentation of the available tuning techniques is carried out, with a deepening in the study of the technique of tuning indexes, guiding the best way as to the use, since it is imperative that there be knowledge and responsibility in order not to cause negative impacts to the system. The techniques studied were applied in a case study, in which it was possible to verify that if correctly applied, they improve the performance of the queries in PostgreSQL.

Keywords: Database. Performance. Indexes. PostgreSQL. Tuning.

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE SIGLAS

ACID	Atomicidade, Consistência, Isolação e Durabilidade
DBA	Administrador de Banco de Dados
GB	*Gigabyte*
GHz	*Gigahertz*
GIN	*Generalized Inverted Index*
GiST	*Generalized Search Tree*
MVCC	Controle de Concorrência Multiversão
OLTP	*Online Transaction Processing*
RAM	*Random Access Memory*
SSD	*Solid State Drive*
SGBD	Sistema Gerenciador de Banco de Dados
SQL	Linguagem de Consulta Estruturada
TPC-C	*Transaction Processing Performance Council*

SUMÁRIO

1 INTRODUÇÃO

Atualmente os sistemas de bancos de dados são essenciais na vida da sociedade moderna, eles estão presentes na maioria das atividades: retirar ou depositar dinheiro em um banco, comprar um produto em um *site*, buscar um número na agenda do telefone ou nos aplicativos dos nossos telefones celulares. Nas empresas são usados para armazenar dados de estoques, vendas, setor de pessoal e contabilidade.

Para Rob e Coronel (2011) um banco de dados é uma estrutura computacional compartilhada que armazena um conjunto de dados do usuário final. Para manipularmos esses dados, usamos o Sistema Gerenciador de Banco de Dados (SGBD) que "[...] é um conjunto de programas que gerenciam a estrutura do banco de dados e controlam o acesso aos dados armazenados. ". (ROB; CORONEL, 2011, p.6), além de garantir a disponibilidade dos dados de forma eficaz.

Geralmente logo após a implementação de um sistema, o SGBD consegue disponibilizar as informações com rapidez. Porém, com o passar do tempo ocorre um aumento expressivo e muitas vezes não planejado de informações no banco de dados, que por muitas vezes acaba comprometendo o desempenho do SGBD. Rob e Coronel (2011, p. 470) descrevem que: "Infelizmente, a mesma consulta pode funcionar bem um dia e não tão bem dois meses depois". É nessa hora que são aplicadas as técnicas de *tuning*.

A palavra *tuning* pode ser traduzida como sintonia ou ajuste. *Tuning* trata do ajuste do desempenho do banco de dados ou de comandos de Linguagem de Consulta Estruturada (SQL), que visa minimizar o caminho usado pelo SGBD para buscar os dados em seu banco de dados, conforme Rao e Krishna (2014).

Para um banco de dados ter um desempenho adequado é necessário tomar boas decisões durante o projeto desse sistema, estimando corretamente o volume de dados que irá receber, relações de tabelas, consultas que serão realizadas e a frequência delas. Entretanto muitas decisões tomadas na fase de projeto podem se mostrar ineficientes, sendo que o real funcionamento do banco de dados é conhecido após a sua implementação e conforme a carga de dados for aumentando expressivamente.

1.1 OBJETIVOS DO TRABALHO

O principal objetivo deste trabalho é verificar a melhora de performance do banco de dados *PostgreSQL* com a aplicação de técnicas de *tuning* através de um estudo de caso. A proposta do estudo de caso será realizada através dos seguintes objetivos específicos:

- Escolher as técnicas de *tuning* que serão aplicadas no estudo de caso;
- Elencar os principais fatores que podem comprometer o desempenho do banco de dados;
- Aplicar as técnicas de *tuning* escolhidas no estudo de caso;
- Analisar os resultados obtidos no estudo de caso.

1.2 ORGANIZAÇÃO DO TRABALHO

O trabalho está dividido em oito capítulos para uma melhor compreensão.

No primeiro capítulo é realizada uma introdução sobre o tema abordado, e também o conceito e o significado da palavra *tuning*. Por fim, são destacados os objetivos do trabalho e sua estruturação.

No segundo capítulo é abordado o conceito de sintonia em banco de dados, tipos existentes e definições, com ênfase na sintonia de índices, no qual são apresentados os conceitos sobre os tipos de índices em bancos de dados relacionais. Ao final deste capítulo, são apresentados alguns trabalhos relacionados a essa área.

No terceiro capítulo são abordados os tipos de índices no *PostgreSQL*, conceituação e exemplos, bem como a apresentação dos fatores que possam comprometer o uso e a utilização de modo eficiente.

No quarto capítulo é abordado o conceito de *Benchmark* e suas classificações, também são citados alguns exemplos de softwares de *Benchmark*.

No quinto capítulo é apresentada uma explicação detalhada sobre o plano de execução montado pelo *PostgreSQL*, abordando os tipos existentes e o funcionamento.

No sexto capítulo é apresentada a proposta de solução, descrevendo o ambiente usado, destacando como será a implementação e os critérios a serem usados.

No sétimo capítulo é apresentado o estudo de caso, contendo a identificação e a descrição das bases de dados usadas, os testes realizados, os dados coletados e as respectivas avaliação dos resultados de cada teste.

No oitavo capítulo são apresentadas as considerações finais, a contribuição do trabalho e possíveis temas para trabalhos futuros.

2 SINTONIA EM BANCO DE DADOS

Quando determinado banco de dados entra em produção, as operações nele realizadas podem apresentar problemas que não foram considerados ou esperados na sua fase de projeto. Através de um monitoramento constante da utilização dos recursos e processamento interno do SGBD, podem ser descobertos gargalos como disputa pelos mesmos dados ou dispositivos e volume de atividades.

Conforme Elmasri e Navathe (2011) os objetivos da sintonia em banco de dados são justamente tratar esses gargalos, fazendo com que as aplicações que usam esse banco de dados sejam executadas com mais rapidez, diminuam o tempo de resposta de transações e melhorem o desempenho geral das transações.

Oliveira et al. (2015) define em sua pesquisa que o principal objetivo da sintonia é a busca pelo melhor desempenho, que pode ser realizada através de ajustes das configurações, parâmetros e projeto físico, seleção de estruturas de acesso e duplicação de estruturas físicas, sempre de acordo com a carga trabalho do banco de dados.

Rob e Coronel destacam que

> A sintonização de desempenho de banco de dados refere-se a um conjunto de atividades e procedimentos projetados para reduzir o tempo de resposta de um sistema de banco de dados - ou seja, para assegurar que uma consulta do usuário final seja processada pelo SGBD no período mínimo de tempo. (2011, p.471).

Conforme destacado pelos autores, a sintonia é um processo que se aplica para ganho de desempenho. O processo de sintonia em um banco de dados consome bastante tempo de um Administrador de Banco de Dados (DBA) ou outros profissionais da área, logo se pode afirmar que só é válido trabalhar a sintonia em bancos de dados que estejam realmente com problemas de desempenho em determinadas operações.

2.1 TIPOS DE SINTONIA

O processo da sintonia pode ser aplicado em todo o projeto do banco de dados, não somente aos comandos SQL. Para Tramontina (2008), os ajustes para otimização podem ser aplicados em sua estrutura lógica, estrutura física e na alocação da memória, ou seja, em todos os setores do projeto em que é possível realizar algum tipo de ajuste que traga melhorias.

Nas próximas seções serão abordados os seguintes processos de sintonização:

- Sintonia do SGBD: Ajustar os parâmetros de configuração usados pelo SGBD para realizar as operações sobre o banco de dados.
- Sintonia de consultas: Reescrita de consultas para melhorar o desempenho do otimizador e evitar recuperar informações que não serão usadas.
- Sintonia de índices: Criação de estruturas auxiliares para evitar acessos à memória secundária e com isso melhorar o desempenho do banco de dados.

Por definição do autor, a técnica de sintonia de índices será a mais desenvolvida nesse estudo e posteriormente aplicada no estudo de caso.

2.1.1 Sintonia do SGBD

Muitas vezes o administrador do banco de dados realiza a instalação padrão do banco de dados, não levando em conta que existem muitos parâmetros que podem não estar de acordo com o tipo e quantidade de dados que serão armazenados e processados.

Tramontina (2008) define que a sintonia dos parâmetros de configuração do SGBD é um processo muito importante, pois ela visa sintonizar parâmetros como ajuste de alocação de memória, ajustar o número máximo de conexões simultâneas, ajustes dos caminhos de acessos, ajustes de entrada e saída (I/O) para se obter uma alocação apropriada de recursos e manter os registros da base de dados contíguos.

2.1.2 Sintonia de consultas

Grande parte das consultas realizadas muitas vezes não alcançam o desempenho esperado por serem escritas pensando no resultado, e não na sua performance. Motivos como, a falta de conhecimento técnico dos desenvolvedores ou o tempo disponível para entrega de um projeto, são vistos frequentemente. Somando isso ao fato da maioria das transações serem de consultas, a sintonia de consultas é um processo que toma muito tempo dos profissionais dessa área.

Fritchey define que

> Sintonia de desempenho de consulta continua a ser um mecanismo vital para melhorar o desempenho dos seus sistemas de gerenciamento de banco de dados. A beleza da sintonia de desempenho de consulta é isso, em muitos casos, uma pequena mudança para um índice ou uma consulta SQL pode resultar em um aplicativo muito mais eficiente a um custo muito baixo. Nesses casos, o aumento de desempenho pode ser de ordem de magnitude melhor do que o oferecido por uma CPU incrementalmente mais rápida ou um otimizador um pouco melhor. (2012, P. 1).

Se uma consulta SQL for mal escrita, ela pode levar o otimizador a escolher um caminho que não é o mais adequado, fazendo com que sejam realizadas mais operações do que era realmente necessário para conseguir recuperar as informações solicitadas. A sintonia de consultas visa a recuperação dos dados desejados no menor tempo possível.

A principal forma de realizar a sintonia em consultas é a reescrita delas, escrevendo-as de tal forma que o otimizador não precise perder tempo reescrevendo a consulta e evitando que seja escolhido um caminho que não seja o mais adequado ou que recupere mais informações do que realmente seja necessário. Como um exemplo disso, pode-se citar consultas que usam *SELECT * FROM*, que devem ser analisadas e reescritas para trazer somente os dados realmente necessários, pois esse tipo de consulta realiza uma busca sequencial e acarreta em trazer toda a informação guardada, acarretando uma possível perda de desempenho.

A principal justificativa para a técnica de sintonia de consultas ser uma das primeiras a ser usada é que ela afeta somente a consulta que será reescrita, não se propagando para os demais comandos, evitando maiores intervenções no funcionamento do banco de dados, conforme Sasha e Bonnet (2003).

2.1.3 Sintonia de índices

"A indexação talvez seja o método de otimização mais importante, pois é o que consegue obter melhores resultados em menor espaço de tempo. ". (MILANI, 2008, p. 220). Porém para falar sobre sintonia de índices, é necessário falar um pouco do que são estruturas de índices e como funcionam.

Alves (2014) define que um índice nada mais é do que um arquivo auxiliar que contém um campo de determinada tabela (quando um índice simples) ou mais de um campo (caso seja um índice composto) definidos como campos de indexação que possuem o valor do campo e ponteiros que direcionam para o registro em si dentro da tabela de dados.

Rob e Coronel (2011) definem que um índice é uma estrutura ordenada de chaves e ponteiros, que são utilizados pelos SGBD's para recuperar dados de forma mais eficiente e recuperar dados ordenados por mais de um atributo.

Elmasri e Navathe (2011) classificam os índices em:

- Índices ordenados de um único nível;
- Índices multiníveis dinâmicos;
- Índice de *hash;*
- Índice *bitmap*.

Já Silberschatz, Korth e Sudarshan (2012) classificam os índices em:

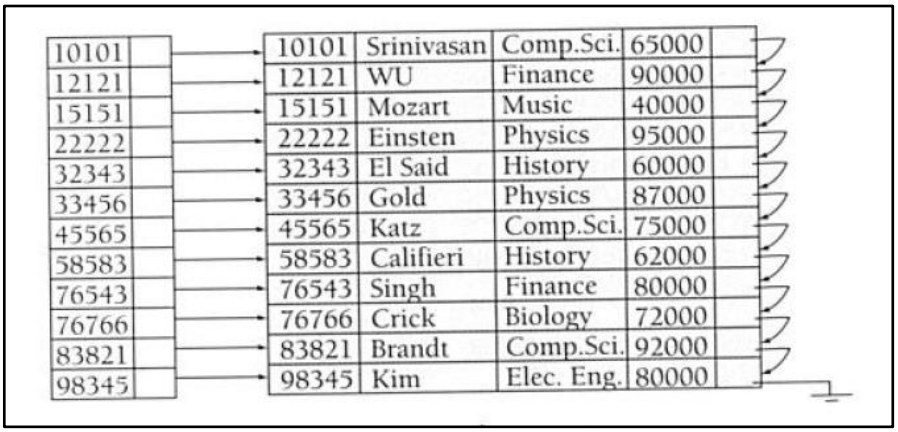
- Índices ordenados;
    - Densos;
    - Esparsos.
- Índice de *hash*.

2.1.3.1 Índice denso

Aparece um registro de índice para cada valor de chave de busca no arquivo. Pode-se ter um índice de agrupamento denso, se cada registro de índice contém o valor da chave de busca e um ponteiro para o primeiro registro de dados com esse valor, e um índice denso não agrupado, onde o índice armazena uma lista de ponteiros para todos os registros com o mesmo valor caso a chave de busca não seja uma chave primária, conforme Silberschatz, Korth e Sudarshan (2012).

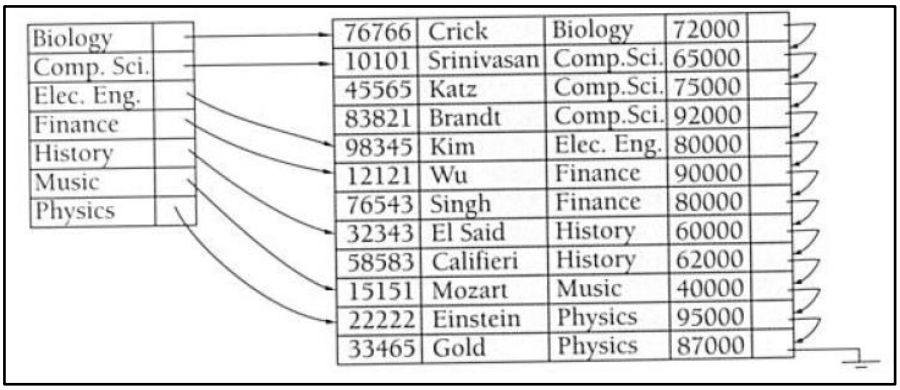
A Figura 1 mostra um exemplo de um índice denso sobre uma chave de busca primária, já a Figura 2 mostra um exemplo de um índice denso agrupado para um campo com valores repetidos.

Figura 1 - Índice denso sobre chave primária



Fonte: Silberschatz, Korth e Sudarshan (2012).

Figura 2 - Índice denso agrupado



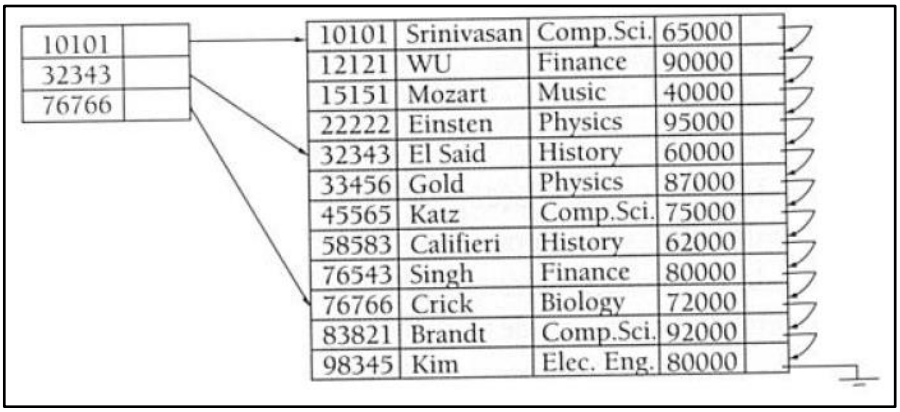
Fonte: Silberschatz, Korth e Sudarshan (2012).

2.1.3.2 Índice esparso

Uma entrada de índice aparece apenas para alguns valores de chave de busca. Esse tipo de índice pode ser usado apenas se for um índice agrupado, ou seja, se for armazenado ordenadamente conforme a chave de busca. Cada registro de índice contém o valor da chave de busca e um ponteiro para o primeiro registro de dados com esse valor, conforme Silberschatz, Korth e Sudarshan (2012).

A Figura 3 mostra um exemplo de um índice esparso sobre uma chave de busca primária.

Figura 3 - Índice esparso sobre chave primária



Fonte: Silberschatz, Korth e Sudarshan (2012).

2.1.3.3 Índices ordenados de um único nível

Elmasri e Navathe (2011) e Silberschatz, Korth e Sudarshan (2012) definem que índices ordenados de um único nível são estruturas que podem ser assimiladas com um índice ordenado de um livro, onde temos uma lista de capítulos e seções e seus devidos endereços, que são os números de páginas correspondentes ao conteúdo especificado. Esse tipo de índice armazena o valor do campo de índice juntamente com uma lista de ponteiros para todos os blocos de disco que contém registros com esse valor de campo, esses valores são ordenados para que seja possível aplicar uma pesquisa binária no índice.

2.1.3.3.1 Índices primários e de agrupamento

Silberschatz, Korth e Sudarshan (2012) definem que um índice de agrupamento é aquele cujo a chave de busca define a ordem sequencial do arquivo, e que esses índices também são chamados de primários. Um índice primário pode ser montado baseado em qualquer chave de busca desde que essa chave defina a sequencialidade do arquivo, não sendo necessariamente a chave primária.

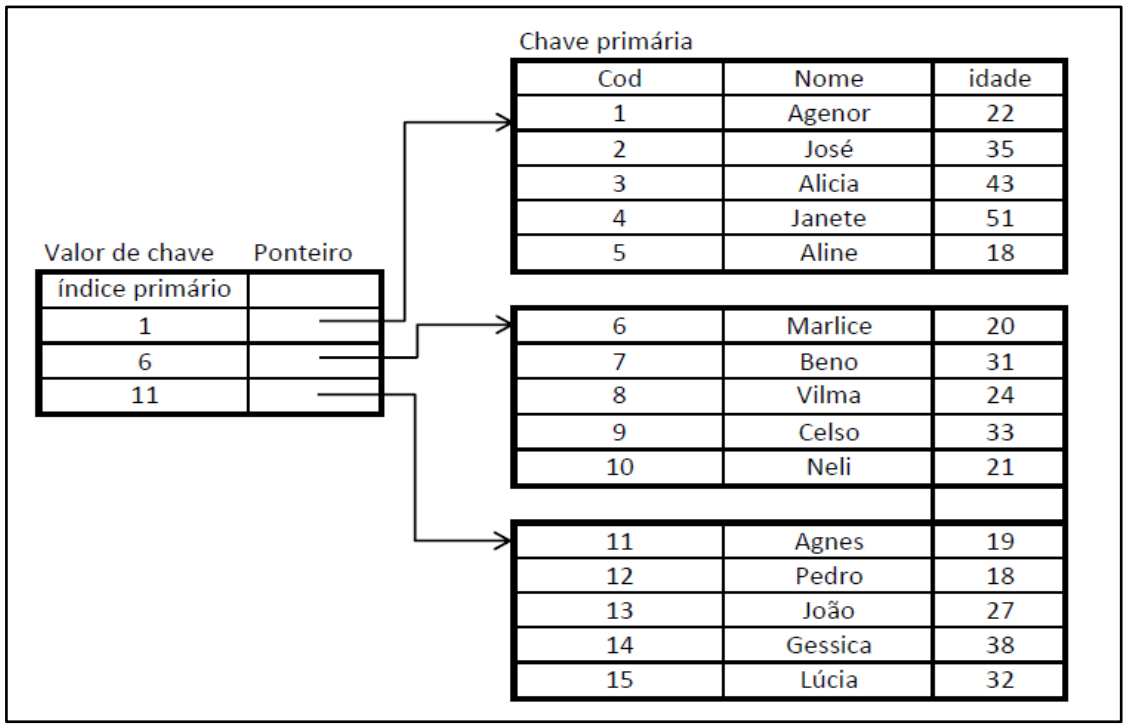
Elmasri e Navathe definem que

> Um **índice primário** é um arquivo ordenado cujos registros são de tamanho fixo com dois campos, e ele atua como uma estrutura de acesso para procurar acessar de modo eficiente os registros de dados em um arquivo. O primeiro campo é do mesmo tipo de dado do campo de chave de ordenação - chamado de **chave primária** - do arquivo de dados, e o segundo campo é um ponteiro para um bloco de disco (um endereço de

> bloco). Existe uma **entrada de índice (**ou **registro de índice)** no arquivo de índice para cada *bloco* no arquivo de dados. Cada entrada de índice tem o valor do campo de chave primária para o *primeiro* registro em um bloco e um ponteiro para esse bloco como seus dois valores de campo. (2011, p. 425).

A Figura 4 mostra um exemplo de um índice primário esparso, onde o número total de entradas no arquivo de índice é igual ao número de blocos de disco no arquivo de dados ordenados. O primeiro registro em cada bloco de arquivo de dados é o registro de âncora de cada bloco.

Figura 4 - Índice primário esparso



Fonte: Autor.

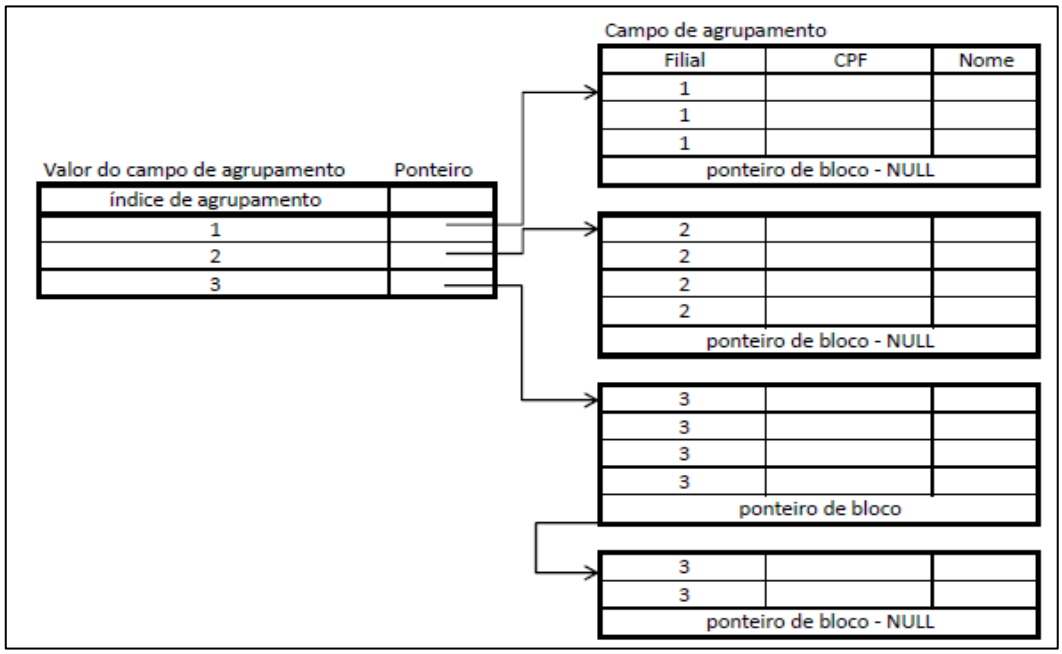
As principais vantagens de usar um índice primário é o fato de que pode-se realizar uma pesquisa binária no arquivo de índices, realizando menos acessos de blocos de disco do que uma pesquisa no arquivo de dados, e que o arquivo de índices ocupa um espaço bem menor em relação ao arquivo de dados por ter bem menos entradas de índices, e pelo fato de que cada entrada de índice geralmente é menor em relação a entrada de dados, por ter apenas duas colunas.

A principal desvantagem do índice primário é a inclusão e exclusão de novos registros, pois se incluirmos um registro no seu devido lugar definido pelo ordenamento no arquivo de dados, além de mover registros no arquivo de dados, provavelmente será necessário a alteração do arquivo de índices para alterar os registros âncora de alguns blocos.

Os índices de agrupamento são criados a partir de um campo do arquivo de dados que não tenham um valor distinto para cada registro e que contenham registros com o mesmo valor de campo, que formam os agrupamentos. Isso difere esse tipo de índice de um índice primário, que exige que o campo de ordenação do arquivo de dados tenha valor distinto para cada registro. Para cada valor distinto do campo de agrupamento há uma entrada no índice de agrupamento e o valor do ponteiro para o primeiro bloco no arquivo de dados que tem um registro com esse valor para seu campo de agrupamento, conforme Elmasri e Navathe (2011).

A Figura 5 mostra um exemplo de índice de agrupamento com um cluster de bloco separado para cada grupo de registros com o mesmo valor do campo de agrupamento.

Figura 5 - Índice de agrupamento com separação de blocos por agrupamento



Fonte: Autor.

A principal vantagem desse tipo de índice é que a inserção e exclusão é simplificada pelo fato de termos um bloco inteiro para cada valor do campo de agrupamento, não sendo necessária a reorganização constante do arquivo de índices para alterar os registros âncora.

2.1.3.3.2 Índices secundários

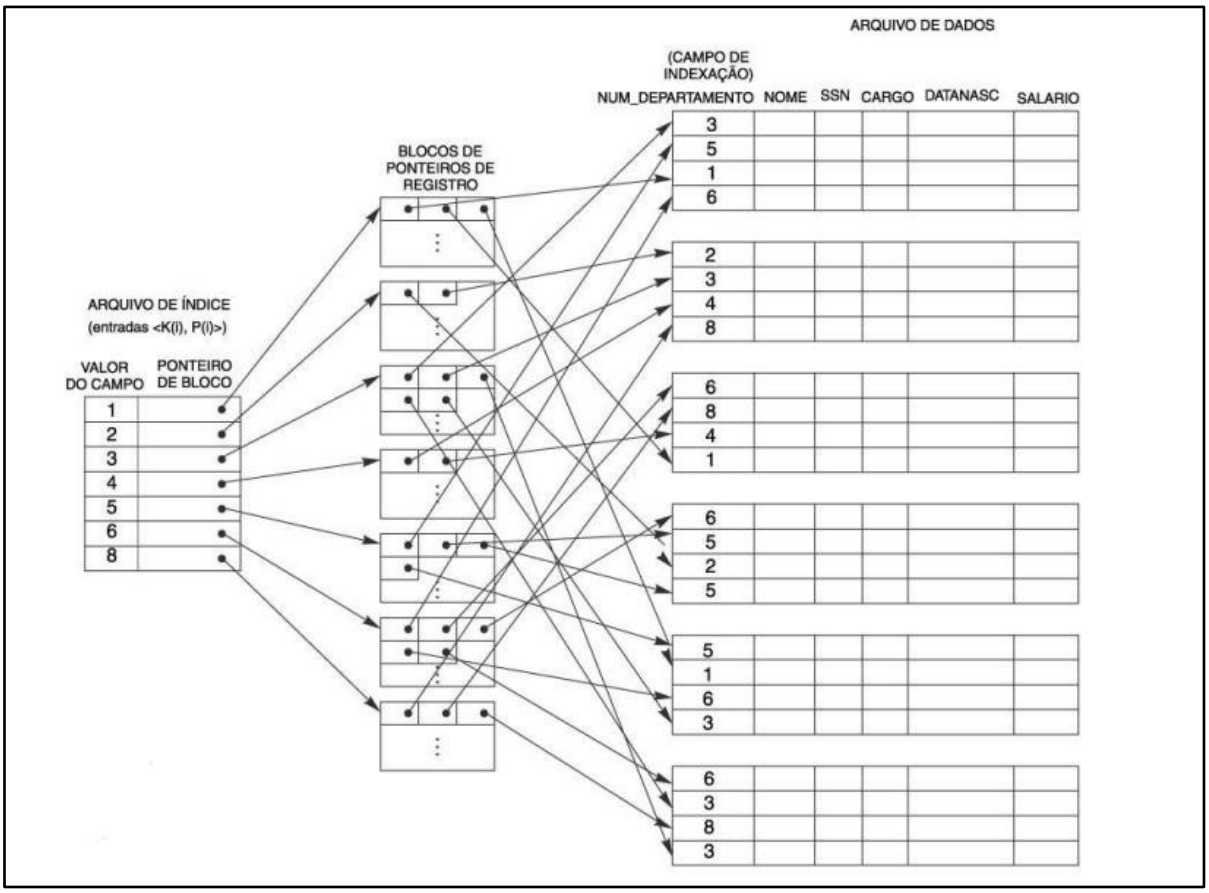
Conforme Elmasri e Navathe (2011), um índice secundário oferece um meio secundário para o acesso de um arquivo de dados em que já exista um acesso primário. Pode ser criado em

quaisquer campos não ordenados e que tenham valores distintos ou duplicados. O arquivo de índice continua sendo ordenado e com dois campos, o primeiro campo é do mesmo tipo de dado de algum campo não ordenado do arquivo de dados que seja um campo de índice, e o segundo campo é o ponteiro de bloco ou ponteiro de registro.

Silberschatz, Korth e Sudarshan (2012) definem que índices secundários são aqueles onde a chave de busca especifica uma ordem diferente daquela especificada pelo arquivo. Por esse fato, os índices secundários sempre são densos, pois se ele armazenar apenas alguns registros da chave de busca, os registros intermediários não serão encontrados se não for feita uma pesquisa no arquivo inteiro.

A Figura 6 mostra um exemplo de índice secundário com entradas de índices de tamanho fixo e um arquivo de dados com o campo de índice com valores duplicados, onde o endereço do ponteiro aponta para um bloco de disco que contém um conjunto de ponteiros, em que cada um desses aponta para um dos registros do arquivo de dados. Caso um valor ocorrer muitas vezes no arquivo de dados, de modo que seus ponteiros de registro não caibam em um único bloco de disco, uma lista ligada de blocos ou cluster é utilizada.

Figura 6 - Índice secundário



Fonte: Elmasri e Navathe (2011).

A principal vantagem de um índice secundário é a melhora do desempenho de consultas que usam chaves diferentes da chave de busca do índice agrupado. A principal desvantagem é que necessita de mais espaço de armazenamento devido a seu número de entradas e a sobrecarga para modificações de registros.

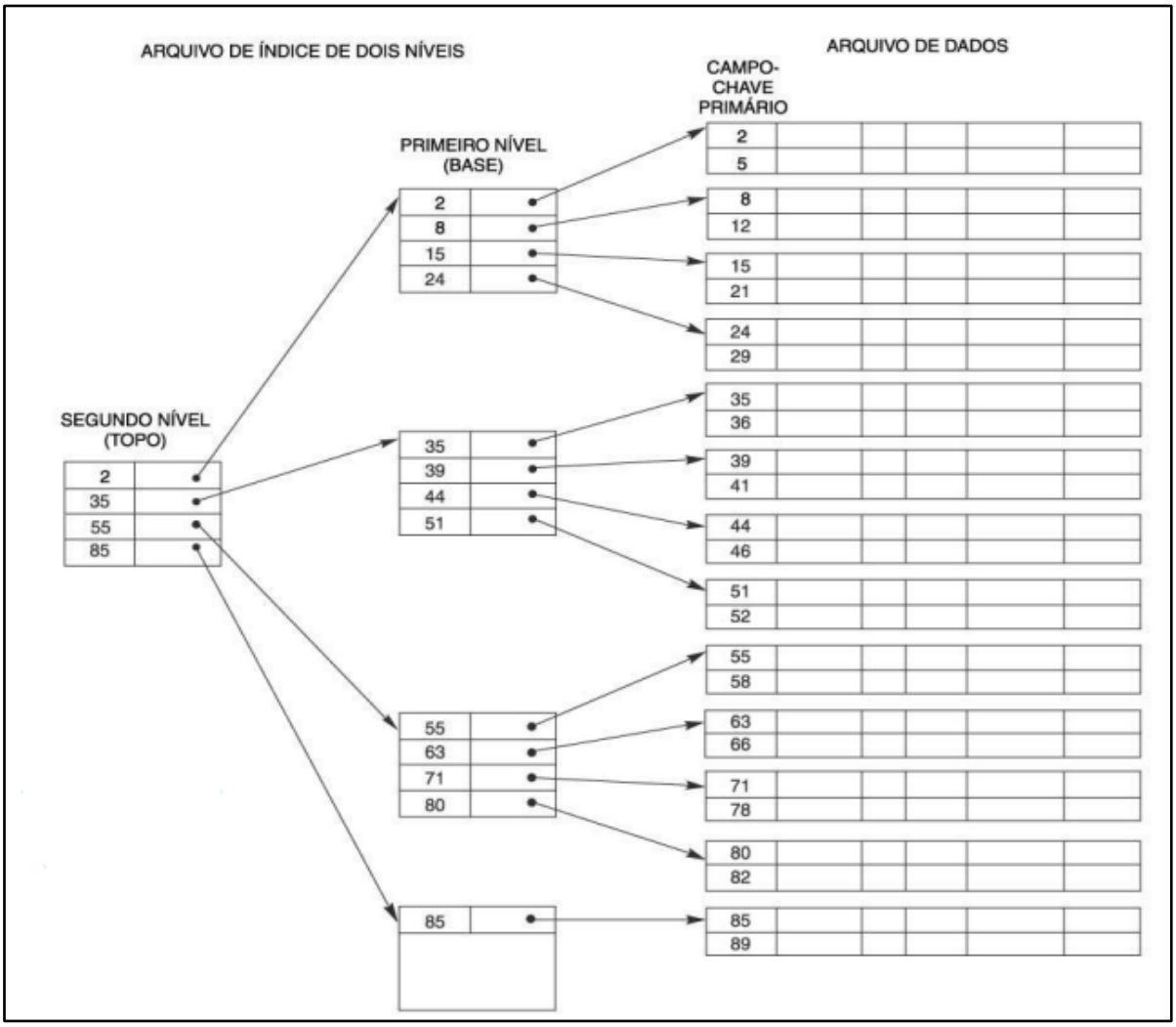
2.1.3.4 Índices multiníveis

Elmasri e Navathe (2011) definem que índices multiníveis podem ser classificados como um índice de um índice, onde o primeiro nível é um arquivo ordenado pela chave de indexação, valores distintos e entradas de tamanho fixo. Os demais níveis são índices primários sobre o índice do nível anterior. Nó último nível o índice deve ocupar apenas um bloco. O número de acessos nesse tipo de índice é um a cada nível e mais um ao arquivo de dados.

Para Silberschatz, Korth e Sudarshan (2012), um índice multinível é composto de um índice esparso (externo) sobre um índice agrupado (interno) onde as entradas de índice estarão sempre ordenadas. Para localizar um registro é realizada uma busca binária no índice externo, após encontrar o registro de valor menor ou igual, o ponteiro dele apontará para o bloco interno, onde será feita nova busca para encontrar o registro de valor menor ou igual que apontará para o arquivo de dados que contém o valor.

A Figura 7 representa um exemplo de índice multinível construído em um índice primário.

Figura 7 - Índice multinível



Fonte: Elmasri e Navathe (2011).

Elmasri e Navathe (2011) e Silberschatz, Korth e Sudarshan (2012) definem que índices de um único nível são vantajosos enquanto arquivos que podem ser mantidos na memória principal. Num arquivo de índice muito grande é necessário realizar vários acessos a disco, gerando perda de sua eficiência. Ao usar os índices multiníveis, se manusear um arquivo muito grande, pode-se criar níveis de índices adicionais para que possam ser carregados na memória principal, reduzindo ao máximo o número de acessos a disco e evitar acessos sequenciais aos arquivos.

Índices multiníveis estão bastante relacionados às estruturas de árvores, normalmente são implementadas usando estruturas de dados chamadas árvore B e sua variação, a árvore B+, que estão descritas a seguir.

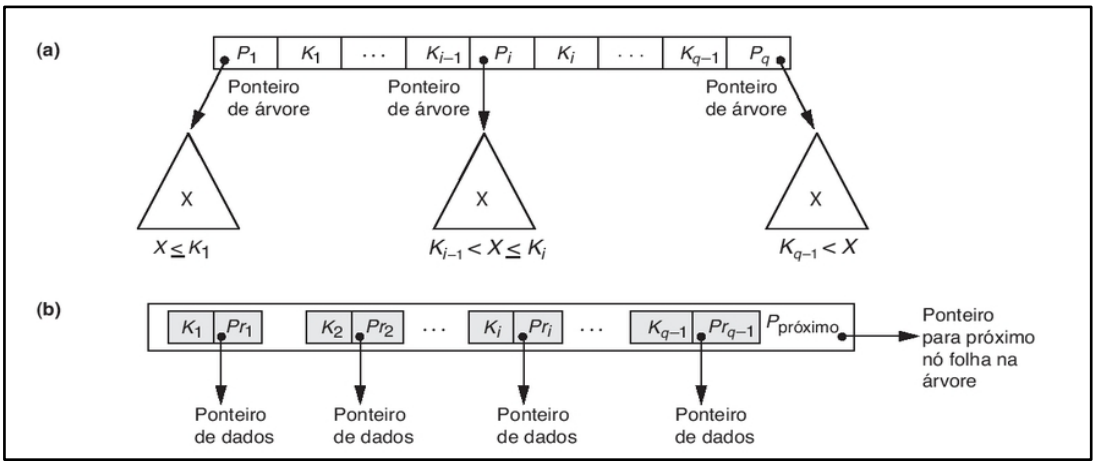
2.1.3.4.1 Índices de árvore B+

Rob e Coronel (2011) recomendam usar o índice de árvore B+ em tabelas que o valor das colunas é repetido poucas vezes. Definem o índice de árvore B+ como uma estrutura de dados ordenada, disposta como árvore descendente armazenada separadamente dos dados, sendo que as folhas de menor nível contém ponteiros para as linhas de dados.

Elmasri e Navathe (2011) definem que a estrutura de um índice de árvore B+ difere entre os nós internos e folha. Os nós folha mantém uma entrada para cada valor do campo de pesquisa acompanhado de um ponteiro de dados que aponta para o registro/bloco de registros caso o campo de pesquisa seja chave e normalmente são ligados. Num campo de pesquisa não chave o ponteiro aponta para um bloco com ponteiros para os registros do arquivo de dados.

A Figura 8 mostra a estrutura de uma árvore B+, a primeira parte demonstra os nós internos, já a segunda parte demonstra os nós folha.

Figura 8 - Estrutura de árvore B+

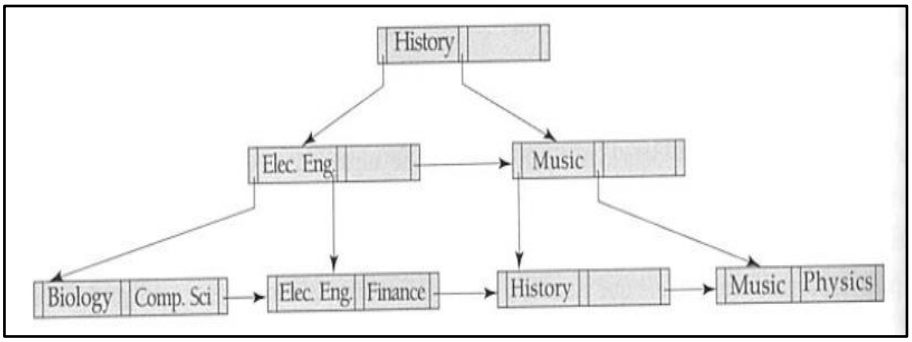


Fonte: Elmasri e Navathe (2011).

Uma variação de um índice de árvore B+ é a chamada árvore *B-link* de Lehman-Yao, esse tipo exige que cada nó, tanto interno como folha precisa manter um ponteiro para o seu irmão da direita.

A Figura 9 mostra o exemplo de um índice de uma *B-link* de Lehman-Yao.

Figura 9 - Índice de árvore *B-link*



Fonte: Silberschatz, Korth e Sudarshan (2012).

As principais vantagens do índice de árvore B+ é o desempenho com arquivos grandes, pelo fato de ter a estrutura de árvore balanceada que exige menos acessos ao disco. “As pesquisas nas árvores B+ são diretas e eficientes. Porém, a inserção e exclusão são um pouco mais complicadas, mas ainda eficientes.”. (SILBERSCHATZ; KORTH; SUDARSHAN, 2012, P. 333).

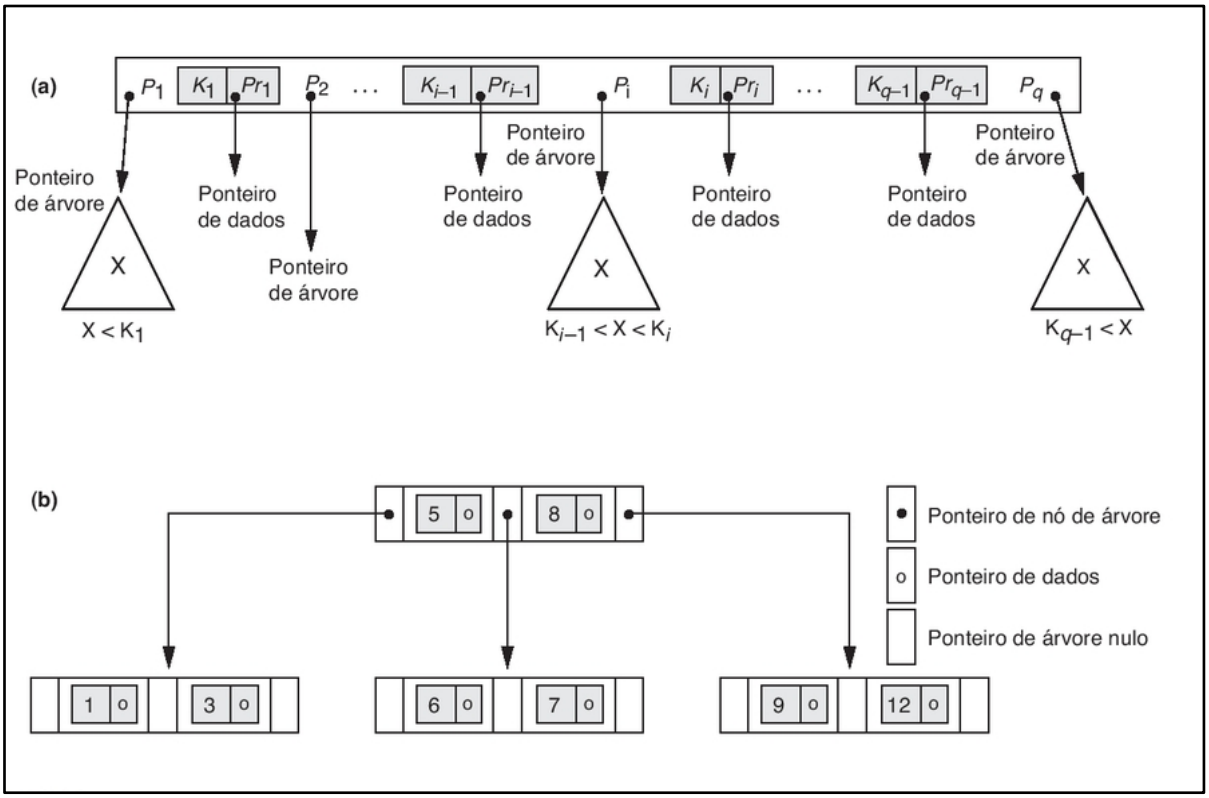
2.1.3.4.2 Índices de árvore B

Silberschatz, Korth e Sudarshan (2012) destacam que a principal diferença entre um índice de árvore B para um índice de árvore B+, é que em um índice de árvore B elimina a redundância de valores chaves, pois tem um ponteiro adicional para cada nó não folha que aponta para o registro de arquivos.

Elmasri e Navathe (2011) resumem a árvore B como uma estrutura de acesso multinível no formato de uma árvore balanceada em que cada nó precisa estar pelo menos com 50 % da capacidade, de ordem *p*, sendo que cada nó pode ter *p* - 1 valores de pesquisa.

A Figura 10 mostra a estrutura de um índice de árvore B, na primeira parte, um nó de uma árvore B. A segunda parte, mostra um índice de árvore B de ordem 3.

Figura 10 - Índice de árvore B



Fonte: Elmasri e Navathe (2011).

Pelo fato de um índice de árvore B armazenar um ponteiro adicional nos nós não folha, faz esse tipo de índice resultar em árvores geralmente maiores se comparadas a um índice de árvore B+ equivalente, já que a ordem costuma ser grande. Tendo uma árvore maior, a pesquisa nesse tipo de árvores muitas vezes é maior.

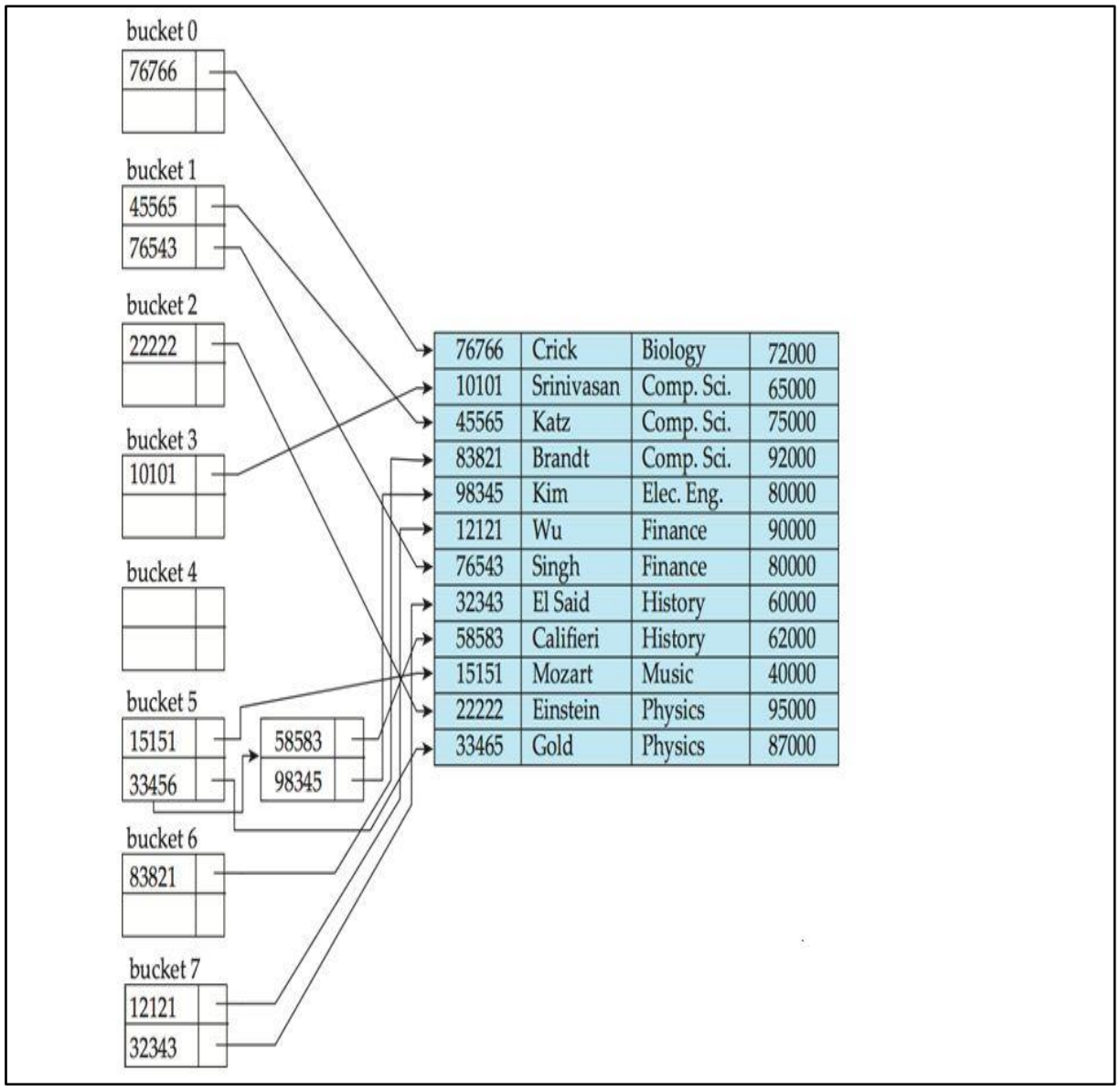
2.1.3.5 Índices de *hash*

"[...] Utiliza-se um algoritmo para criar um valor de *hash* a partir de uma coluna de chave. Esse valor aponta para uma entrada em uma tabela de *hash* que, por sua vez, aponta para a localização real da linha de dados.". (ROB; CORONEL, 2011, p.481).

Para Silberschatz, Korth e Sudarshan (2012), índices *hash* organizam as chaves de busca com ponteiros associados em uma estrutura de arquivo *hash*. Uma função de *hash* é aplicada sobre uma chave de busca para identificar um *bucket*, nele são armazenados a chave e os ponteiros associados. Caso um *bucket* fique cheio é criado um *bucket* de estouro.

A Figura 11 representa um índice *hash*, cada *bucket* possui um tamanho 2, porém um *bucket* possui mais registros associados, de modo que seja criado um *bucket* de estouro. A função *hash* usada é o módulo 8 sobre a soma dos dígitos.

Figura 11 - Índice hash



Fonte: Silberschatz, Korth e Sudarshan (2012).

O índice *hash* visto é chamado de *hash* estático ou linear. Conforme o banco de dados vai crescendo esse tipo de índice vai perdendo o seu desempenho, Silberschatz, Korth e Sudarshan (2012) citam algumas técnicas para lidar com isso:

- Escolher uma função *hash* com base no tamanho atual do arquivo. Irá perder desempenho conforme o banco cresce.
- Escolher uma função *hash* com base no tamanho futuro do arquivo. Pode ocorrer desperdício de espaço, mas ganho de desempenho.
- Escolher uma nova função conforme o crescimento do banco de dados e reorganizar toda a estrutura de *hash*. Operação custosa e é necessário proibir o acesso ao arquivo durante a reorganização.

Existem formas de *hashing* dinâmico que lidam com o crescimento do banco de dados, esse tipo de função é calculada, ou modificada de acordo com o crescimento do arquivo. Não será abordada nenhuma forma de *hashing* dinâmico nesta seção, pois as formas de tratamento mudam conforme o banco de dados que está sendo usado.

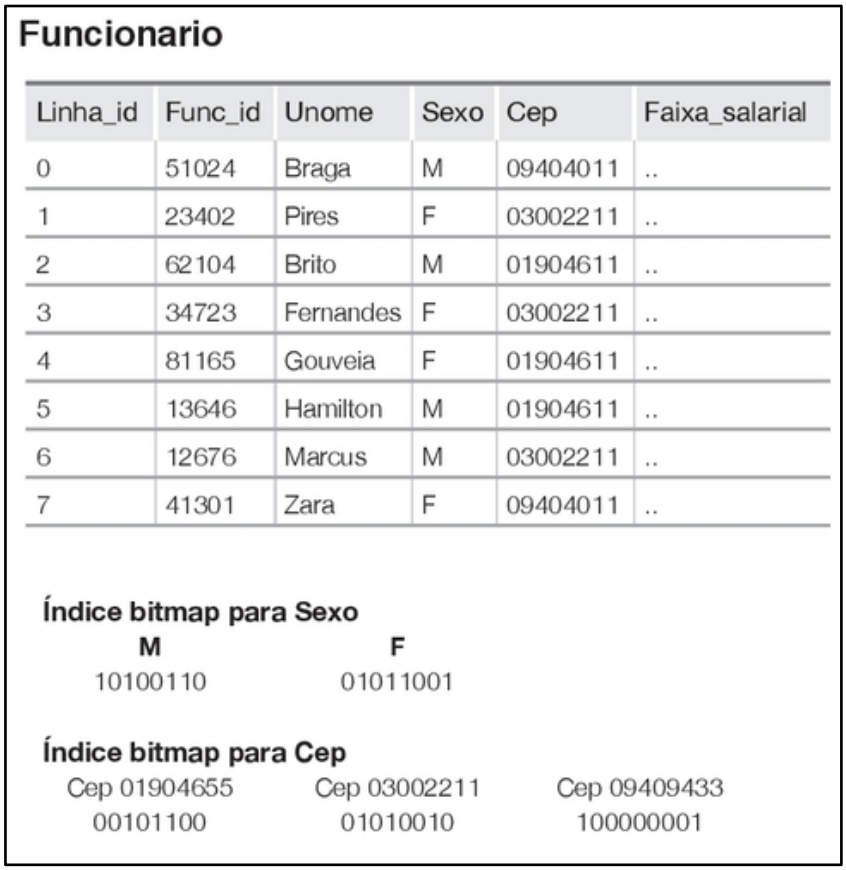
2.1.3.6 Índices *bitmap*

Um índice *bitmap* nada mais é do que um mapa de *bits*, Silberschatz, Korth e Sudarshan (2012) definem que são um tipo especializado de índice criado para consultas sobre múltiplas chaves e os classificam como um *array* de *bits*. Onde cada mapa de bits possui o mesmo número de bits do que o número de registros de uma relação.

Rob e Coronel (2011) definem que os índices de *bitmap* são utilizados quando temos tabelas com um grande número de linhas e os valores da coluna se repetem muitas vezes. Para Elmasri e Navathe (2011) o índice *bitmap* é um vetor de *bits* definido em um valor específico da coluna em uma relação. Criado normalmente quando temos colunas com poucos valores únicos e relações com muitas linhas.

A Figura 12 mostra um exemplo de índice *bitmap* para os campos “Sexo e Cep”. Para encontrar funcionários do Sexo = M e Cep = 01904655, realiza-se uma interseção dos bitmaps correspondentes ‘101001100’ e ‘00101100’.

Figura 12 - Índice *bitmap*



Fonte: Elmasri e Navathe (2011).

A vantagem do uso de índices *bitmap* é que o espaço de armazenamento necessário é extremamente pequeno comparado aos outros tipos e a recuperação é precisa, por se tratar de operações em *bits*. Pode-se citar como desvantagem a renumeração de linhas e deslocamento de bits quando houver uma exclusão de registros.

2.1.3.7 O processo de sintonia de índices

A sintonia de índices consiste em selecionar, criar e reconstruir estruturas de índices, que visam reduzir o tempo de processamento de cargas de trabalho. Para Rob e Coronel (2011, p. 480) “Os índices são fundamentais para acelerar o acesso aos dados, pois facilitam as buscas, classificação e utilização de funções agregadas e, até mesmo, de operações de junção.”.

Após a implementação é que se pode verificar e acompanhar o funcionamento do banco de dados, para verificar se o projeto físico do banco está em sintonia com a carga de dados em execução. Os índices criados precisam ser revisados para verificar se todos os que foram criados

estão sendo realmente utilizados com base nas operações que estão sendo realizadas, se existem consultas que estão demorando por falta de um índice ou até mesmo se existem índices que geram sobrecarga por serem frequentemente atualizados.

Grande parte dos SGBD's possuem um comando que demonstra o plano gerado para a execução de consultas e operações, onde é possível avaliar estruturas de índices que foram utilizados, ou onde deveria ser usado e não possui nenhuma estrutura de índice criada. Através da avaliação desse plano é possível diagnosticar a causa de parte dos problemas citados, conforme pesquisa de Elmasri e Navathe (2011).

Estruturas de índices são fundamentais para reduzir tempo de acesso aos dados, porém sua criação exige responsabilidade, pois elas podem ser a solução de muitos problemas de desempenho, se criadas corretamente, caso contrário, podem além de não trazer ganhos de desempenho, gerar mais operações de atualização no banco de dados. Por esses motivos a necessidade de sintonia de índices é de extrema importância para um bom funcionamento de aplicações que fazem uso do banco de dados.

2.2 TRABALHOS RELACIONADOS

O trabalho de Fuentes e Lifschitz (2016) propõem a criação automática de índices parciais para melhorar o desempenho de um sistema de banco de dados. Através de um algoritmo que analisa cada consulta relevante, e o conjunto de atributos indexáveis para os quais a criação de um índice parcial poderia influenciar o otimizador de consultas na geração de planos mais eficientes. Após chegar em uma proposta de índices parciais, é realizada uma análise de sintonia fina em função da seleção de uma configuração de índices parciais e índices completos. A seção 3.5 deste trabalho aborda os índices parciais, trazendo uma breve explicação e exemplificando o seu uso.

Rao e Krishna (2014) em seu trabalho de *SQL Performance Tuning* analisam e usam *hints* para mostrar como melhorar a performance do banco de dados, através de criação de índices, reescritas de consultas analisando o uso de *OR*, *IN*, *UNION*, *GROUP* e demais comandos em bancos ORACLE.

Magalhães et al. (2015) apresentou um comparativo sobre os dois algoritmos de otimização do SGBD do *PostgreSQL* e testou se sua utilização padrão é ideal em algumas situações. Foi utilizada a metodologia de gerar o plano de execução usando o comando

*EXPLAIN ANALY*SE para verificar qual algoritmo foi utilizado como padrão e o seu comportamento. Após, foram feitas as alterações necessárias nas configurações do *PostgreSQL* para a consulta ser realizada com o outro algoritmo. Por fim foi feita a conclusão com as métricas de número de linhas retornadas e o tempo de execução para as operações realizadas com cada algoritmo.

Telles, Galante e Dorneles (2011) desenvolveram um artigo que aplicou algumas técnicas de sintonia de SGBD no banco de dados *PostgreSQL*. Foram aplicadas operações de inserção, alteração, exclusão e seleção antes e após a aplicação das técnicas de sintonia de SGBD e a métrica avaliada para demonstrar se houve ganho de performance foi o tempo de execução de cada operação em milissegundos.

3 ÍNDICES NO *POSTGRESQL*

Os índices têm um papel muito importante no banco de dados, pois eles auxiliam a uma recuperação mais eficiente se utilizados da maneira correta e se for usado o tipo correto para determinado tipo de dado.

O *PostgreSQL* é um poderoso banco de dados relacional do código fonte aberto, com mais de 15 anos de desenvolvimento ativo. Pode ser instalado nos mais diversos sistemas operacionais como *Windows, Linux, MacOS,* Solaris, entre outros. Totalmente compatível com Atomicidade, Consistência, Isolação e Durabilidade (ACID). Possui recursos como Controle de Concorrência Multiversão (MVCC), *backups* em linha e *savepoints*, por exemplo. Atualmente está na sua versão 10, *PostgreSQL* (2017). Pelo fato da versão 10 ter sido lançada em 05 de outubro de 2017, o trabalho em questão será realizado na versão 9.3 do *PostgreSQL*, por se tratar de uma versão amplamente disseminada e estável.

3.1 TIPOS DE ÍNDICES DO *POSTGRESQL*

A Documentação do *PostgreSQL* 9.3.19 (2017), especifica que o *PostgreSQL* somente cria índices por *default* quando é criada uma chave primária, todos os índices criados por meio de chave primária ou através do comando *CREATE INDEX* são índices de árvores B. Também classifica os índices da seguinte forma:

- Índice *B-tree* ou índice de árvore B: É um tipo de índice baseado nas árvores B+, baseado nas árvores *B-link* de Lehman-Yao, segundo Silberschatz, Korth e Sudarshan (2012), esse tipo de índice baseado em árvore *B-link* está descrito na seção *2.1.3.4.1* e representado na figura 9. Definidos para lidar com consultas de igualdade, de faixa em dados passíveis de classificação em alguma ordem, comandos envolvendo *LIKE* e *ILIKE*;
- Índice *Hash*: São uma implementação de *hashing* linear, esse tipo de índice está descrito na seção 2.1.3.5 e representado na figura 11. Podem tratar apenas operações de igualdade simples. É criado através do comando *CREATE INDEX* nome *ON* tabela *USING HASH* (coluna). Esse tipo de índice não apresenta desempenho de pesquisa superior ao de árvores B, possuem tamanho e custos de manutenção maiores. Além

disso, é um tipo de índice incompatível com a recuperação de falhas. Dessa maneira seu uso é desencorajado.

- Índice *Generalized Search Tree* (GiST): Não são um único tipo de índice, diferentes técnicas de indexação podem ser implementadas. Os operadores específicos com os quais esse tipo de índice pode ser usado variam de acordo com a estratégia de indexação usada. Podem ser utilizados para indexar tipos de dados geométricos. Silberschatz, Korth e Sudarshan (2012) o definem um método balanceado de acesso estruturado por árvore.
- Índice *Generalized Inverted Index* (GIN): índices invertidos que podem manipular valores que contém mais de uma chave. Também suporta várias estratégias de indexação definidas pelo usuário. Pode ser utilizado para indexação de documentos para pesquisa de texto completo. Para Silberschatz, Korth e Sudarshan (2012), o índice GIN interpreta chaves de índices e de pesquisa como conjuntos. Quando avalia uma pesquisa, identifica as chaves de índices que se sobrepõem a chave de busca e calcula um bitmap indicando quais elementos pesquisados são membros da chave de índice.

3.2 ÍNDICES DE MÚLTIPLAS COLUNAS

São índices criados sobre mais de uma coluna. Somente índices de árvore B, GiST e GIN suportam índices de múltiplas colunas. Podem ser especificadas até 32 colunas, porém é recomendado que sejam feitos testes com medições de tempo verificando se é viável a sua criação, visto que na maioria das vezes uma coluna é o suficiente. Um índice com mais de três colunas tem uma grande chance de não ser útil, a não ser que os dados sejam muito peculiares, conforme Documentação do *PostgreSQL* 9.3.19 (2017).

3.3 ÍNDICES ÚNICOS

A Documentação do *PostgreSQL* 9.3.19 (2017), define que quando um índice é declarado como único, não poderá existir mais de uma linha com valor igual. Somente os índices de árvore B podem ser declarados como únicos. Quando numa tabela é criada uma chave

primária ou é definida uma regra de unicidade o *PostgreSQL* cria automaticamente um índice único.

3.4 ÍNDICES EM EXPRESSÕES

Pode ser definido uma função ou uma expressão escalar computada sobre uma ou mais colunas da tabela, como uma coluna de índice. Um exemplo pode ser usar a função *lower* para não diferenciar letras maiúsculas de minúsculas, CREATE INDEX idx_lower_nome ON tabela (*lower*(coluna)). O custo para manutenção desse tipo de índice é relativamente alto, devendo ser utilizados somente se o número de consultas for muito grande, e o número de atualizações pequeno, conforme Documentação do *PostgreSQL* 9.3.19 (2017).

3.5 ÍNDICES PARCIAIS

São criados sobre um subconjunto de dados da tabela, sendo que esse subconjunto é definido por uma expressão condicional. O índice contém apenas entradas para as tuplas que satisfazem a expressão condicional. Seu uso é adequado em casos que tenhamos uma coluna com uma grande quantidade de ocorrências de um número pequeno de valores. Índices parciais tem um custo menor de manutenção, visto que grande parte dos registros de uma tabela não participam dele.

Como um exemplo de uso pode-se citar uma tabela que contenha tanto pedidos faturados como os não faturados, e constantemente tem-se pesquisas sobre os pedidos não faturados, pode-se criar um índice parcial sobre os pedidos não faturados, conforme Documentação do *PostgreSQL* 9.3.19 (2017). O trabalho de Fuentes e Lifschitz (2016) apresentado na seção 2.2 apresenta outro caso do uso de índices parciais.

3.6 FATORES QUE COMPROMETEM O USO DE ÍNDICES

A Documentação do *PostgreSQL* 9.3.19 (2017), define alguns cuidados que devem ser tomados ao criar comandos de consultas SQL, para evitar que um índice que foi criado para otimizar uma consulta seja descartado para a operação em questão, ou seja ineficaz:

- Ao utilizarmos o operador *LIKE*, o arquivo de índice devidamente criado só será considerado se estiver na forma *LIKE* ‘abc%’, mas não em *LIKE* ‘%abc’;
- Índices com várias colunas só podem ser utilizados se as cláusulas envolvendo as colunas indexadas estiverem conectadas com *AND*;
- Um índice de árvore B é mais eficiente se há restrições nas colunas mais à esquerda;
- Um índice GiST é extremamente ineficiente se sua primeira coluna tiver somente alguns valores distintos, mesmo que nas outras colunas haja diversos valores distintos.

Já Milani (2008), traz algumas técnicas para otimizar o uso dos índices:

- O *PostgreSQL* utiliza melhor os índices quando estão de forma isolada em expressões SQL. Uma coluna que esteja indexada, porém na expressão é usada como ‘coluna - 1’, faz com que o índice não seja usado, pois será necessário recalcular todos os valores da coluna indexada;
- Em uma união, utilizar o índice que descarta o maior número de linhas por primeiro, ou que tenha menos linhas, pode trazer benefícios para a operação, mesmo que o otimizador tem inteligência programada para realizar esse procedimento;
- Evitar o uso da cláusula *IN*, pois ela não executa corretamente a utilização de índices ao realizar sua busca interna, podendo reduzir de forma significativa o desempenho de determinada consulta. Sugere-se o uso do *OR* no seu lugar, caso seja possível modificar o comando;
- O uso dos sub-selects em um determinado comando também prejudica o desempenho do servidor. Deverão ser realizados testes com medição de tempo de execução para verificar se será necessária e viável, a reescrita do comando em questão;
- O *PostgreSQL* e seu SGBD se baseiam em estatísticas para gerenciar e otimizar comandos SQL, inclusive para decisão de uso de índices. A atualização das estatísticas no *PostgreSQL* ocorre através da execução do comando *ANALYSE*, caso essa

atualização não seja realizada com frequência, o otimizador está trabalhando com dados incorretos, e muito provavelmente as suas escolhas não serão as mais otimizadas;

- O *PostgreSQL* realiza as atualizações nos arquivos de índices de forma automática, porém, se for necessário realizar um número muito grande de inserções ou exclusões em um determinado momento, poderá ser vantajoso desabilitar os índices e habilitá-los somente no final da operação, assim o arquivo de índices é atualizado somente uma vez, e não a cada operação.
- Comparações usando *NOT* ou *NULL* podem invalidar o uso do índice. Quando se busca valores nulos não é possível utilizar um índice, pois esse tipo de valor só pode ser encontrado fazendo uma busca sequencial no arquivo.
- A condição *AND* é avaliada da esquerda para a direita, por isso, se for possível determinar a expressão menos provável ou em caso de igualdade, a expressão menos complexa, para colocar ela por primeiro na condição. Ao contrário do *AND*, ao usar a condição *OR*, a expressão mais provável deve ser colocada por primeiro. Isso porque o uso da condição *OR* gera testes adicionais se a primeira expressão for falsa, enquanto o uso da condição *AND* gera novos testes somente se a primeira expressão for verdadeira.

3.7 RECOMENDAÇÕES DE USO DE ÍNDICES

A Documentação do *PostgreSQL* 9.3.19 (2017), define que índices devem ser utilizados quando o número de dados é relativamente grande em uma tabela, pois as consultas sequenciais em uma tabela com poucos registros com certeza serão mais eficientes que o custo da criação e manutenção de um índice. A análise de determinado comando com o *EXPLAIN* é fundamental para verificar as etapas que estão sendo realizadas no processamento de uma consulta e suas estimativas de custo, é essencial que se faça testes com esse comando para verificar a estimativa de custo de determinada operação de consulta com e sem o índice.

Milani (2008) sugere o uso de índices nos seguintes casos:

- *Where*, *join*: Quando as colunas indexadas são chave de união do parâmetro *join*, ou parâmetros de filtro para o *where*, as consultas são encerradas quando for detectado que determinado resultado não pode mais ser encontrado.

- *Group by, order by*: As colunas indexadas já ordenam e separam os valores da coluna, encerrando a consulta assim que for detectado que o resultado não pode mais ser encontrado.
- *Distinct*: Esse parâmetro pode em determinadas ocasiões, ter comportamento similar ao *group by*, podendo sua velocidade de processamento melhorar se as colunas em questão ou outras combinadas com um parâmetro *where* estiverem indexadas.

O uso de índices deve ser evitado em tabelas que contenham pouca distinção dos valores, pois uma consulta sobre esse tipo de dados costuma trazer a maior parte de seus registros, assim como, a questão do uso de índices em tabelas relativamente pequenas. Geralmente nestes casos, o custo de criação e manutenção do índice é maior que uma busca sequencial na tabela em questão.

3.8 CLASSES DE OPERADORES

No *PostgreSQL* existem as classes de operadores, elas identificam os operadores que serão utilizados pelo índice para determinada coluna. Geralmente a classe de operadores padrão é suficientemente eficiente, porém para determinadas situações e tipos de dados uma determinada classe de operadores pode ser mais eficiente.

A Documentação do *PostgreSQL* 9.3.19 (2017), a classe *varchar_pattern_ops* por exemplo, fornece suporte a índices *B-tree* do tipo *varchar*, e sua principal diferença a classe padrão é que seus valores são comparados estritamente caractere por caractere, não seguindo regras de intercalação. Esse tipo de funcionamento torna esse tipo de operador adequado para uso com expressões regulares como o LIKE.

4 BENCHMARK

Conforme Seng, Yao e Hevner (2005), *benchmark* pode ser definido como um programa usado para testar o desempenho de determinado software, hardware ou sistema. E um *benchmark* de banco de dados pode ser definido como um conjunto de instruções executáveis, usadas para mensurar e conferir através da execução de experimentos controlados o desempenho relativo e quantitativo de sistemas de banco de dados. *Benchmarks* podem ser classificados em sintéticos ou empíricos:

- *Benchmark* sintético: emulam aplicações típicas para um problema pré-determinado e criam uma carga de trabalho sintética correspondente.
- *Benchmark* empírico: utilizam informações reais em operações de teste.

Testes com implementações e dados reais são considerados ideais, porém o custo de implementação e uso faz com que geralmente se use o *benchmark* sintético. Alguns dados do *benchmark*, como a taxa de transferência, tempo por unidade de trabalho, o tempo de resposta, relação de preço, o tamanho equivalente do banco de dados e as interações da *Web* por segundo, servem para prever e avaliar o desempenho de determinado sistema, auxiliando em tomadas de decisões como planejamento de capacidade, detecção de gargalos e aquisição de softwares, conforme Gray (1993).

4.1 SOFTWARES DE BENCHMARK

A ferramenta *BenchmarkSQL* é um software livre desenvolvido em Java que utiliza drivers JDBC para se comunicar com banco de dados como o *PostgreSQL*, *EnterpriseDB* e *Oracle*. A ferramenta utiliza o padrão *Transaction Processing Performance Council* (TPC-C) que é considerado padrão para avaliar o desempenho *Online Transaction Processing* (OLTP), que testa a maior parte das funcionalidades de um banco de dados, conforme Benchmarksql (2017).

Swingbench é um software livre desenvolvido em Java por Dominic Giles, que executa e monitora cargas de dados em banco de dados *Oracle*. Pode ser instalado em *Windows*, *Linux*

e outros sistemas operacionais. Principais indicadores de performance apresentadas são a média de transações por minuto e média de transações por segundo, conforme Giles (2010).

O Quadro 1 traz um comparativo dos softwares de *Benchmark*, apresentando sua compatibilidade, os pontos avaliados e principais métricas utilizadas.

Quadro 1 - Comparação de softwares de *Benchmark*

Software	Compatibilidade	Pontos avaliados	Métricas
BenchmarkSQL	*EnterpriseDD, Firebird, Oracle e PostgreSQL*	Operações de leitura e escrita	Número de transações por minuto Número de transações correntes
Swingbench	*Oracle*	Operações de leitura e escrita	Número transações por segundo/minuto

Fonte: Autor

5 PLANO DE EXECUÇÃO

O *PostgreSQL* monta um plano de execução para cada consulta que for solicitada. Nesse plano gerado pode-se verificar o tipo da busca que foi planejada e se haverá uso de índices. Para realizar o gerenciamento do banco de dados *PostgreSQL*, será usado o sistema de gerenciamento *PgAdmin* III, que está disponível para sistemas *Unix* e *Windows* sob os termos da licença do *PostgreSQL*, conforme The Pgadmin Development Team (2002 - 2016).

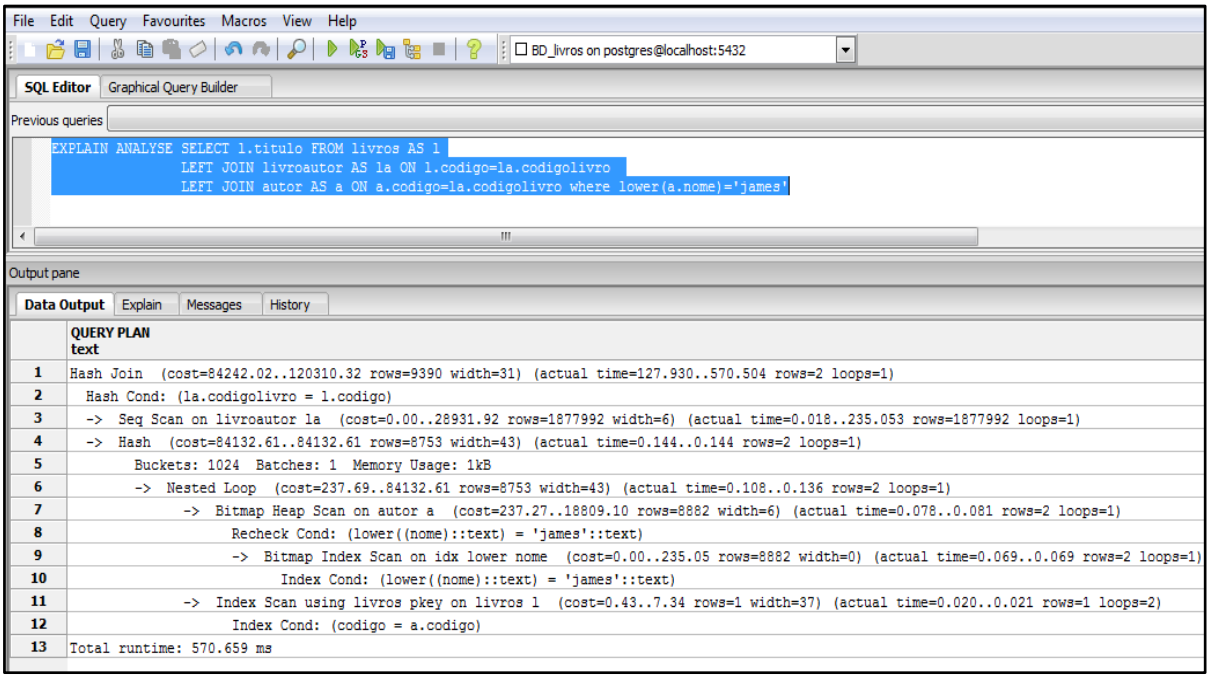
O *PgAdmin* III mostra o plano de execução montado pelo servidor em forma de texto e na forma gráfica. Nas próximas seções será abordado como esse plano pode ser verificado e quais as principais informações que apresenta, conforme Documentação do *PostgreSQL* 9.3.19 (2017).

5.1 PLANO DE EXECUÇÃO TEXTUAL ATRAVÉS DO COMANDO *EXPLAIN*

Comando responsável por mostrar o plano de execução gerado pelo planejador em forma de texto na guia Data Output. Demonstra como as tabelas são varridas, quais algoritmos de junção são utilizados nas uniões de tabelas e o custo estimado de execução do comando, bem como o número estimado de linhas retornadas.

O comando *EXPLAIN* não executa a consulta, apenas a planeja. A fim de realizar a execução deve ser utilizada a opção *ANALYSE*. Ao adicionarmos essa opção ao comando *EXPLAIN*, algumas informações importantes são adicionadas: tempo total decorrido em milissegundos e o número real de linhas retornadas. A Figura 13 mostra um exemplo de uso do comando *EXPLAIN ANALYSE*.

Figura 13 - Comando *EXPLAIN ANALYSE*



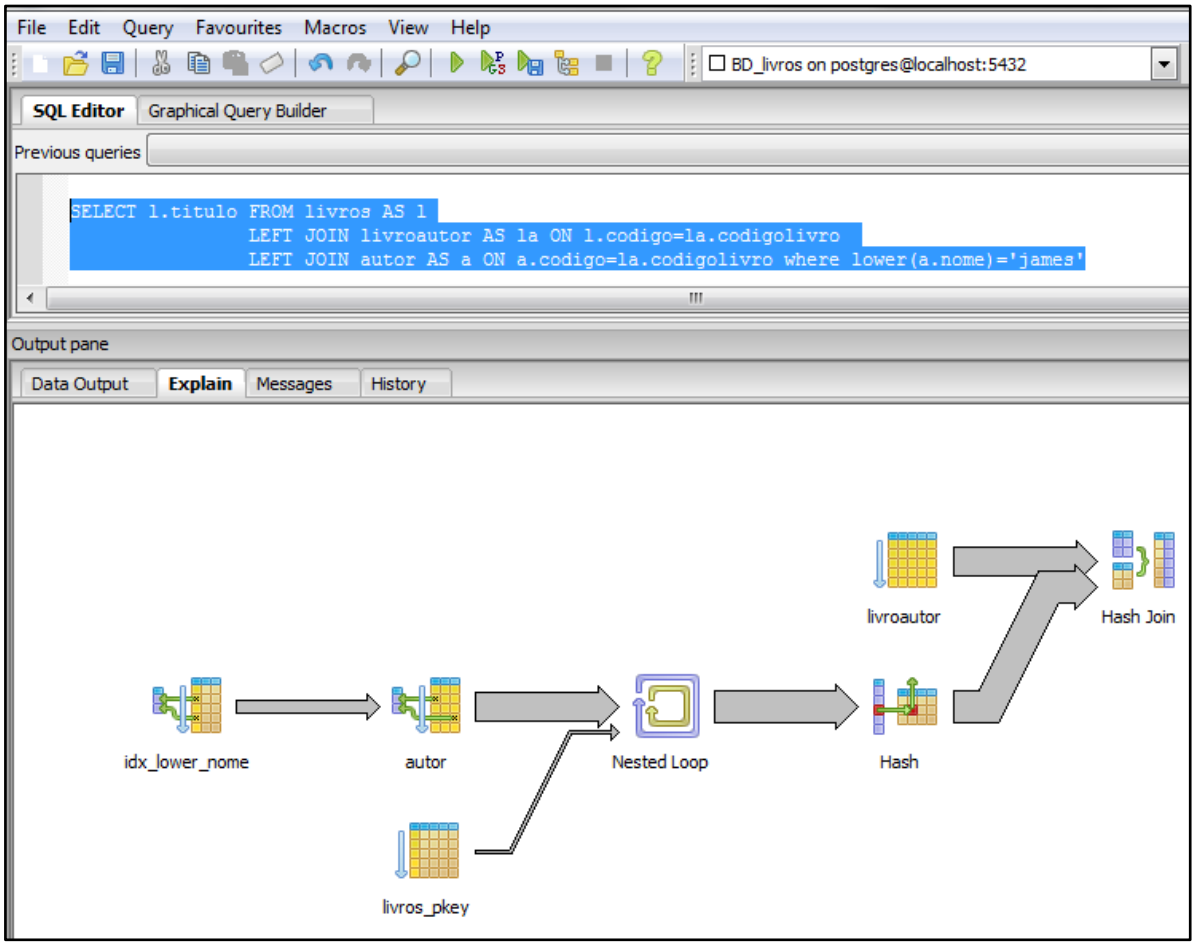
Fonte: Autor.

5.2 PLANO DE EXECUÇÃO EM MODO GRÁFICO

Conforme The Pgadmin Development Team (2002 - 2016), é possível analisar o plano de execução gerado pelo servidor no formato gráfico, o que facilita a compreensão dos passos executados em uma determinada operação realizada no banco de dados.

O plano de execução gráfico pode ser gerado executando uma consulta selecionada através do botão F7, ou selecionando no menu *Query – Explain*. A Figura 14 mostra um exemplo de geração do plano gráfico.

Figura 14 - Plano de execução gráfico



Fonte: Autor.

Como é possível observar, o plano de execução retorna muitas informações, para realizar uma correta análise dessas informações é necessário entender como ele funciona. Na próxima seção será apresentada uma explicação mais detalhada sobre o funcionamento e suas etapas.

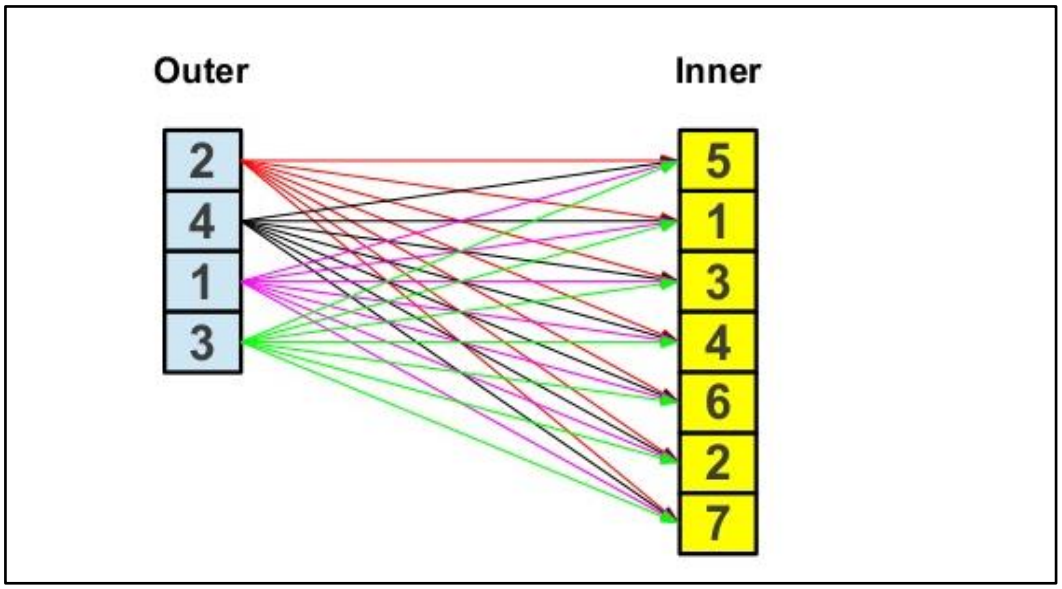
5.3 FUNCIONAMENTO DO PLANO DE EXECUÇÃO

Ao gerar as possibilidades de planos para determinada operação, o planejador começa gerando os possíveis planos para varrer individualmente cada relação no comando. O plano da varredura sequencial é sempre criado, caso haja algum arquivo de índices é criado outro plano a fim de utilizá-lo. Alguns exemplos dessas varreduras são *Seq Scan*, para buscas sequenciais, e variações de *Index Scan*, para varreduras realizadas com índices.

Após a geração dos possíveis planos para varrer as relações únicas, são criados os planos para juntar as relações. Existem três estratégias de junção disponíveis no *PostgreSQL*:

- *nested loop join*: Conhecida também como junção de laço aninhado, para cada linha encontrada na relação da esquerda, a relação da direita é varrida uma vez. A Figura 15 ilustra um exemplo dessa estratégia de junção.

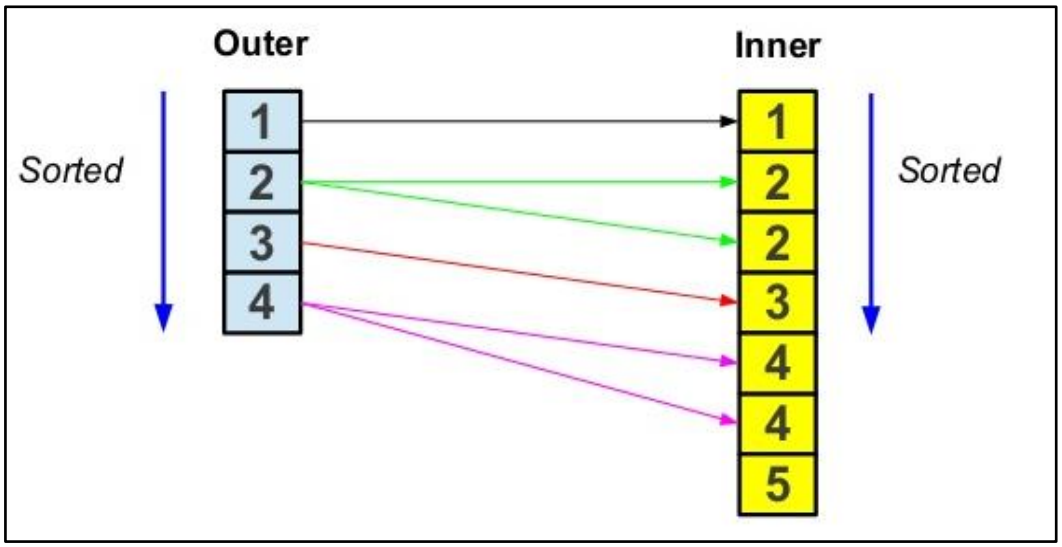
Figura 15 - N*ested loop join*



Fonte: https://pt.slideshare.net/fabriziomello/planejador-de-consultas-do-*postgresql*(Adaptado)

- *merge join*: Junção por classificação e mesclagem. Cada relação é classificada pelo atributo de junção. As linhas correspondentes são combinadas para formar as linhas juntadas enquanto as duas relações são varridas em paralelo. A Figura 16 ilustra um exemplo dessa estratégia de junção.

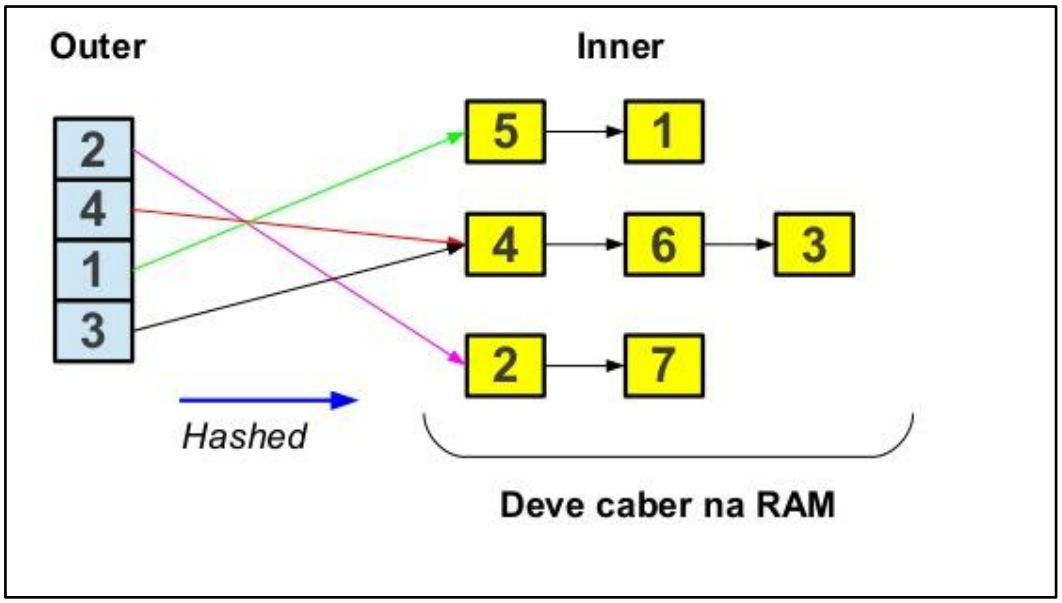
Figura 16 - *Merge join*



Fonte: https://pt.slideshare.net/fabriziomello/planejador-de-consultas-do-*postgresql*(Adaptado)

- *hash join*: O primeiro passo é varrer a relação da direita e carregar numa tabela de *hash*, utilizando os atributos de junção como chaves de *hash*. O segundo passo é varrer a relação da esquerda, e os valores apropriados de cada linha encontrada são utilizados como chave de *hash*, para localizar as linhas correspondentes na tabela. A Figura 17 ilustra um exemplo dessa estratégia de junção.

Figura 17 - Hash join



Fonte: https://pt.slideshare.net/fabriziomello/planejador-de-consultas-do-*postgresql*(Adaptado)

Caso o comando contenha mais de duas relações, uma árvore de passos de junções é construída. O planejador examina as diferentes possibilidades de sequência de junção para descobrir a menos custosa.

Após a análise das junções, são agregados os planos para classificações ou funções de agregação. A maioria desses planos possuem capacidade de desprezar linhas através de condições *booleanas* ou cálculos de funções. Esse é o processo resumido de análise de um plano de execução montado pelo planejador do *PostgeSQL*, necessário para entender, interpretar e avaliar a necessidade de aplicação da sintonia de índices.

O plano de execução é gerado através de estatísticas sobre os dados armazenados nas tabelas, para manter essas estatísticas atualizadas é necessário executar periodicamente o comando *ANALYSE* a fim de que o planejador tome decisões sobre dados confiáveis.

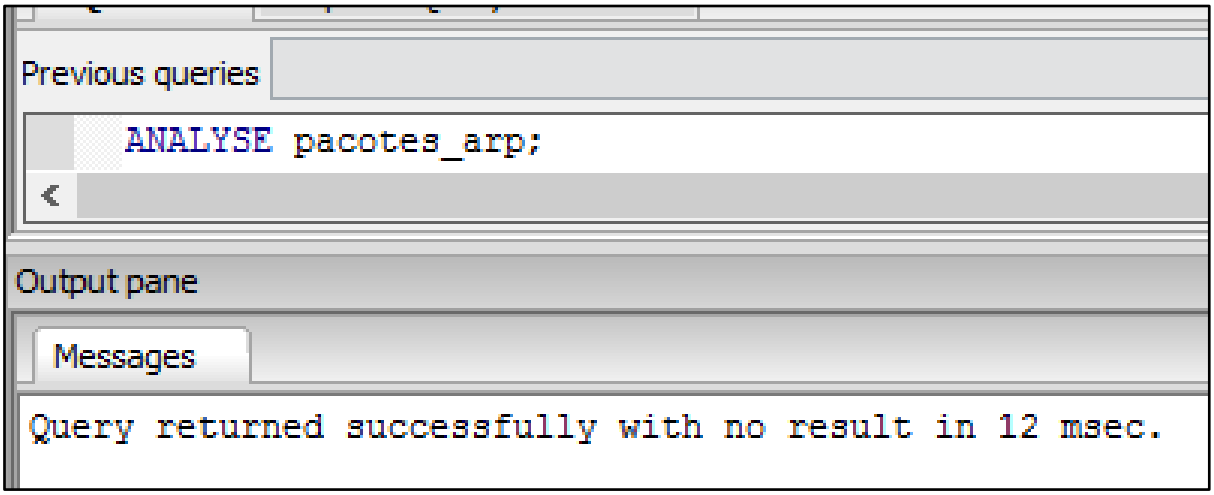
5.4 COMANDO *ANALYSE*

Realiza a coleta de estatísticas sobre o conteúdo das tabelas do banco de dados e armazena em tabela específica. Pode ser realizada sobre uma determinada coluna, sobre a tabela inteira, ou então sobre todo o banco de dados. Os parâmetros informados podem ser referentes a uma tabela ou coluna desejada, caso não seja especificado nenhum parâmetro, ele realiza a operação sobre o banco de dados inteiro.

A sua execução é recomendada sempre que tiver uma atualização considerável em determinada tabela e antes do uso de índices. Uma boa estratégia é executar o comando diariamente em horário de pouca utilização.

Entre as estatísticas coletadas pelo *ANALYSE*, está uma lista dos valores mais comuns de cada coluna e um histograma da distribuição dos dados em cada coluna. Em tabelas muito grandes, apenas amostras aleatórias são coletadas, em vez de percorrer todas as linhas da tabela. A Figura 18 mostra o uso do comando sobre uma tabela específica, a tabela “pacotes-arp”.

Figura 18 - Comando *ANALYSE*



Fonte: Autor.

A coleta das estatísticas também pode ser realizada associada a outro comando: o *VACUUM*.

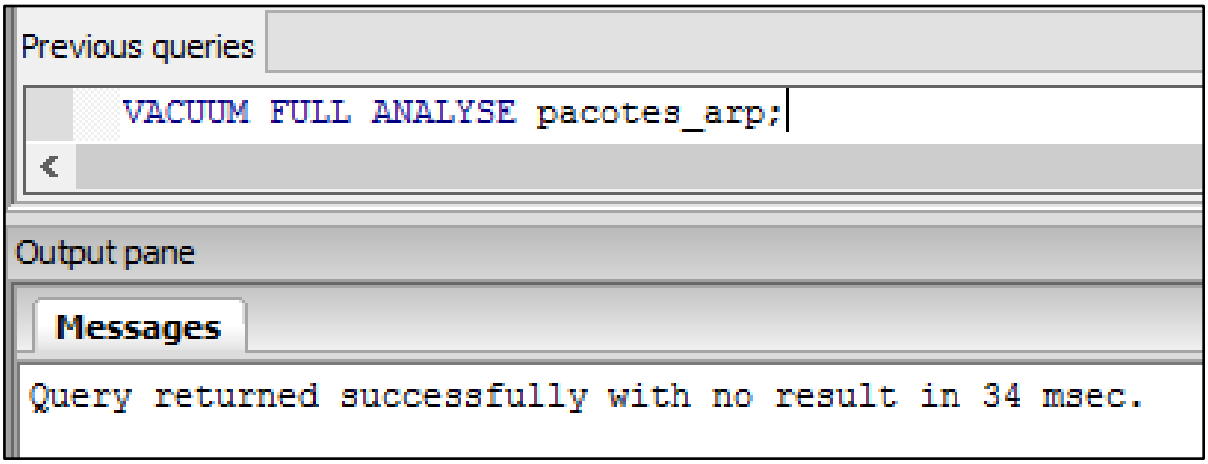
5.5 COMANDO *VACUUM*

O comando realiza a limpeza de um banco de dados. Essa limpeza recupera a área de armazenamento das tuplas que foram excluídas. Quando uma remoção ocorre no *PostgreSQL*, ela não ocorre fisicamente até a execução do comando *VACUUM*. Por isso é muito importante executar o comando periodicamente, principalmente se a tabela passa por constantes atualizações.

O comando *VACUUM* pode ser executado em conjunto com o comando *ANALYSE*, na forma de *VACUUM ANALYSE*, assim, é executada primeiro a limpeza e após a coleta das estatísticas. Essa combinação é bastante útil após atualizações.

Um exemplo de utilização é mostrado na Figura 19, onde é realizada uma limpeza completa com o *VACUUM* utilizando o parâmetro *FULL*, associado ao comando *ANALYSE*.

Figura 19 - Comando *VACUUM*



Fonte: Autor.

Para realizar uma limpeza completa é necessário utilizar o parâmetro *FULL*, que além da recuperação do espaço também inclui a movimentação de tuplas entre blocos de disco para compactar a área utilizada, porém é mais demorada e bloqueia a tabela em modo exclusivo.

6 PROPOSTA DE IMPLEMENTAÇÃO

No decorrer deste estudo foram apresentadas as principais formas de sintonia aplicáveis em um banco de dados, sendo que, a mais explorada foi a sintonia de índices. Foi possível verificar que, para aplicar esse tipo de sintonia é necessário conhecer bem os dados manipulados. Um dos principais motivos para o uso da sintonia de índices é evitar as buscas sequenciais em operações de consulta, pois se há uma grande quantidade de dados e a consulta retornar poucos resultados haverá uma melhora significativa no tempo de execução.

6.1 ESPECIFICAÇÃO DO AMBIENTE UTILIZADO NO ESTUDO DE CASO

O estudo de caso foi realizado em um *Desktop* com processador *Intel Core* I5 3.0 *Gigahertz* (GHz), com 8 *Gigabyte* (GB) de memória *Random Access Memory* (RAM), disco rígido S*olid State Drive* (SSD) *Kingston*, modelo SV300S37A, com 120 GB e velocidade de leitura e gravação de 450MB/s. O *Desktop* contém a instalação *Windows* 10 Pro na sua instalação padrão, com a instalação do SGBD *PostgreSQL* 9.3 juntamente com seu software de administração, o *pgAdmin* III.

6.2 DEFINIÇÃO DA PROPOSTA

A proposta de implementação deste estudo consiste em realizar operações de consultas usando o comando *EXPLAIN ANALYSE* para mostrar o plano de execução gerado pelo planejador do *PostgreSQL*. Definiram-se alguns critérios considerados relevantes para análise dessa execução, que poderão influenciar no uso de índices:

- O caminho percorrido. A verificação do caminho planejado é um passo muito importante para a aplicação da sintonia de índices, pois é possível verificar se haverá busca sequencial. Um dos principais objetivos da criação de índices é evitar buscas sequenciais;
- O número de linhas retornadas. Analisar o número de linhas retornadas é essencial para o planejamento da criação de um índice, caso o número de linhas retornadas se

aproximar do número total de linhas, a criação de um índice provavelmente não será muito vantajosa;

- O tempo de execução para o comando fornecido. Nos comandos em que temos um tempo de execução muito grande, é necessário verificar se alguma técnica de sintonia é aplicável para melhorar o tempo de execução.

Com o resultado da análise do plano de execução gerado, a análise dos dados de cada tabela e conforme o que foi estudado na seção 3, a sintonia de índices será aplicada tendo como base alguns critérios considerados relevantes:

- O uso de índices de uma coluna, em determinadas ocasiões, conforme a disposição dos dados, de múltiplas colunas, pode resumir o caminho percorrido e o número de dados carregados em memória. Assim como a criação de índices sobre as colunas usadas como filtro e chave de uniões em consultas.
- Em consultas em que se use os parâmetros *order e group* by, ao usar índices, a consulta acaba sendo encerrada quando não for mais possível encontrar determinado dado. Assim como a criação de índices parciais, que podem reduzir o número de dados que são armazenados no arquivo de índices, sendo a consulta sobre esse arquivo mais eficiente, desde que o conjunto de dados suporte esse tipo de índice.
- O uso de operadores combinados com a coluna usada para criar determinado índice deve ser evitado, pois o mesmo impossibilita o seu uso. Assim como o uso da expressão *LIKE* deve ser usada na forma especificada pela documentação do *PostgreSQL*, para que o índice devidamente criado seja utilizado.
- Utilizar determinada coluna ou expressão mais à esquerda ou à direita em determinadas consultas com os operadores *AND* e *OR*, pode acarretar em ganho ou perda de desempenho.

Esses critérios são apresentados de forma resumida no Quadro 2.

Quadro 2 - Critérios da sintonia de índice

Criação de índices
Únicos e de múltiplas colunas;
Sobre colunas que são usadas como filtro da cláusula *where*;
Sobre colunas que são chave de união da cláusula *join* em relacionamento de várias tabelas;
Sobre colunas em que ao executar uma consulta, se use os parâmetros *order by* e *group by*;
Parciais em determinadas operações que buscam dados de um determinado tipo;
Operações com colunas que usam índices
Evitar o uso de funções ou operadores combinados com a coluna de índice;
Uso do *LIKE* na forma em que o *PostgreSQL* faz uso do índice, se devidamente criado;
Determinar a expressão que é colocada a esquerda com o uso dos operadores *AND* e *OR*;

Fonte: Autor.

Depois da definição e aplicação da sintonia de índices, as operações que foram executadas antes da aplicação da sintonia serão novamente executadas. Serão avaliadas as informações retornadas pelo próprio *PostgreSQL*, sendo essas:

- O número final de linhas retornadas;
- O tempo de execução da operação.

O tempo pode variar em alguns milissegundos a cada execução, pois este depende de fatores como uso de memória cache ou uso dos componentes físicos pelo computador. Para minimizar esses fatores e evitar que possam mascarar a avaliação de desempenho, a cada consulta realizada o sistema operacional será reiniciado a fim de encerrar todos os serviços e limpar a memória cache. Adicionalmente, determinou-se relevante cada comando ser executado cinco vezes, onde será feita a média (soma dos cinco tempos, dividido por cinco) para cada um. Esses resultados serão utilizados como métrica para fazer a verificação de melhora ou não de desempenho, justificando cada resultado usando como base o que foi estudado nas seções 2 e 3.

A decisão de utilizar esse tipo de verificação baseada no tempo de execução de cada operação foi tomada levando em consideração o objetivo desse estudo e a possibilidade de verificar cada técnica de sintonia aplicada. O que também auxiliou nessa escolha foi o uso de métodos semelhantes em trabalhos importantes como em Magalhães et al. (2015) e Telles, Galante e Dorneles (2011), ambos apresentados na seção 2.2 deste estudo.

Considerou-se usar algum software de *Benchmark* para a realização do estudo de caso, porém na pesquisa realizada foi encontrado apenas o software *BenchmarkSQL* abordado na seção 4.1 como sendo compatível com o *PostgreSQL*. Também se detectou que a principal métrica utilizada para avaliação do desempenho com esse software é o número de transações por determinado período de tempo, e que os dados gerados são armazenados em um arquivo de texto que precisa ser interpretado e analisado.

Outro fator que foi levado em consideração para não utilizar o *BenchmarkSQL*, é que para executar os testes é necessário criar uma estrutura de índices pré-determinada, porém uma das propostas desse estudo é realizar operações de consulta sobre uma carga de dados sem a criação de estruturas de índices, e repeti-las após a criação delas conforme os critérios de sintonia de índices apresentados nessa seção, para ao final avaliar os resultados gerados.

Com base no que foi estudado, no objetivo desse estudo e nesses dois trabalhos, considera-se que o método escolhido é relevante.

7 ESTUDO DE CASO

Nesse capítulo será apresentada a implementação da proposta apresentada no capítulo 6, com a aplicação dos conceitos apresentados nos capítulos 2 e 3. Em cada seção será abordada uma determinada técnica de sintonia de índices, onde serão apresentados os resultados de determinada consulta sem e com a aplicação da mesma.

7.1 BASE DE DADOS

O estudo de caso será realizado sobre mais de uma base de dados. Para atender os critérios especificados no capítulo 6, a base de dados precisa ter algumas características como, várias colunas, tabelas, e diferentes tamanhos por tabela. A Tabela 1 mostra as principais características das bases de dados, cada uma terá uma identificação, que será usada no decorrer do trabalho para simplificar sua citação. Os comandos para criação das bases de dados, bem como as suas chaves primárias e secundárias, se encontram nos anexos A, B e C.

Tabela 1 - Características de cada base de dados

Identificação da base de dados	Identificação das tabelas	Número de linhas	Número de colunas	Tipo de dados
Base de dados livros – BD1	autor	1776355	2	character numeric
	edicao	1801125	3	character integer numeric
	livroautor	1877992	2	numeric
	livros	1747605	2	character numeric
Base de dados filiados – BD2	filiados_partido_y	1469908	13	bigint character date numeric
	filiados_partido_z	2139008	13	bigint character date numeric
	municipios_filiados	3648	3	character integer
	pessoas_filiados_y	1427309	2	bigint character
	pessoas_filiados_z	2080615	2	bigint character

Identificação da base de dados	Identificação das tabelas	Número de linhas	Número de colunas	Tipo de dados
Base de dados ENEM_RS – BD3	enem_rs_2015	418695	166	character numeric text

Fonte: Autor.

7.2 SINTONIA DE ÍNDICES COM O USO DO PARÂMETRO *GROUP BY*

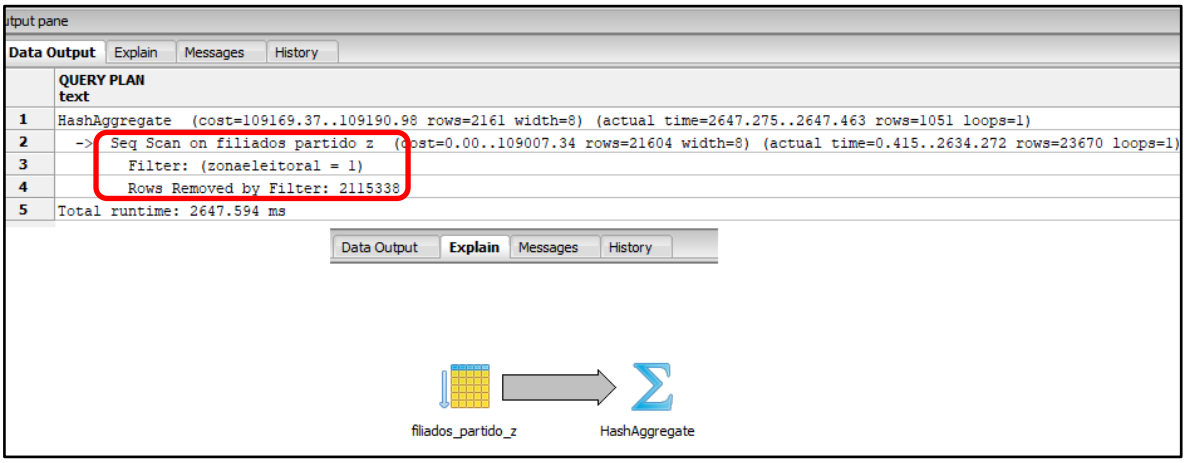
Sobre a BD2 é realizada determinada consulta usando como parâmetro o *group by*, com a finalidade de retornar o número de filiados que votam em determinado agrupamento zona – seção.

- EXPLAIN ANALYSE SELECT zonaeleitoral, secaoeleitoral, COUNT(*) AS numeroinscricao FROM filiados_partido_z WHERE zonaeleitoral = 1 GROUP BY zonaeleitoral, secaoeleitoral

7.2.1 Análise do plano de execução gerado

A consulta da seção anterior retornou o plano de execução apesentado na Figura 20, apresentado em seu modo textual e gráfico:

Figura 20 - Plano de execução da consulta usando *group by*



Fonte: *Print screen* da tela com o plano de execução apresentado no PgAdmin(Adaptado)

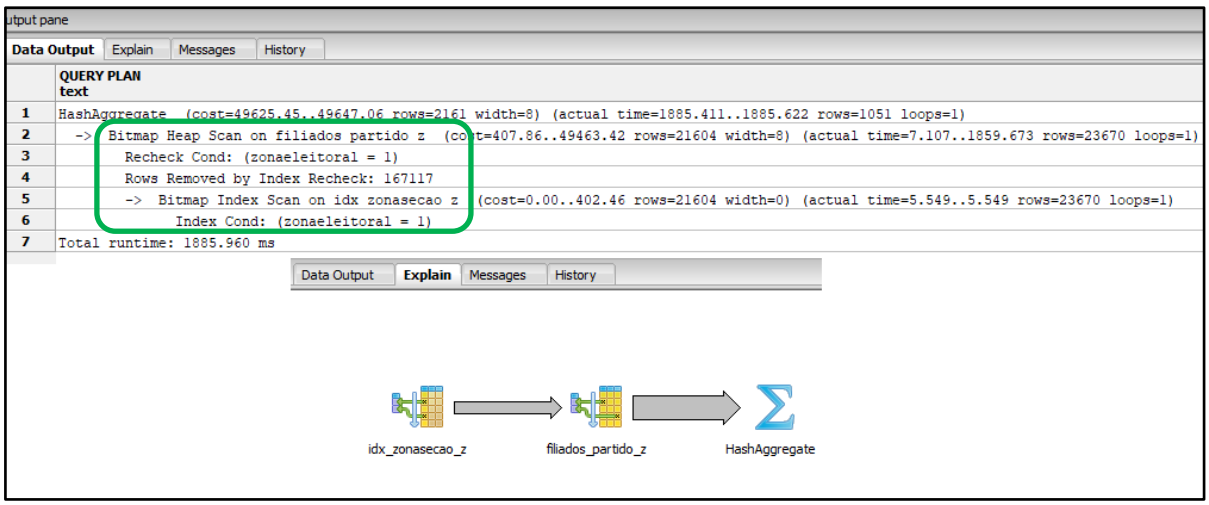
Pode-se notar que nessa consulta é realizada uma busca sequencial na tabela filiados_partido_z, pois não temos nenhum arquivo de índices criado para as colunas que são usadas como parâmetro do *group by* e o número de registros carregados e descartados através do filtro é considerável.

Pode-se aplicar a sintonia nessa consulta criando um arquivo de índices sobre as colunas zonaeleitoral e secaoeleitoral.

- CREATE INDEX idx_zonasecao_z ON filiados_partido_z (zonaeleitoral, secaoeleitoral)

Realizando novamente a mesma consulta pode-se verificar o novo plano de execução que foi gerado na Figura 21 apresentado em seu modo textual e gráfico:

Figura 21 - Plano de execução da consulta usando *group by* com sintonía de índices



Fonte: *Print screen* da tela com o plano de execução apresentado no *PgAdmin*(Adaptado)

Pode-se notar que após a aplicação da sintonia de índices, não temos mais a busca sequencial na tabela filiados_partido_z, temos agora uma busca *bitmap* no arquivo de índices criado. O número de linhas removidas pelo filtro diminui de forma expressiva, pois através do índice é possível detectar que o valor usado como filtro não pode ser mais encontrado, não sendo necessário varrer todo o arquivo.

7.2.2 Análise dos dados e conclusão

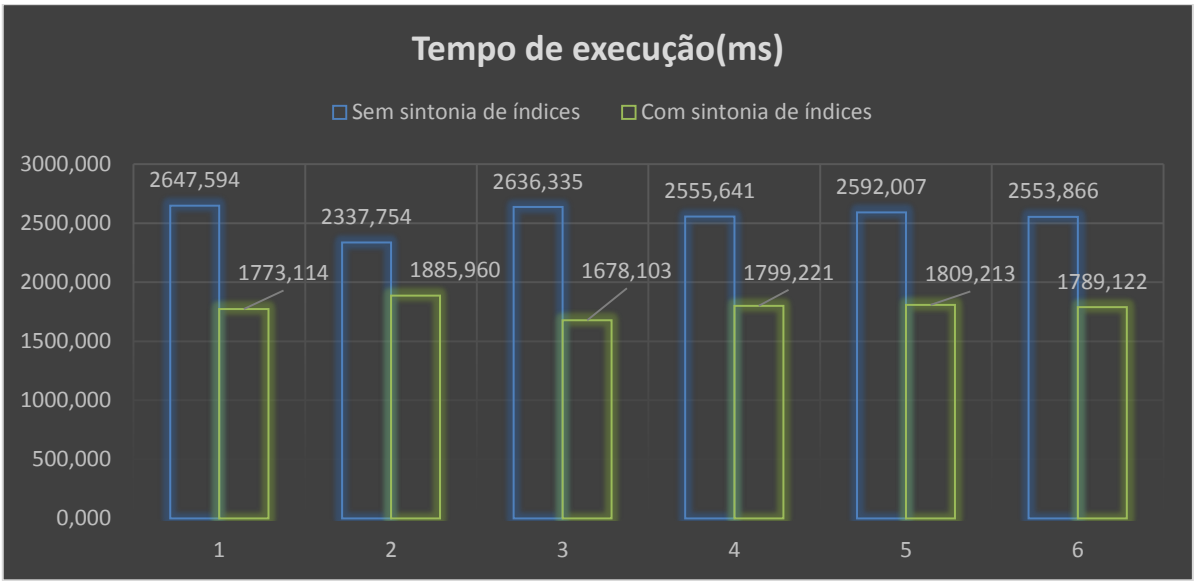
A Tabela 2 e o Gráfico 1 apresentam os resultados coletados ao realizar as consultas da seção 7.2 e 7.2.1.

Tabela 2 - Sintonia de índices com o uso do parâmetro *group by*

	Número de linhas retornadas	Tempo de execução (ms)	Desempenho da sintonia
Sem sintonia de índices	1051	2647,594	**Tempo de execução reduzido em: 30%**
		2337,754	
		2636,335	
		2555,641	
		2592,007	
Tempo médio de execução (ms)		**2553,866**	
Com sintonia de índices	1051	1773,114	
		1885,960	
		1678,103	
		1799,221	
		1809,213	
Tempo médio de execução (ms)		**1789,122**	

Fonte: Autor.

Gráfico 1 - Sintonia de índices com o uso do parâmetro *group by*



Fonte: Autor.

Levando em consideração os dados da Tabela 2 e do Gráfico1, pode-se analisar que a criação do arquivo de índices resultou na diminuição do tempo de execução da consulta em todas as execuções. Considerando a média de todas elas, o tempo de execução reduziu em torno de 30%.

Adicionalmente foram realizadas consultas que gerassem o relatório de número de filiados por agrupamento de zona – seção da tabela inteira, pode-se verificar que nesse caso é necessário retornar o número total de linhas do arquivo de índices. Para tal tipo de consulta não é viável a criação de um arquivo de índices, pois após percorrer todo o arquivo, é necessário realizar nova checagem na tabela para depois realizar o agrupamento, sendo que a busca sequencial é realizada diretamente na tabela e após já realiza o agrupamento.

Foi-se reduzindo o número de linhas retornadas através de restrição do número de zonas em que se buscava o número de seções, onde foi verificado que a partir do momento em que a consulta retorna mais de 37% dos registros do arquivo de índices, o mesmo não é mais utilizado.

7.3 SINTONIA DE ÍNDICES COM O USO DO OPERADOR *LIKE*

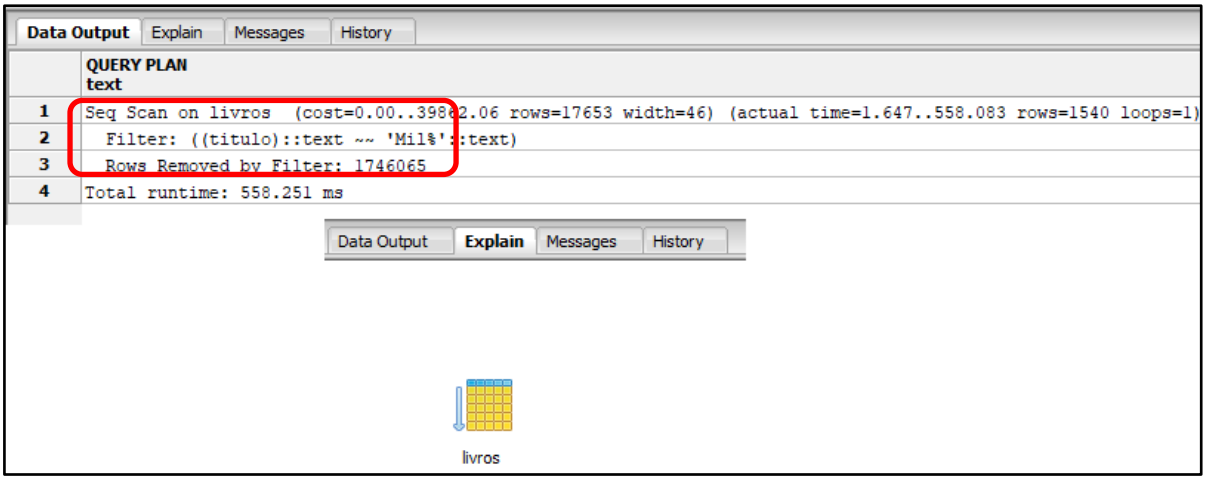
Sobre a BD1 é realizada determinada consulta usando o operador *LIKE* para retornar os títulos dos livros que contenham determinada sequência de caracteres.

- EXPLAIN ANALYSE SELECT titulo FROM livros WHERE titulo LIKE 'Mil%'

7.3.1 Análise do plano de execução gerado

A consulta da seção anterior retornou o plano de execução apesentado na Figura 22 apresentado em seu modo textual e gráfico:

Figura 22 - Plano de execução da consulta com o uso do operador *like*



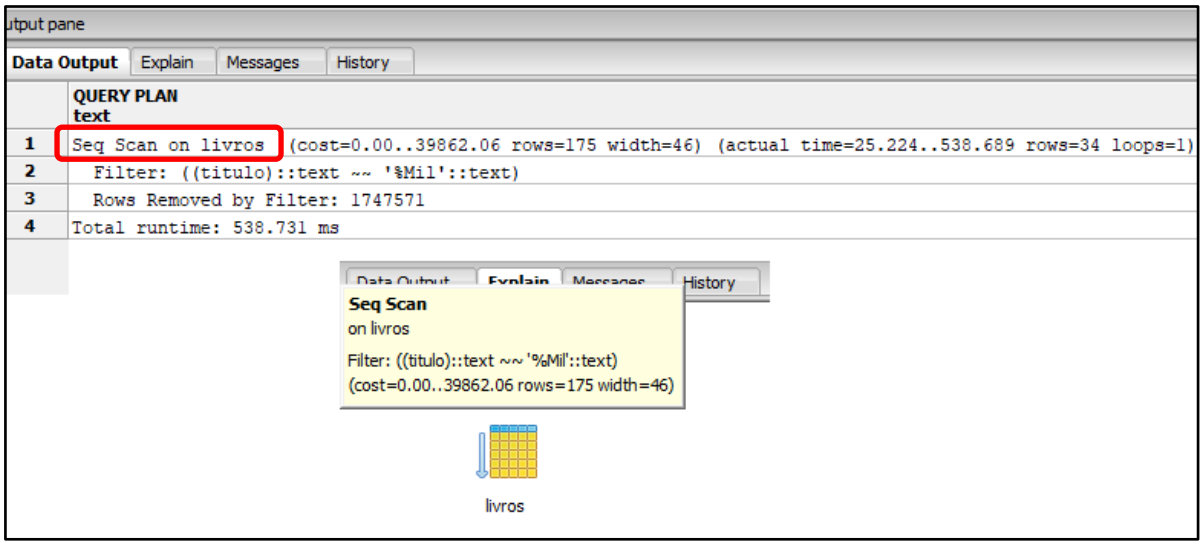
Fonte: *Print screen* da tela com o plano de execução apresentado no *PgAdmin*(Adaptado)

Pode-se verificar que a busca acabou sendo executada de maneira sequencial na tabela livros, por não ter nenhum arquivo de índices criado, realizando assim a varredura na tabela em sua totalidade. Pode-se aplicar a sintonia de índices criando um índice sobre a coluna titulo.

- CREATE INDEX idx_titulo ON livros (titulo)

Caso a consulta com o uso do operador LIKE não seja realizada conforme especificado na documentação (EXPLAIN ANALYSE SELECT titulo FROM livros WHERE titulo LIKE '%Mil'), o arquivo de índices não será usado, conforme pode ser verificado no plano de execução apesentado na Figura 23 apresentado em seu modo textual e gráfico:

Figura 23 - Plano de execução da consulta com o uso o operador *like* com sintonia de índices

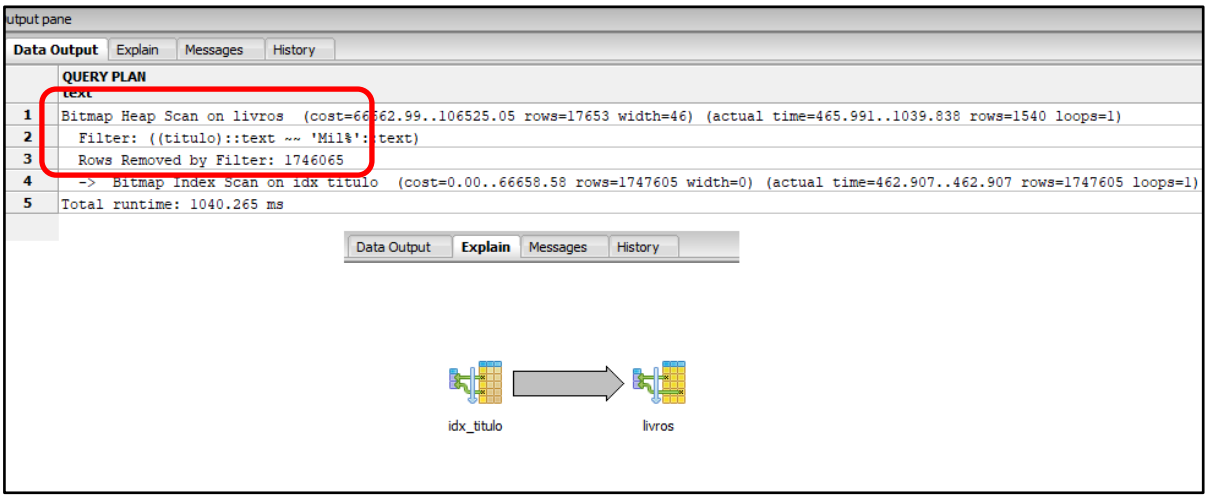


Fonte: *Print screen* da tela com o plano de execução apresentado no *PgAdmin*(Adaptado)

Pode-se verificar que a busca acabou sendo executada de maneira sequencial na tabela livros, não usando a estrutura de índices já criada.

Realizando a consulta conforme especificação da documentação do *PostgreSQL* e descrita na seção 7.3 para que ela faça o uso do arquivo de índices criado, pode-se analisar o novo plano de execução que foi gerado na Figura 24 apresentado em seu modo textual e gráfico:

Figura 24 - Plano de execução da consulta com o uso o operador *like* com sintonia de índices no formato especificado



Fonte: *Print screen* da tela com o plano de execução apresentado no *PgAdmin*(Adaptado)

Pode-se verificar que após o uso do operador *LIKE* conforme a documentação, a busca passou a ser realizada usando a estrutura de índices que havia sido criada, fazendo assim uma busca *bitmap* nessa estrutura, porém a busca tem um desempenho inferior a busca sequencial, pois o número de linhas descartadas pelo filtro ainda é o mesmo, adicionalmente sendo necessário buscar os registros no arquivo de índices e após na tabela original.

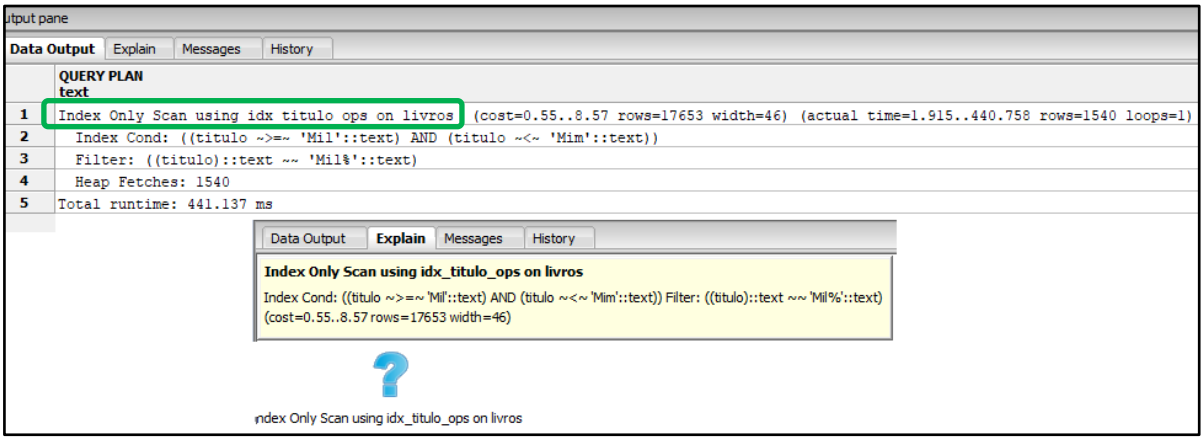
Para que o SGBD usasse o arquivo de índices criado, foi necessário desabilitar a busca sequencial através do comando SET ENABLE_SEQSCAN = OFF, pois o planejador/otimizador possui recursos suficientes para verificar qual é o plano mais eficiente.

A seção 3.8 aborda o uso de classes de operadores, sendo que o seu uso pode melhorar o desempenho de consultas que fazem o uso do operador *LIKE*. Sendo a coluna usada nesta consulta como filtro, do tipo *varchar*, pode ser aplicada a classe de operadores *varchar_pattern_ops*. Dessa forma a sintonia de índices com classes de operadores pode ser aplicada nessa consulta criando um índice desse tipo sobre a coluna titulo.

- CREATE INDEX idx_titulo_ops ON livros (titulo VARCHAR_PATTERN_OPS)

Realizando a consulta novamente, pode-se verificar o novo plano de execução que foi gerado na Figura 25 apresentado em seu modo textual e gráfico:

Figura 25 - Plano de execução da consulta usando classe de operadores na sintonia de índices



Fonte: Print screen da tela com o plano de execução apresentado no PgAdmin(Adaptado)

Pode-se analisar que a estrutura de índices usada foi a que usa a classe de operadores *varchar_pattern_ops*, sendo que a busca *bitmap* no arquivo de índices e na tabela livros, usada anteriormente foi substituída por uma busca indexada diretamente no arquivo de índices criado com a classe de operadores especificada.

7.3.2 Análise dos dados e conclusão

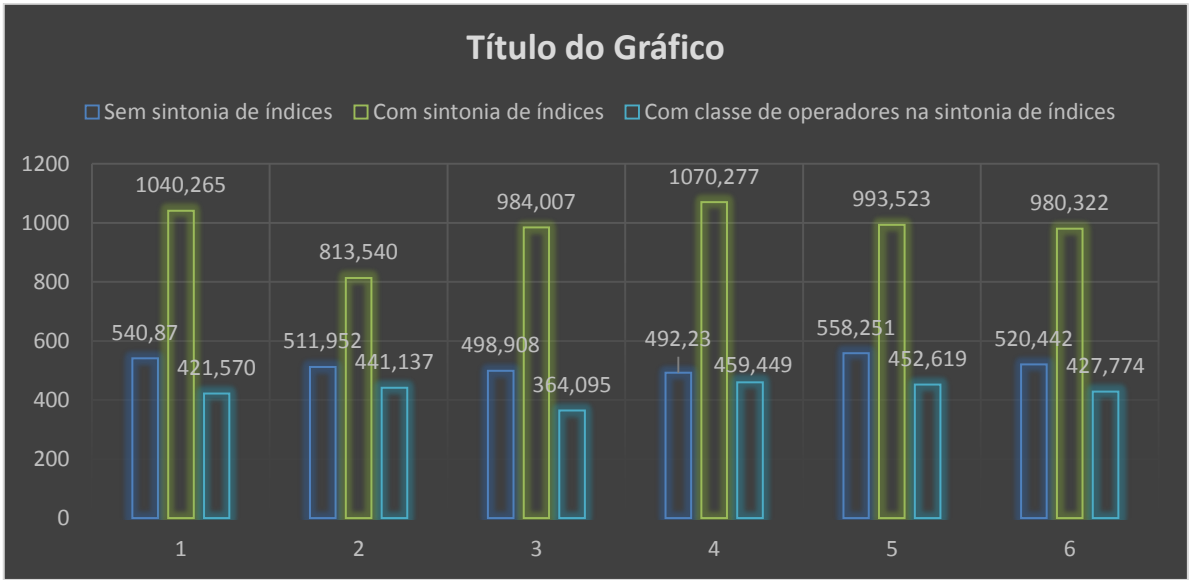
A Tabela 3 e o Gráfico 2 apresentam os resultados coletados ao realizar as consultas da seção 7.3 e 7.3.1.

Tabela 3 - Sintonia de índices com o uso do operador *like*

	Número de linhas retornadas	Tempo de execução (ms)	Desempenho da sintonia
Sem sintonia de índices	1540	540,870	**Tempo de execução reduzido em: 18%**
		511,952	
		498,908	
		492,230	
		558,251	
Tempo médio de execução (ms)		**520,442**	
Com sintonia de índices	1540	1040,265	
		813,540	
		984,007	
		1070,277	
		993,523	
Tempo médio de execução (ms)		**980,322**	
Com classe de operadores na sintonia de índices	1540	421,570	
		441,137	
		364,095	
		459,449	
		452,619	
Tempo médio de execução (ms)		**427,774**	

Fonte: Autor.

Gráfico 2 - Sintonia de índices com o uso do operador *like*



Fonte: Autor.

Levando em consideração os dados da Tabela 3 e do Gráfico 2, pode-se analisar que a aplicação da sintonia de índices sem a classe de operadores, utilizando o *LIKE* resultou num aumento expressivo no tempo de execução, mostrando não ser uma técnica de sintonia de índices eficiente.

Já o uso de índices onde se especifica a classe de operadores resultou em uma busca mais eficiente, conforme estudado na seção 3.8. Levando em consideração a média dos tempos de execução das consultas, a redução obtida foi de 18% em relação as consultas realizadas sem a aplicação da sintonia de índices.

7.4 SINTONIA DE ÍNDICES SOBRE COLUNAS QUE SÃO CHAVE DE UNIÃO DA CLÁUSULA *JOIN* E FILTRO DA CLÁUSULA *WHERE*

Sobre a BD2 é realizada determinada consulta a cláusula *JOIN* para realizar a união de duas tabelas para retornar o número de inscrição de todos os filiados a determinado partido e município.

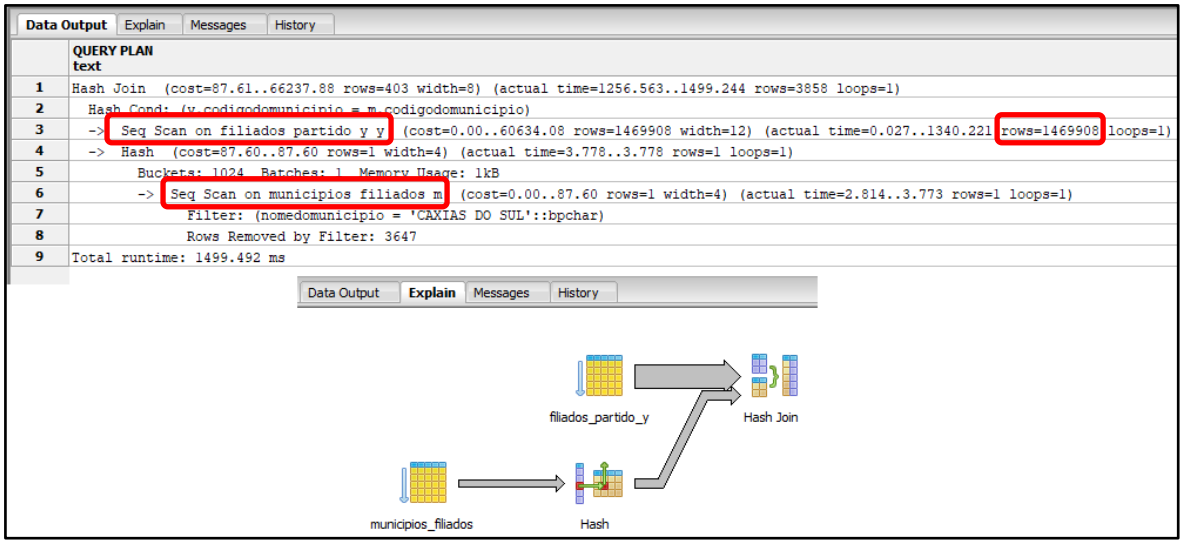
- EXPLAIN ANALYSE SELECT y.numeroinscricao FROM filiados_partido_y AS y INNER JOIN municipios_filiados AS m ON

```
y.codigodomunicipio=m.codigodomunicipio WHERE m.nomedomunicipio =
'CAXIAS DO SUL'
```

7.4.1 Análise do plano de execução gerado

A consulta da seção anterior retornou o plano de execução apesentado na Figura 26 apresentado em seu modo textual e gráfico:

Figura 26 - Plano de execução da consulta usando *join* e *where*



Fonte: *Print screen* da tela com o plano de execução apresentado no *PgAdmin*(Adaptado)

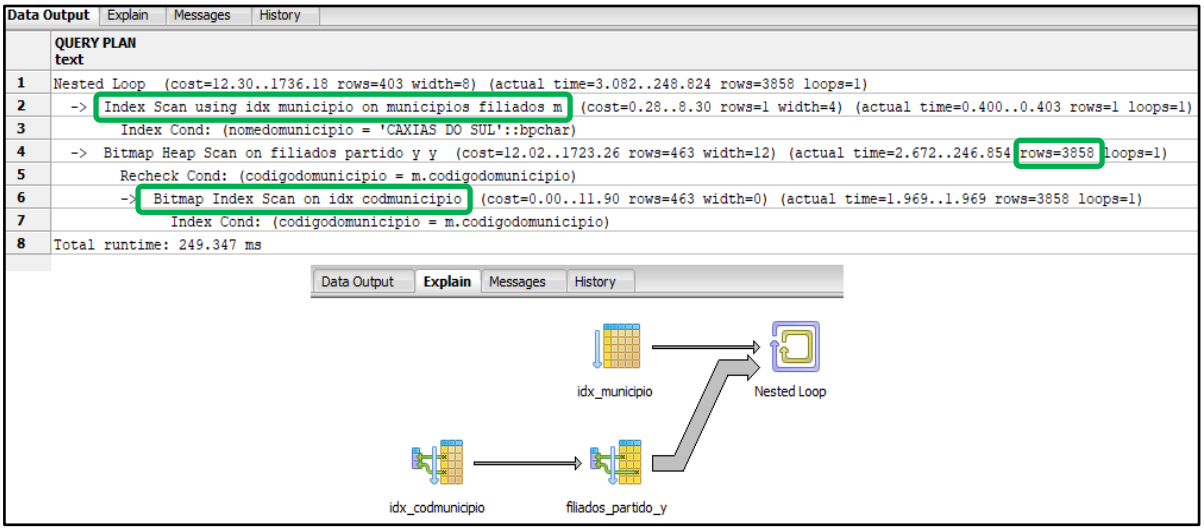
Pode-se notar que nessa consulta ocorrem duas buscas sequenciais, tanto na coluna que é usada como filtro da cláusula *where*, quanto na coluna usada como chave da união das duas tabelas, pois não temos nenhum arquivo de índices criado para essas colunas.

A sintonia de índices pode ser aplicada nessa consulta, criando um arquivo de índices sobre a coluna nomedomunicipio da tabela municípios_filiados e outro sobre a coluna codigodomunicipio da tabela filiados_partido_y.

- CREATE INDEX idx_municipio ON municipios_filiados (nomedomunicipio)
- CREATE INDEX idx_codmunicipio ON filiados_partido_y (codigodomunicipio)

Realizando novamente a mesma consulta pode-se verificar o novo plano de execução que foi gerado na Figura 27 apresentado em seu modo textual e gráfico:

Figura 27 - Plano de execução da consulta usando *join* e *where* com sintonia de índices



Fonte: *Print screen* da tela com o plano de execução apresentado no *PgAdmin*(Adaptado)

Pode-se notar que os caminhos planejados para a nova consulta são alterados. Além da eliminação das buscas sequenciais nas tabelas envolvidas, também foi alterada a forma de realizar a junção das operações, agora sendo realizada através do *nested loop*. Com o uso dos índices, temos buscas que varrem um número menor de linhas, pois a busca agora é indexada, e a busca na tabela filiados_partidos_y é feita através de uma varredura *bitmap*.

7.4.2 Análise dos dados e conclusão

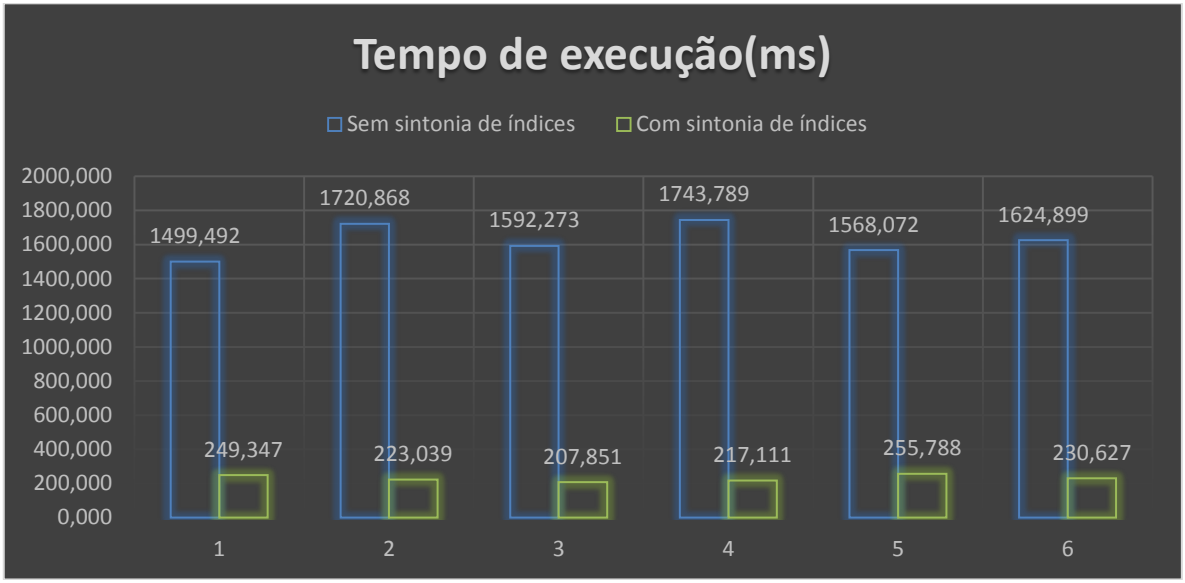
A Tabela 4 e o Gráfico 3 apresentam os resultados coletados ao realizar as consultas da seção 7.4 e 7.4.1.

Tabela 4 - Sintonia de índices sobre colunas chave de união e filtro da cláusula *where*

	Número de linhas retornadas	Tempo de execução (ms)	Desempenho da sintonia
Sem sintonia de índices	3858	1499,492	**Tempo de execução reduzido em: 85%**
		1720,868	
		1592,273	
		1743,789	
		1568,072	
Tempo médio de execução (ms)		**1624,898**	
Com sintonia de índices	3858	249,347	
		223,039	
		207,851	
		217,111	
		255,788	
Tempo médio de execução (ms)		**230,627**	

Fonte: Autor.

Gráfico 3 - Sintonia de índices sobre colunas chave de união e filtro da cláusula *where*



Fonte: Autor.

Levando em consideração os dados da Tabela 4 e do Gráfico 3, pode-se analisar que a criação das duas estruturas auxiliares reduziu significativamente o tempo de execução da consulta. Levando em conta a média desses tempos, a redução foi em torno de 85%.

O principal fator que gerou esse ganho em tempo de execução, se deve ao fato de que, as buscas podem ser encerradas assim que for detectado que o valor usado como filtro não pode mais ser encontrado, conforme estudado na seção 3.7.

7.5 SINTONIA DE ÍNDICES DE MULTIPLAS COLUNAS

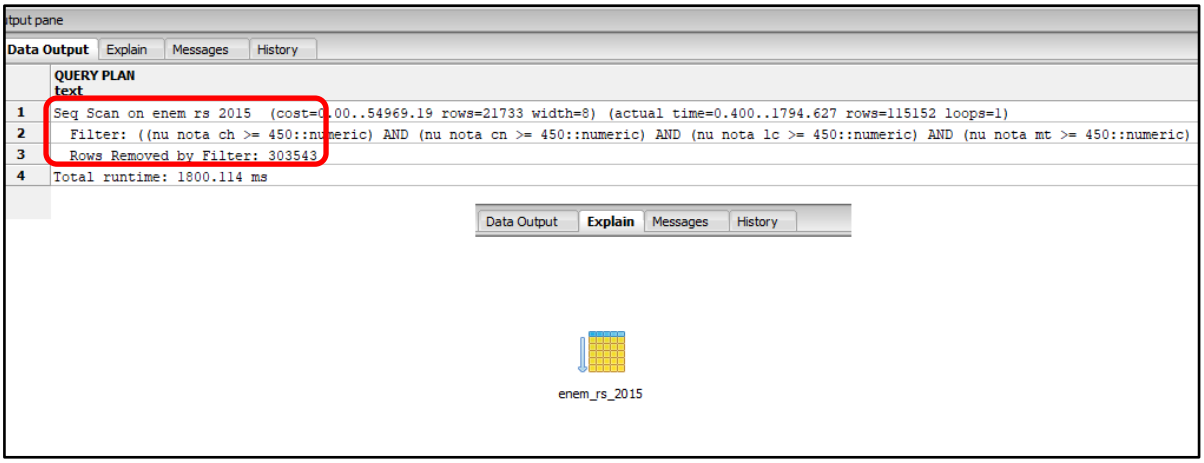
Sobre a BD3 é realizada uma consulta que gera um relatório de todos os alunos que tiveram uma nota igual ou maior que 450 pontos em todas as áreas de conhecimento e uma nota maior que 0 na redação.

- EXPLAIN ANALYSE SELECT nu_inscricao FROM enem_rs_2015 WHERE nu_nota_mt >= 450 AND nu_nota_cn >= 450 AND nu_nota_ch >= 450 AND nu_nota_lc >= 450 AND nu_nota_redacao > 0

7.5.1 Análise do plano de execução gerado

A consulta da seção anterior retornou o plano de execução apesentado na Figura 28 apresentado em seu modo textual e gráfico:

Figura 28 - Plano de execução da consulta com múltiplas colunas



Fonte: *Print screen* da tela com o plano de execução apresentado no *PgAdmin*(Adaptado)

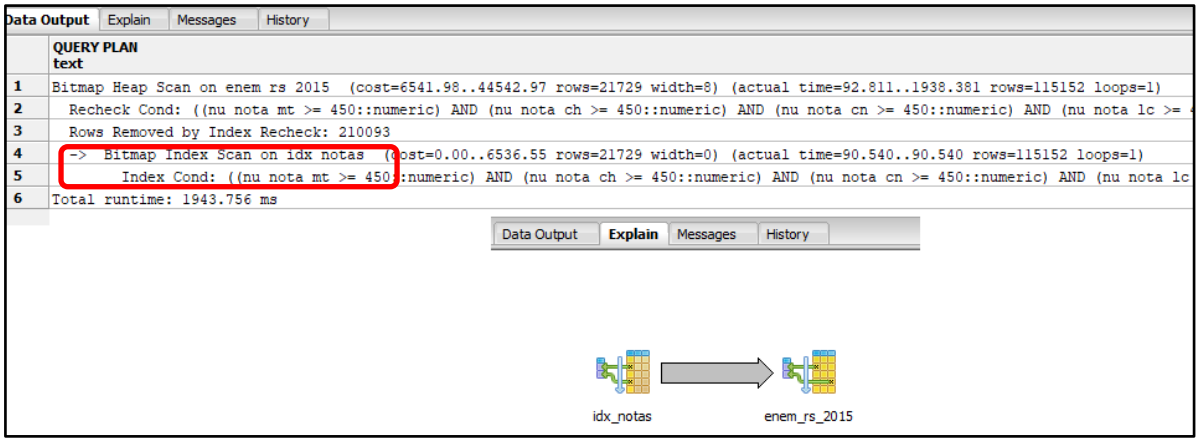
Pode-se notar que nessa consulta é realizada uma busca sequencial no arquivo, pois não temos nenhuma ordenação nos campos usados como filtros por não ter nenhum arquivo de índices ou não serem campos chave na tabela, e um número elevado de linhas são carregadas na memória e descartadas pelo filtro.

A sintonia de índices nessa consulta é aplicável criando um arquivo de índices de múltiplas colunas sobre as colunas filtro da cláusula *where*.

- CREATE INDEX idx_notas ON enem_rs_2015 (nu_nota_mt, nu_nota_cn, nu_nota_ch, nu_nota_lc, nu_nota_redacao)

Executando a consulta novamente, temos o seguinte plano de execução demonstrado na Figura 29 apresentado em seu modo textual e gráfico:

Figura 29 - Plano de execução da consulta com múltiplas colunas com sintonia de índices



Fonte: *Print screen* da tela com o plano de execução apresentado no *PgAdmin*(Adaptado)

Analisando o novo plano de execução gerado, pode-se verificar que a busca sequencial foi substituída por uma busca *bitmap* indexada, porém o número de registros descartados pelo filtro ainda continua sendo um número relevante. Analisando os dois planos de execução, verificamos que a busca sem sintonia de índices é planejada e executada em apenas um passo, e a busca com a sintonia de índices em dois passos.

7.5.2 Análise dos dados e conclusão

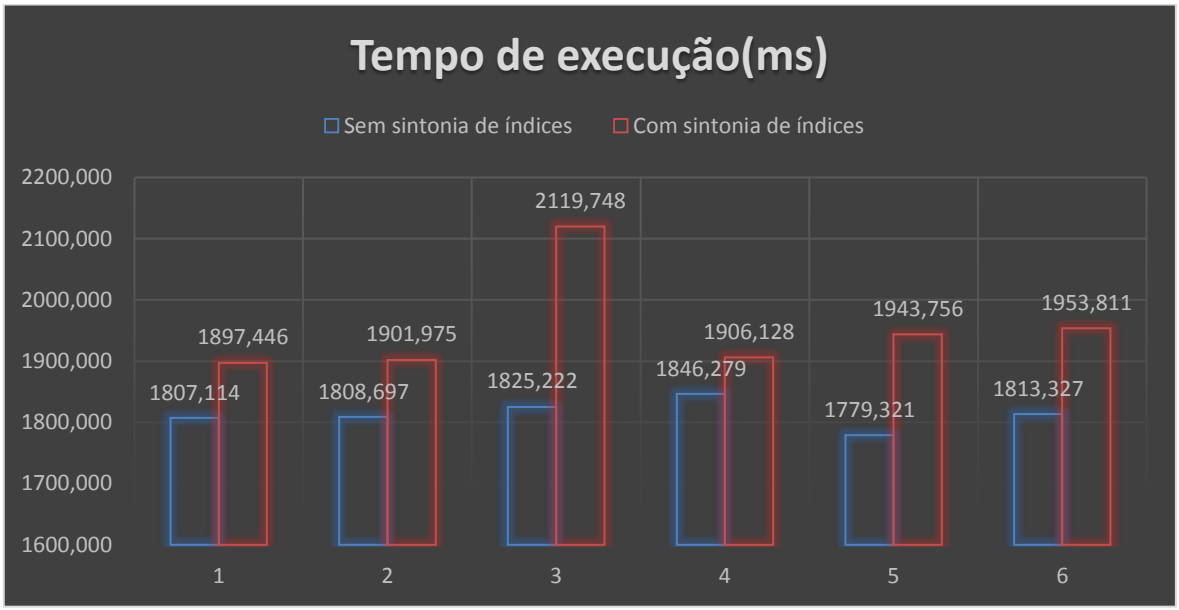
A Tabela 5 e o Gráfico 4 apresentam os resultados coletados ao realizar as consultas da seção 7.5 e 7.5.1.

Tabela 5 - Sintonia de índices de múltiplas colunas

	Número de linhas retornadas	Tempo de execução (ms)	Desempenho da sintonia
Sem sintonia de índices	115152	1807,114	**Tempo de execução aumentou em: 7%**
		1808,697	
		1825,222	
		1846,279	
		1779,321	
Tempo médio de execução (ms)		**1813,326**	
Com sintonia de índices	115152	1897,446	
		1901,975	
		2119,748	
		1906,128	
		1943,756	
Tempo médio de execução (ms)		**1953,810**	

Fonte: Autor.

Gráfico 4 - Sintonia de índices de múltiplas colunas



Fonte: Autor.

Levando em consideração os dados da Tabela 5 e do Gráfico 4, pode-se analisar que a criação do índice de múltiplas colunas nesse caso ocasionou o aumento do tempo de execução em torno de 7%.

Esse aumento ocorreu devido ao elevado número de linhas que essa consulta retorna (em torno de 28%), ademais o número total de registros dessa tabela é relativamente pequeno, sendo a BD3 a base com menor número de registros. Isso faz com que o carregamento completo dos dados da tabela seja realizado de modo mais eficiente, do que o carregamento do arquivo de índices e após a busca dos dados na tabela, mesmo a busca indexada varrendo menos linhas, conforme estudado na seção 3.7.

Para que o SGBD usasse o arquivo de índices criado, foi necessário desabilitar a busca sequencial através do comando SET ENABLE_SEQSCAN = OFF, pois o planejador/otimizador possui recursos suficientes para verificar qual é o plano mais eficiente, e por isso descarta o plano gerado com o uso de índices pelo fato do tempo de execução dessa consulta ser maior do que a busca sequencial. Sendo que neste teste a sintonia de índices se mostrou ineficiente, não melhorando o desempenho dessa consulta.

7.6 SINTONIA DE ÍNDICES PARCIAIS

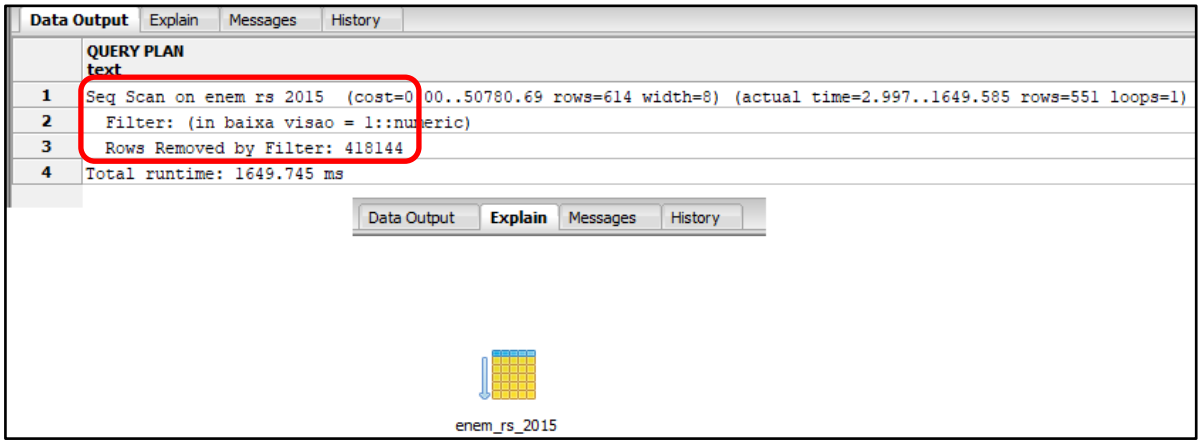
Sobre a BD3 é realizada uma consulta que gera um relatório de todos os alunos que possuem baixa visão.

- EXPLAIN ANALYSE SELECT nu_inscricao FROM enem_rs_2015 WHERE in_baixa_visao = 1

7.6.1 Análise do plano de execução gerado

A consulta da seção anterior retornou o plano de execução apesentado na Figura 30 apresentado em seu modo textual e gráfico:

Figura 30 - Plano de execução da consulta sobre a coluna in_baixa_visao



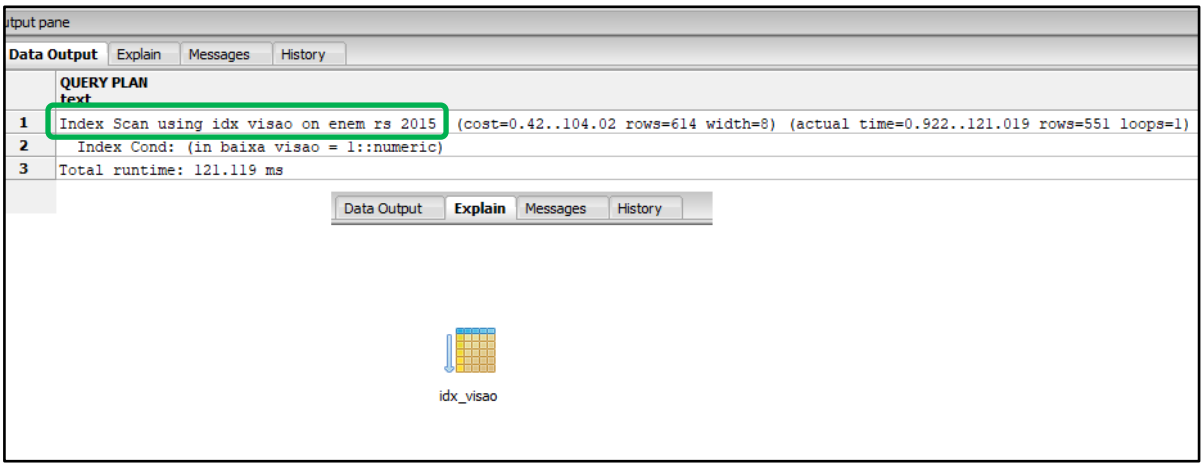
Fonte: *Print screen* da tela com o plano de execução apresentado no *PgAdmin*(Adaptado)

Pode-se notar que essa consulta é realizada através de uma busca sequencial na tabela inteira, e o número de registros carregados e removidos pelo filtro é elevado por não termos um arquivo de índices criado sobre a coluna usada como filtro. A sintonia pode ser aplicada criando um arquivo de índices sobre a coluna usada como filtro da cláusula *where*, assim como foi realizado na seção 7.4.

- CREATE INDEX idx_visao ON enem_2015_rs (in_baixa_visao)

Após a aplicação da sintonia de índices, e aplicando novamente a mesma consulta, temos o seguinte plano de execução demonstrado na Figura 31 apresentado em seu modo textual e gráfico:

Figura 31 - Plano de execução da consulta com sintonia de índices



Fonte: *Print screen* da tela com o plano de execução apresentado no *PgAdmin*(Adaptado)

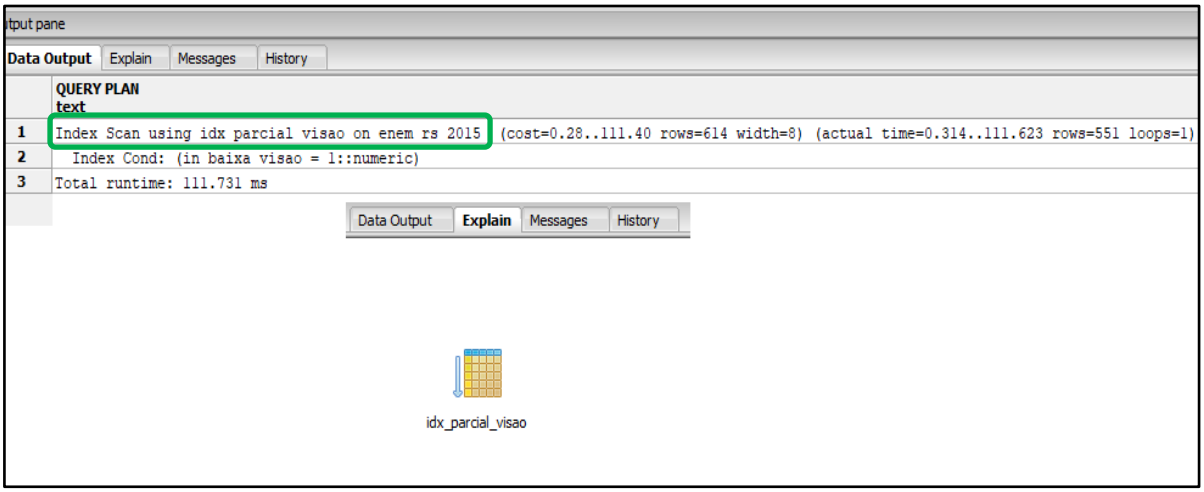
Pode-se notar que a busca sequencial deixou de ser realizada, sendo planejada e executada uma busca indexada, que é realizada de modo mais eficiente, pois o tamanho do arquivo de índices é menor que a tabela de dados completa e a busca consegue ser encerrada assim que o valor usado como filtro não pode mais ser encontrado, conforme estudado no capítulo 3.

Porém, na seção 3.5 foi realizada uma abordagem aos índices parciais, explicando seu funcionamento e seu uso. Ao analisarmos os dados contidos na coluna in_baixa_visao e a consulta da seção 6.6 pode-se notar que esse pode ser o caso de criarmos um índice parcial, pois esse tipo de consulta descarta os dados que contenham o valor 0 nessa coluna, e que são a maior parte dos dados dessa coluna.

- CREATE INDEX idx_parcial_visao ON enem_2015_rs (in_baixa_visao) WHERE in_baixa_visao = 1

Aplicando novamente a mesma consulta, temos o seguinte plano de execução demonstrado na Figura 32 apresentado em seu modo textual e gráfico:

Figura 32 - Plano de execução da consulta com sintonia de índices parciais



Fonte: *Print screen* da tela com o plano de execução apresentado no *PgAdmin*(Adaptado)

Analisando o novo plano de execução gerado, pode-se verificar que continua sendo realizada uma busca indexada, porém agora sobre o arquivo de índices parcial que foi criado.

7.6.2 Análise dos dados e conclusão

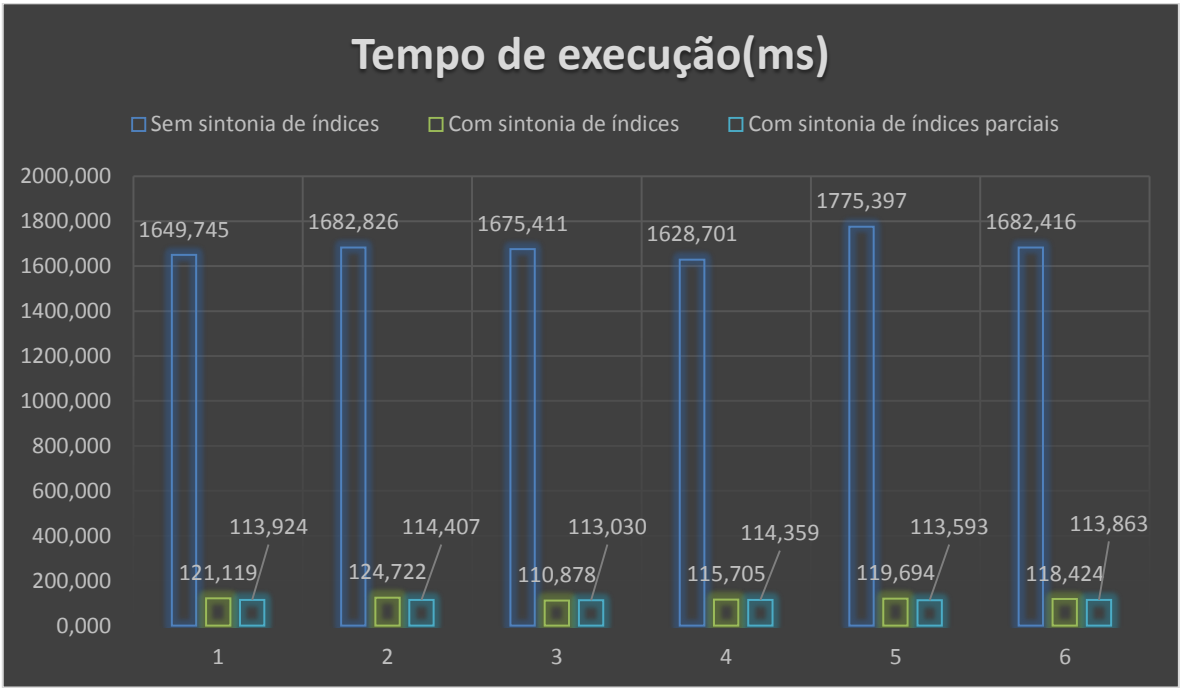
A Tabela 6 e o Gráfico 5 apresentam os resultados coletados ao realizar as consultas da seção 7.6 e 7.6.1.

Tabela 6 - Sintonia de índices parciais

	Número de linhas retornadas	Tempo de execução (ms)	Desempenho da sintonia
Sem sintonia de índices	551	1649,745	**Tempo de execução reduzido em: 93%**
		1682,826	
		1675,411	
		1628,701	
		1775,397	
Tempo médio de execução (ms)		**1682,416**	
Com sintonia de índices	551	121,119	
		124,722	
		110,878	
		115,705	
		119,694	
Tempo médio de execução (ms)		**118,423**	
Com sintonia de índices parciais	551	113,924	
		114,407	
		113,030	
		114,359	
		113,593	
Tempo médio de execução (ms)		**113,862**	

Fonte: Autor.

Gráfico 5 - Sintonia de índices parciais



Fonte: Autor.

Levando em consideração os dados da Tabela 6 e do Gráfico 5, pode-se analisar que após a aplicação da sintonia de índices sobre a coluna usada como filtro da cláusula *where* obtivemos um ganho de cerca de 92% em tempo de execução. A aplicação da sintonia de índices parciais gerou mais um ganho de aproximadamente 1%, ocorrendo um ganho total de 93% em comparação com o tempo de execução da consulta sem sintonia.

O ganho após a aplicação do índice parcial não é um número muito expressivo, isso se deve ao fato que essa base de dados não ter um número muito elevado de registros. Porém pode ser levado em conta que a aplicação desse tipo de sintonia gera um arquivo de índices de menor tamanho em comparação ao arquivo de índices sobre a coluna completa, conforme abordado na seção 3.5.

7.7 SINTONIA DE ÍNDICES EM EXPRESSÕES

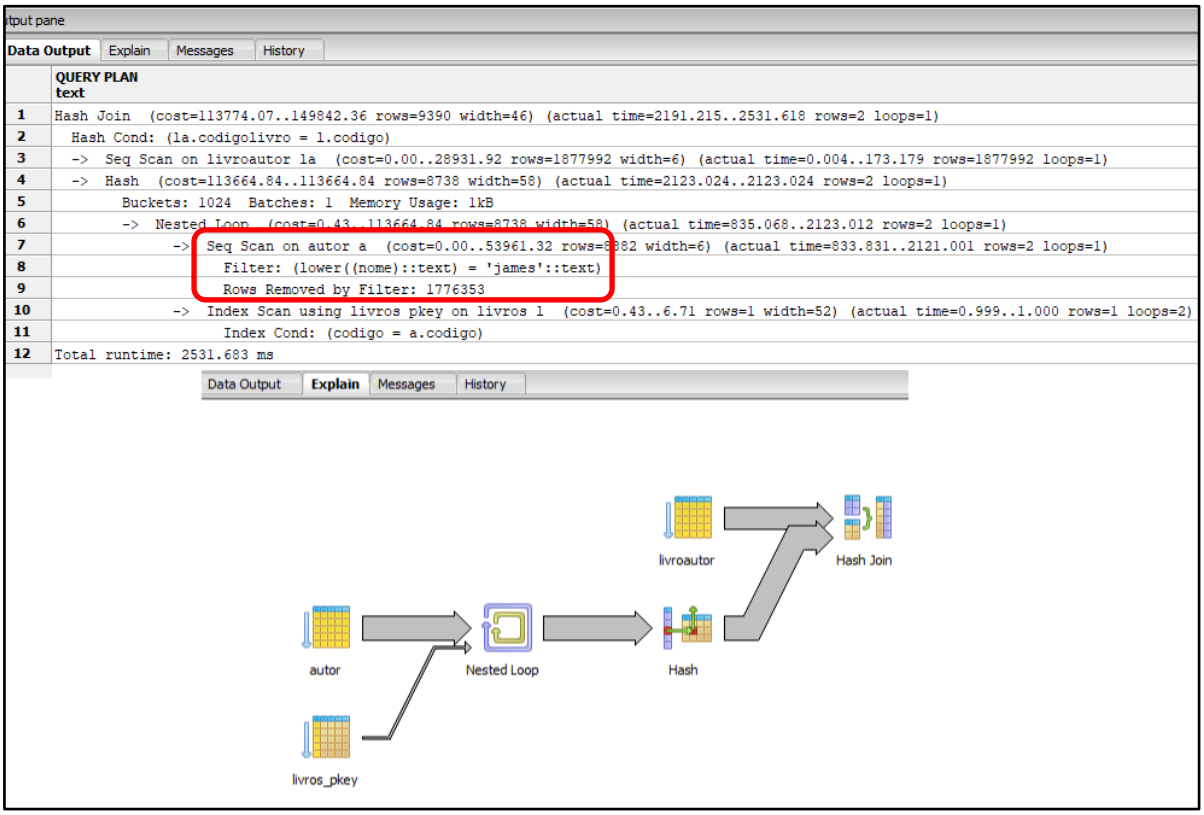
Sobre a BD1 é realizada uma consulta para retornar todos os títulos dos livros que sejam de um determinado autor, sendo que a busca pelo autor é realizada através da coluna nome do autor com o uso da função *lower* para não diferenciar letras maiúsculas e minúsculas.

- EXPLAIN ANALYSE SELECT l.titulo FROM livros AS l LEFT JOIN livroautor AS la ON l.codigo = la.codigolivro LEFT JOIN autor AS a ON a.codigo = la.codigolivro WHERE LOWER (a.nome) = 'james'

7.7.1 Análise do plano de execução gerado

A consulta da seção anterior retornou o plano de execução apesentado na Figura 33 apresentado em seu modo textual e gráfico:

Figura 33 - Plano de execução da consulta com função



Fonte: *Print screen* da tela com o plano de execução apresentado no *PgAdmin*(Adaptado)

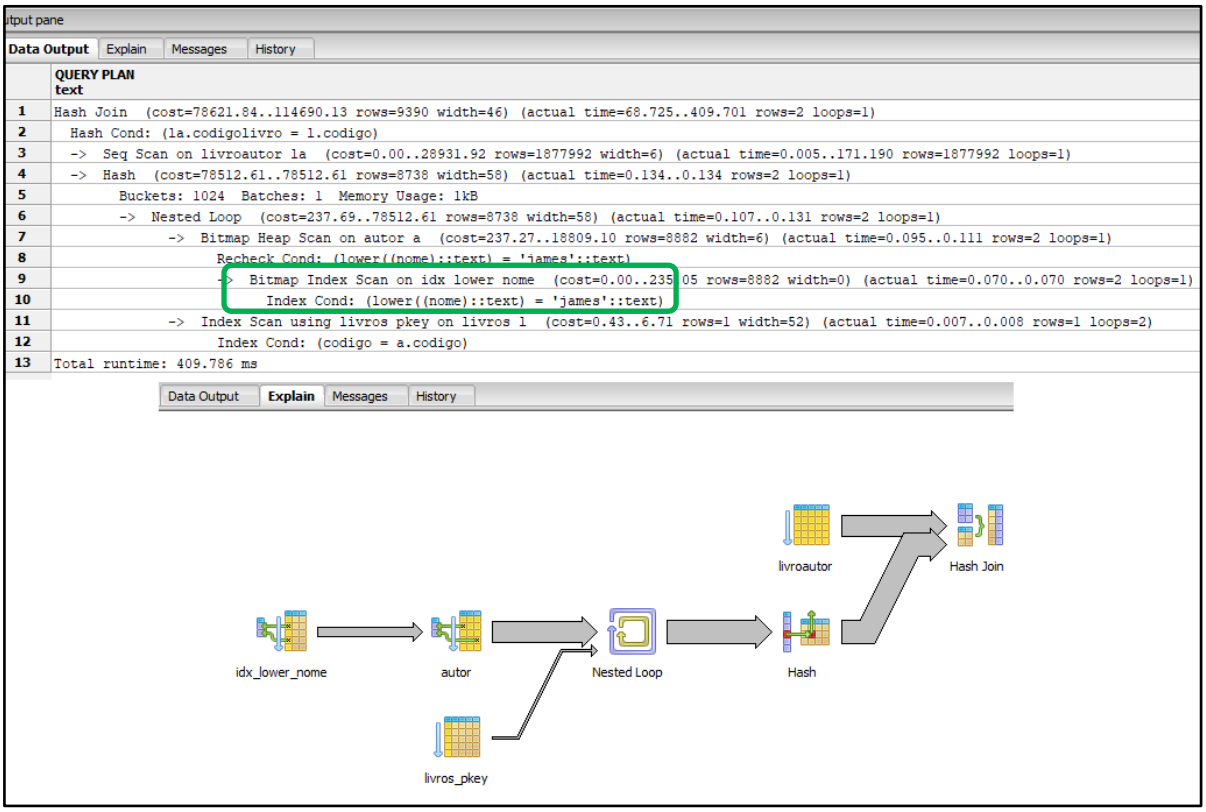
Pode-se notar no plano de execução gerado, que ao realizar a busca executando a função *lower* ocorre uma busca sequencial na tabela autor e um número elevado de registros são descartados pelo filtro, pois não temos nenhuma estrutura de índices criada. Além da busca sequencial, a cada execução haverá o custo para executar a função associada a consulta.

Para que ocorra o uso de uma estrutura de índices com esse tipo de consulta, pode-se aplicar a sintonia de índices em expressões, em que o índice é criado sobre o resultado da função.

- CREATE INDEX idx_lower_nome ON autor (LOWER(nome))

Aplicando novamente a mesma consulta, temos o seguinte plano de execução demonstrado na Figura 34 apresentado em seu modo textual e gráfico:

Figura 34 - Plano de execução da consulta com sintonia de índices em expressões



Fonte: *Print screen* da tela com o plano de execução apresentado no *PgAdmin*(Adaptado)

Pode-se verificar que a estrutura de índices criada foi usada, realizando uma busca *bitmap* no arquivo de índices criado sobre o resultado da função *lower*, isso faz com que a estrutura do plano de execução montado seja alterada na sua primeira parte. Onde havia uma busca sequencial na tabela autor, temos agora a busca no arquivo de índices, o que faz com que menos dados sejam carregados para a memória.

7.7.2 Análise dos dados e conclusão

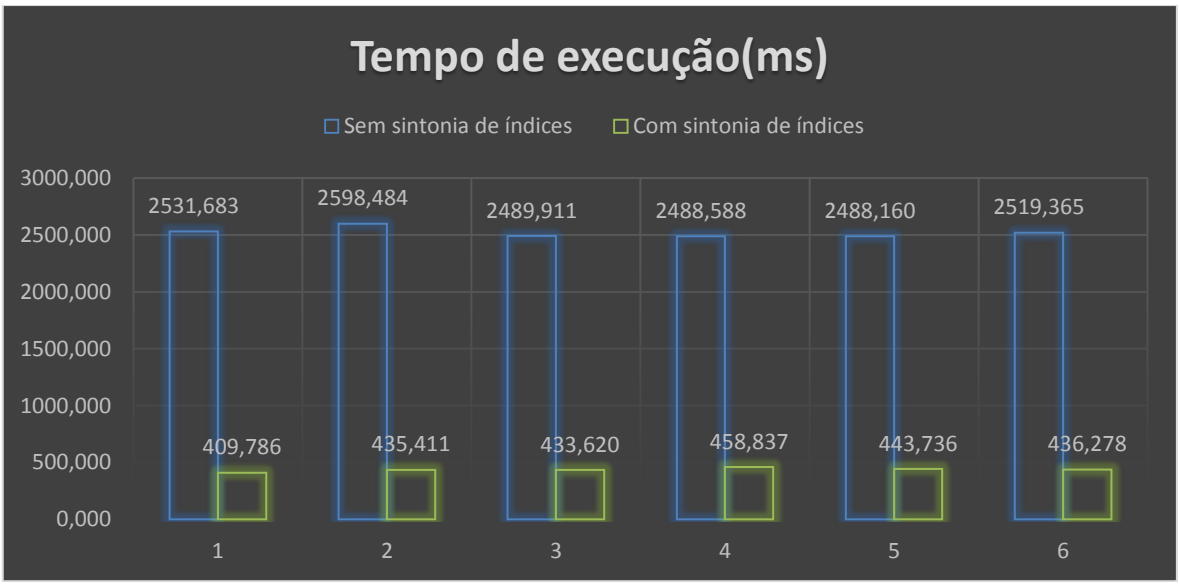
A Tabela 7 e o Gráfico 6 apresentam os resultados coletados ao realizar as consultas da seção 7.7 e 7.7.1.

Tabela 7 - Sintonia de índices em expressões

	Número de linhas retornadas	Tempo de execução (ms)	Desempenho da sintonia
Sem sintonia de índices	2	2531,683	**Tempo de execução reduzido em: 82%**
		2598,484	
		2489,911	
		2488,588	
		2488,160	
Tempo médio de execução (ms)		**2519,365**	
Com sintonia de índices	2	409,786	
		435,411	
		433,620	
		458,837	
		443,736	
Tempo médio de execução (ms)		**436,278**	

Fonte: Autor.

Gráfico 6 - Sintonia de índices em expressões



Fonte: Autor.

Levando em consideração os dados da Tabela 7 e do Gráfico 6, pode-se analisar que, com a aplicação da sintonia de índices nessa consulta, obtivemos um ganho de cerca de 82% em tempo de execução. Esse ganho expressivo se deve ao fato de não haver necessidade de

realizar a execução da função associada a consulta, o que ocorre no momento da criação da estrutura de índices e em atualizações e inserções na tabela autor.

Esse tipo de índice é aconselhável quando o número de atualizações e inserções na tabela envolvida for estritamente pequena, e o número de consultas sejam expressivas.

7.8 RECOMENDAÇÕES DE USO DE ÍNDICES

Após a aplicação das técnicas de sintonia de índices, é possível sugerir algumas recomendações de uso de índices levando em consideração os tipos de dados que foram usados no estudo de caso. Vale ressaltar que são apenas recomendações baseadas em testes realizados em um conjunto específico de dados, e que não garantem a eficiência em todos os casos em que forem empregadas, porém essas recomendações foram aplicadas em um estudo de caso e tiveram melhoras de desempenho em cinco dos seis testes apresentados, o que se considera relevante.

Para uma melhor compreensão das sugestões que seguem, a Tabela 8 apresenta um resumo com alguns dados relevantes, como conjunto de linhas retornadas e o desempenho de cada técnica de sintonia de índices aplicada no estudo de caso.

Tabela 8 - Resumo do estudo de caso

Técnica de sintonia aplicada	Número de registros das tabelas envolvidas	Tabelas envolvidas	Número de registros retornados	Desempenho apresentado
Sintonia de índices com uso *group by*	**2139008**	**1**	**1051**	**Melhora de 30%**
Sintonia de índice classe de operadores com *like*	**1747605**	**1**	**1540**	**Melhora de 18%**
Sintonia de índices com uso *where* e *join*	**1469908 3648**	**2**	**3858**	**Melhora de 85%**
Sintonia de múltiplas colunas	**418695**	**1**	**115152**	**Piora de 7%**
Sintonia de índices parciais	**418695**	**1**	**551**	**Melhora de 93%**
Sintonia de índices em expressões	**1801125 1877992 1747605**	**3**	**2**	**Melhora de 82%**

Fonte: Autor

- Criação de arquivos de índices sobre colunas que fazem uso dos parâmetros *order* e *group by*;
- Uso do operador *LIKE* na forma especificada pela documentação do *PostgreSQL* para que faça uso da estrutura de índices já criada. Além desse fator, o uso de classes de operadores existentes no *PostgreSQL*, podem ser fundamentais para gerar um salto no desempenho da consulta que faz o uso do *LIKE*;
- Criação de arquivos de índices nas colunas usadas como filtro da cláusula *where* e chave de união do *join*;
- Criação de índices parciais, caso determinada consulta seja feita somente sobre um conjunto de dados de determinada coluna, isso resulta num arquivo de índices de menor tamanho, consequentemente menos dados para serem carregados na memória. Porém esse tipo de índice tem um custo maior de criação;
- Criação de índices em expressões melhoram o desempenho de uma consulta de forma expressiva, porém deve-se atentar o seu alto custo de manutenção;
- Todas as recomendações anteriores, tem melhor desempenho em tabelas grandes. No caso de uso desenvolvido, o desempenho de consultas na tabela com menos de quinhentos mil registros, não mostraram ganhos de desempenho relevante;
- Para a criação de arquivos de índices, é necessário realizar uma análise da proporção no número de atualizações (inserção, alteração e deleção) em relação ao número de consultas que são realizadas nas colunas indexadas, pois índices geram custos de manutenção, e isso afeta diretamente o desempenho do banco de dados;
- A recomendação considerada mais relevante, após a aplicação dos testes no caso de uso, é que quanto maior for o conjunto de dados do arquivo de índices que determinada consulta for retornar, menor será a chance desse arquivo de índices gerar ganho de desempenho, e até de ser usado. Na BD3, consultas que retornam mais de 28% dos dados de determinado arquivo de índices acarretam em perda de desempenho, ao contrário do esperado. Na BD2, quando esse número passa de 37% ocorre a mesma situação. Pode-se concluir que quanto maior for o tamanho da tabela envolvida, esse número aumenta um pouco, porém cuidados devem ser tomados quando esse conjunto de dados retornado for expressivamente grande.

8 CONSIDERAÇÕES FINAIS

A sintonia de índices é um processo que requer profundo conhecimento sobre os tipos disponíveis, os dados que estão sendo manipulados e a forma como estão dispostos. Também é fundamental para manter o bom funcionamento do banco de dados, um monitoramento contínuo, analisando sua evolução em termos de volume de dados e o funcionamento de consultas e demais operações.

Para realizar esse monitoramento, pode-se citar a importância do uso do comando EXPLAIN ANALYSE, para realizar testes e análises sobre a execução das consultas, a fim de ajustar da melhor forma o uso de índices em cada banco de dados. Com a utilização deste comando no estudo de caso, foi possível realizar a coleta dos dados que foram posteriormente analisados, e serviram como base para definição dos tipos de índices que foram criados.

A aplicação da sintonia de índices, não significa criar índices para todas as colunas envolvidas em consultas, mas sim, principalmente, quando a consulta retorna um pequeno conjunto de dados em relação ao número total de dados de determinada tabela.

Outro ponto que determina a aplicação da sintonia de índices é a quantidade de atualizações e inserções que determinada tabela sofra em relação as suas consultas. Estruturas de índices tem um custo de criação e manutenção, e sua criação em demasia, pode em vez de resultar em melhoria de desempenho, sobrecarregar ainda mais determinado sistema ou aplicação.

Após o desenvolvimento bibliográfico desse trabalho e a aplicação do conhecimento adquirido no estudo de caso, pode-se concluir que a correta aplicação da sintonia de índices através de pequenos ajustes, pode acarretar em um salto no ganho de desempenho das consultas no banco de dados *PostgreSQL*.

O processo de sintonia de índices é um processo muito importante, que deve ser aplicado constantemente durante o ciclo de vida de um banco de dados. Esses ajustes devem ser feitos de maneira adequada, a fim de perceber quaisquer modificações realizadas, e evitar consequências que possam comprometer a integridade dos dados.

É importante ressaltar, que existem outras formas de sintonias disponíveis, como a sintonia de consultas e a sintonia de SGBD que forma abordadas na seção 2 desse trabalho, e que podem ser combinadas com a sintonia de índices, podendo resultar dessa forma, uma melhora mais significativa em termos de desempenho em cada banco de dados.

8.1 TRABALHOS FUTUROS

Este trabalho foi desenvolvido usando apenas índices *B-tree* devido aos tipos de dados que foram manipulados. Como sugestão para trabalhos futuros, poderia ser desenvolvido um trabalho sobre os demais tipos de índices do *PostgreSQL* como por exemplo os índices GiST e GIN descritos na seção 3.1.

Como esses índices são recomendados para diferentes tipos de dados, geométricos e texto completo e seu funcionamento variar de acordo com a estratégia de indexação usada, ficam como sugestão para um trabalho futuro para verificar o seu comportamento na aplicação da sintonia de índices.

REFERÊNCIAS BIBLIOGRÁFICAS

ALVES, Willian Pereira. **Banco de dados.** São Paulo: Érica Ltda, 2014. 160 p.

BANCO de dados. São Paulo: Pearson Education do Brasil, 2014. 196 p.

BENCHMARKSQL. 2017. © 2017 Slashdot Media. All Rights Reserved.. Disponível em: <https://sourceforge.net/projects/benchmarksql/>. Acesso em: 06 out. 2017.

ELMASRI, Ramez; NAVATHE, Shamkant B.. **Sistemas de banco de dados.** 6. ed. São Paulo: Pearson Education do Brasil, 2011. 766 p. Tradução de: Daniel Vieira.

FRITCHEY, Grant. **SQL Server 2012 Query Performance Tuning.** 3. ed. New York: Springer, 2012. 499 p.

FUENTES, Alain Domíngues; LIFSCHITZ, Sérgio. **Sintonia fina automática com índices parciais.** 2016. 82 f. Dissertação (Mestrado) - Curso de Ciência da Computação, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, 2016. Disponível em: <http://www2.dbd.puc-rio.br/pergamum/tesesabertas/1412705_2016_completo.pdf>. Acesso em: 19 set. 2017.

GILES, Dominic. **Swingbench.** 2010. Disponível em: <http://www.dominicgiles.com/swingbench.html>. Acesso em: 06 out. 2017.

GRAY, Jim. **The Benchmark Handbook for Database and Transaction Systems.** 2. ed. Morgan Kaufmann Publishers Inc. 1993. ISBN 1-55860-292-5. Disponível em: <https://jimgray.azurewebsites.net/BenchmarkHandbook/TOC.htm>. Acesso em: 06 out. 2017.

GROUP, T. P. G. D. **Sobre o PostgreSQL.** 2017. Disponível em: <https://www.postgresql.org/about/>. Acesso em: 17 de out. 2017.

GROUP, T. P. G. D. **Documentação do PostgreSQL 9.3.19.** 2017. Disponível em: <https://www.postgresql.org/docs/9.3/static/index.html>. Acesso em: 17 de out. 2017.

MAGALHÃES, Lucas Bianchi et al. ANÁLISE COMPARATIVA DOS ALGORITMOS DE OTIMIZAÇÃO DE CONSULTAS DO POSTGRESQL. **Colloquium Exactarum,** [s.l.], v. 7, n. 1, p.01-21, 20 mar. 2015. Associacao Prudentina de Educacao e Cultura (APEC). http://dx.doi.org/10.5747/ce.2015.v07.n1.e001x.

MILANI, André. **PostgreSQL:** guia do programador. São Paulo: Novatec Editora, 2008. 389 p.

OLIVEIRA, Rafael Pereira de et al. Projeto e implementação do framework Outer-tuning: auto sintonia e ontologia para bancos de dados relacionais. In: XI BRAZILIAN SYMPOSIUM ON INFORMATION SYSTEM, 11., 2015, Goiânia. **Anais... .** Goiânia: Sbc, 2015. p. 171 - 178. Disponível em: <http://www.portal.inf.ufg.br/sbsi2015/anais>. Acesso em: 14 set. 2017.

RAO, Ch.v.n.sanyasi; KRISHNA, Tiruveedula Gopi. SQL Performance Tuning. **International Journal Of Computer Science Engineering And Technology(Ijcset).** [s.l], dez. 2014. p. 344-346. Disponível em: <http://ijcset.net/docs/Volumes/volume4issue12/ijcset2014041203.pdf>. Acesso em: 16 ago. 2017.

ROB, Peter; CORONEL, Carlos. **Sistemas de banco de dados:** projeto, implementação e gerenciamento. Austrália: Cengage Learning, 2011. xxi, 711 p. ISBN 9788522107865.

SECRETARIA DE TECNOLOGIA DA INFORMAÇÃO. **Portal brasileiro de dados abertos**. Disponível em: <http://dados.gov.br/>. Acesso em: 01 fev. 2018.

SENG, Jia-Lang; Yao, Bing S; Hevner, Alan R. **Requirements-driven database systems benchmark method**. Decision Support Systems 38 (2005) 629 – 648.

SHASHA, Dennis; BONNET, Philippe. **Database Tuning**. San Francisco: Morgan Kaufmann, 2003. 440 p. Impressão Morgan Kaufmann.

SILBERSCHATZ, Avi; KORTH, Hank; SUDARSHAN, S.. **Sistema de banco de dados.** 6. ed. Rio de Janeiro: Elsevier Editora Ltda, 2012. 861 p. Tradução de: Daniel Vieira.

TELLES, Marcelo Josué; GALANTE, Renata; DORNELES, Carina F. **Aplicando tuning no PostgreSQL.** Porto Alegre: Sbc, 2011. 1 v.

THE PGADMIN DEVELOPMENT TEAM (Org.). **PgAdmin III.** 2002 - 2016. Disponível em: <https://www.pgadmin.org/docs/pgadmin3/1.22/index.html>. Acesso em: 27 abr. 2016.

TRAMONTINA, Gregório Baggio. **Database Tuning:** Configurando o Interbase e o PostgreSQL. 2008. Disponível em: <http://www.ic.unicamp.br/~geovane/mo410-091/Ch20-ConfigInterbasePosgres-art.pdf>. Acesso em: 11 ago. 2017.

ANEXO A – COMANDOS SQL PARA A CRIAÇÃO DA BASE DE DADOS BD1

```
-- Table: autor

CREATE TABLE autor
(
codigo numeric(10,0) NOT NULL,
nome character varying(35) NOT NULL,
CONSTRAINT autor_pkey PRIMARY KEY (codigo)
)

-- Table: edicao

CREATE TABLE edicao
(
  codigolivro numeric(10,0) NOT NULL,
  numero character(1) NOT NULL,
  ano integer NOT NULL,
  CONSTRAINT edicao_pkey PRIMARY KEY (codigolivro, numero),
  CONSTRAINT edicao_codigolivro_fkey FOREIGN KEY (codigolivro)
     REFERENCES livros (codigo) MATCH SIMPLE
     ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk_edicao FOREIGN KEY (codigolivro)
     REFERENCES livros (codigo) MATCH SIMPLE
     ON UPDATE NO ACTION ON DELETE NO ACTION
)

-- Table: livroautor

CREATE TABLE livroautor
(
  codigolivro numeric(10,0) NOT NULL,
  codigoautor numeric(10,0) NOT NULL,
  CONSTRAINT livroautor_pkey PRIMARY KEY (codigolivro, codigoautor),
```

```
  CONSTRAINT fk_livroautor FOREIGN KEY (codigolivro)
      REFERENCES livros (codigo) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT fk_livroautor1 FOREIGN KEY (codigoautor)
      REFERENCES autor (codigo) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT livroautor_codigoautor_fkey FOREIGN KEY (codigoautor)
      REFERENCES autor (codigo) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT livroautor_codigolivro_fkey FOREIGN KEY (codigolivro)
      REFERENCES livros (codigo) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)

-- Table: livros

CREATE TABLE livros
(
  codigo numeric(10,0) NOT NULL,
  titulo character varying(45) NOT NULL,
  CONSTRAINT livros_pkey PRIMARY KEY (codigo)
)
```

ANEXO B – COMANDOS SQL PARA A CRIAÇÃO DA BASE DE DADOS BD2

```
-- Table: filiados_partido_y

CREATE TABLE filiados_partido_y
(
  numeroinscricao bigint,
  sigladopartido character(10),
  codigodomunicipio integer,
  zonaeleitoral integer,
```

```
  secaoeleitoral integer,
  datadafiliacao date,
  situacaodoregistro character(20),
  tipodoregistro character(20),
  datadoprocessamento character(20),
  datadadesfiliacao character(20),
  datadocancelamento character(20),
  datadaregularizacao character(20),
  motivodocancelamento character(60),
  CONSTRAINT filiados_partido_y_codigodomunicipio_fkey FOREIGN KEY (codigodomunicipio)
      REFERENCES municipios_filiados (codigodomunicipio) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT filiados_partido_y_numeroinscricao_fkey FOREIGN KEY (numeroinscricao)
      REFERENCES pessoas_filiados_y (numeroinscricao) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)

-- Table: filiados_partido_z

CREATE TABLE filiados_partido_z
(
  numeroinscricao bigint,
  sigladopartido character(10),
  nomedopartido character(50),
  codigodomunicipio integer,
  zonaeleitoral integer,
  secaoeleitoral integer,
  datadafiliacao date,
  situacaodoregistro character(20),
  tipodoregistro character(20),
  datadoprocessamento character(20),
  datadadesfiliacao character(20),
```

```
  datadocancelamento character(20),
  datadaregularizacao character(20),
  motivodocancelamento character(60),
  CONSTRAINT filiados_partido_z_codigodomunicipio_fkey FOREIGN KEY (codigodomunicipio)
      REFERENCES municipios_filiados (codigodomunicipio) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION,
  CONSTRAINT filiados_partido_z_numeroinscricao_fkey FOREIGN KEY (numeroinscricao)
      REFERENCES pessoas_filiados_z (numeroinscricao) MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE NO ACTION
)

-- Table: municipios_filiados

CREATE TABLE municipios_filiados
(
  uf character(2),
  codigodomunicipio integer NOT NULL,
  nomedomunicipio character(50),
  CONSTRAINT pk_municipios PRIMARY KEY (codigodomunicipio)
)

-- Table: pessoas_filiados_y

CREATE TABLE pessoas_filiados_y
(
  numeroinscricao bigint NOT NULL,
  nomedofiliado character(100),
  CONSTRAINT pk_filiados_y PRIMARY KEY (numeroinscricao)
)

-- Table: pessoas_filiados_z
```

```
CREATE TABLE pessoas_filiados_z
(
  numeroinscricao bigint NOT NULL,
  nomedofiliado character(100),
  CONSTRAINT pk_filiados_z PRIMARY KEY (numeroinscricao)
)
```

ANEXO C – COMANDOS SQL PARA A CRIAÇÃO DA BASE DE DADOS BD3

```
-- Table: enem_rs_2015

CREATE TABLE enem_rs_2015
(
  nu_inscricao numeric,
  nu_ano numeric,
  co_municipio_residencia numeric,
  no_municipio_residencia text,
  co_uf_residencia numeric,
  sg_uf_residencia character(2),
  in_estuda_classe_hospitalar numeric,
  in_treineiro numeric,
  co_escola numeric,
  co_municipio_esc numeric,
  no_municipio_esc text,
  co_uf_esc numeric,
  sg_uf_esc character(2),
  tp_dependencia_adm_esc numeric,
  tp_localizacao_esc numeric,
  tp_sit_func_esc numeric,
  nu_idade numeric,
  tp_sexo character(1),
  tp_nacionalidade numeric,
```

co_municipio_nascimento numeric,
no_municipio_nascimento text,
co_uf_nascimento numeric,
sg_uf_nascimento character(2),
tp_st_conclusao numeric,
tp_ano_concluiu numeric,
tp_escola numeric,
tp_ensino numeric,
tp_estado_civil numeric,
tp_cor_raca numeric,
in_baixa_visao numeric,
in_cegueira numeric,
in_surdez numeric,
in_deficiencia_auditiva numeric,
in_surdo_cegueira numeric,
in_deficiencia_fisica numeric,
in_deficiencia_mental numeric,
in_deficit_atencao numeric,
in_dislexia numeric,
in_gestante numeric,
in_lactante numeric,
in_idoso numeric,
in_discalculia numeric,
in_autismo numeric,
in_visao_monocular numeric,
in_sabatista numeric,
in_outra_def numeric,
in_sem_recurso numeric,
in_nome_social numeric,
in_braille numeric,
in_ampliada_24 numeric,
in_ampliada_18 numeric,
in_ledor numeric,
in_acesso numeric,

in_transcricao numeric,
in_libras numeric,
in_leitura_labial numeric,
in_mesa_cadeira_rodas numeric,
in_mesa_cadeira_separada numeric,
in_apoio_perna numeric,
in_guia_interprete numeric,
in_maca numeric,
in_computador numeric,
in_cadeira_especial numeric,
in_cadeira_canhoto numeric,
in_cadeira_acolchoada numeric,
in_prova_deitado numeric,
in_mobiliario_obeso numeric,
in_lamina_overlay numeric,
in_protetor_auricular numeric,
in_medidor_glicose numeric,
in_maquina_braile numeric,
in_soroban numeric,
in_marca_passo numeric,
in_sonda numeric,
in_medicamentos numeric,
in_sala_individual numeric,
in_sala_especial numeric,
in_sala_acompanhante numeric,
in_mobiliario_especifico numeric,
"in_material_especÍfico" numeric,
in_certificado numeric,
no_entidade_certificacao text,
co_uf_entidade_certificacao numeric,
sg_uf_entidade_certificacao character(2),
co_municipio_prova numeric,
no_municipio_prova text,
co_uf_prova numeric,

```
sg_uf_prova character(2),
tp_presenca_cn numeric,
tp_presenca_ch numeric,
tp_presenca_lc numeric,
tp_presenca_mt numeric,
co_prova_cn numeric,
co_prova_ch numeric,
co_prova_lc numeric,
co_prova_mt numeric,
nu_nota_cn numeric,
nu_nota_ch numeric,
nu_nota_lc numeric,
nu_nota_mt numeric,
tx_respostas_cn text,
tx_respostas_ch text,
tx_respostas_lc text,
tx_respostas_mt text,
tp_lingua numeric,
tx_gabarito_cn text,
tx_gabarito_ch text,
tx_gabarito_lc text,
tx_gabarito_mt text,
tp_status_redacao numeric,
nu_nota_comp1 numeric,
nu_nota_comp2 numeric,
nu_nota_comp3 numeric,
nu_nota_comp4 numeric,
nu_nota_comp5 numeric,
nu_nota_redacao numeric,
q001 character(1),
q002 character(1),
q003 character(1),
q004 character(1),
q005 numeric,
```

```
q006 character(1),
q007 character(1),
q008 character(1),
q009 character(1),
q010 character(1),
q011 character(1),
q012 character(1),
q013 character(1),
q014 character(1),
q015 character(1),
q016 character(1),
q017 character(1),
q018 character(1),
q019 character(1),
q020 character(1),
q021 character(1),
q022 character(1),
q023 character(1),
q024 character(1),
q025 character(1),
q026 character(1),
q027 character(1),
q028 character(1),
q029 numeric,
q030 numeric,
q031 numeric,
q032 numeric,
q033 numeric,
q034 numeric,
q035 numeric,
q036 numeric,
q037 numeric,
q038 numeric,
q039 numeric,
```

```
  q040 numeric,
  q041 numeric,
  q042 character(1),
  q043 character(1),
  q044 character(1),
  q045 character(1),
  q046 character(1),
  q047 character(1),
  q048 character(1),
  q049 character(1),
  q050 character(1)
)
```