Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

ELAINE PLANET DA SILVA COSTA

SOLUÇÕES EM BANCO DE DADOS POR MEIO DA TECNOLOGIA TUNING

ASSIS
2010

SOLUÇÕES EM BANCO DE DADOS POR MEIO DA TECNOLOGIA TUNING

ELAINE PLANET DA SILVA COSTA

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito de Curso de Graduação, analisado pela seguinte comissão examinadora:

Orientador: Dr. Alex Sandro Romeo de Souza Poletto

Analisador (1): Fernando Cesar de Lima

ASSIS
2010

ELAINE PLANET DA SILVA COSTA

SOLUÇÕES EM BANCO DE DADOS POR MEIO DA TECNOLOGIA TUNING

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, analisado pela seguinte comissão examinadora:

Orientador: Alex Sandro Romeo de Souza Poletto

Área de Concentração: Sistema de Banco de Dados

Assis
2010

DEDICATÓRIA

A Deus por ser onisciente, onipotente, onipresente e ter me dado a vitória.

À minha mãe que muito se esforçou em todos os sentidos para que eu concluísse essa etapa de minha vida, sem a ajuda dela não seria possível minha permanência neste curso.

Ao meu esposo Wendell e a minha filha Ana Beatriz pela compreensão e paciência de minha ausência todos esses anos.

AGRADECIMENTOS

Ao Prof. Dr. Alex Sandro Romeo de Souza Poletto, pela fiel ajuda, paciência, por se preocupar com o meu aprendizado, pela dedicação em transmitir segurança e conhecimento de forma clara e objetiva, com muito profissionalismo e inteligência, por quem tenho muito respeito e admiração, seu acompanhamento foi fundamental para a conclusão desde trabalho.

Ao meu pai Dirceu Planet da Silva (in Memorian) que me amou muito e não mediu esforços para que um dia eu chegasse até aqui, me ensinou a ser uma pessoa de bem, a acreditar nos sonhos e tudo de valor que hoje tenho em minha vida.

Ao meu irmão Thiago Planet da Silva, formado no mesmo curso, que muito me apoiou, demonstrando sua ajuda no material, nas leituras, no entendimento, fazendo com que em nenhum momento me sentisse sozinha.

Ao meu grande amigo Tiago Bungestab que chegou em minha vida; quando pensei não ter mais jeito em aprender, ele me auxiliou por um período considerável e me trouxe ânimo para continuar.

Ao Rafael Antonio da Silva, meu parceiro nos trabalhos é meu ponto de referência.

À minha Profa. Regina Fumie Eto, por acreditar em mim em todos os momentos, por se tornar minha amiga do coração e por ter dedicado tanto tempo em aulas mesmo de outras matérias somente com a intenção de contribuir.

Ao Prof. Dr. Almir Rogério Camolesi, que desde o primeiro ano, foi amigo, conselheiro de todas as horas, pela disposição, pelas idéias, pelo companheirismo, através das palavras de encorajamento e ensinamentos consegui ter forças para continuar, as quais nunca serão apagadas de minha memória.

À Profa. Dra Marisa Atsuko Nitto, pelos momentos agradáveis, descontraídos, sempre demonstrou as maneiras mais fáceis de estudar e, principalmente, em ter

amor e determinação por nossas escolhas, também sempre com muita disposição em me ajudar, uma amiga que guardarei pra sempre.

Ao meu amigo, que conheci no último ano, Luciano Pelizari, que mostrou o lado profissional de como enfrentar as oportunidades e me ensinou diversas coisas importantes que não conheci antes, sua amizade foi essencial para a construção desse trabalho.

Ao meu amigo e Prof. Leandro Manzoni, pelo incentivo em nunca desistir e que também contribuiu na correção deste trabalho.

À minha amiga Viviane Loiola da Visitação, que com seu exemplo de determinação, me apoiou em todos os momentos de dificuldade, me incentivando e me ajudando em tudo o que foi possível.

Ao meu amigo Jovian Regis da Silva, que mesmo em meio a tantas ocupações em sua vida, se prontificou a me dar uma grande ajuda e foi essencial para a finalização deste trabalho.

É claro não poderia deixar de citar aqui meu Prof. Fernando Lima, que sempre ouviu nossas propostas, idéias, comentários, sempre demonstrou muito respeito conosco, trouxe muita alegria, humildade, paz, sinceridade e praticidade, uma pessoa em quem podemos nos espelhar sem medo, sua maneira de viver contagia a todos à sua volta, a quem desejo meus sinceros votos de sucesso.

A todos os meus amigos que me ajudaram diretamente ou indiretamente para a conclusão deste trabalho, nunca os esquecerei.

Meu muito obrigado!

O que eu ouço eu esqueço. O que eu vejo, eu lembro. O que eu faço, eu entendo (Anônimo).

RESUMO

Este trabalho tem por objetivo apresentar a tecnologia Tuning, explicando seu funcionamento e principais características como: arquitetura, topologias, aplicações e dificuldades técnicas enfrentadas. Para tal, será desenvolvido um protótipo para demonstração prática da Instrução SELECT, no sentido de mostrar a garantia de performance, rapidez e integração com base no Sistema Gerenciador de Banco de Dados PostgreSQL, quando do uso das cláusulas Inner Join e Where, bem como de Índices Complementando, o DBA, fará diagnóstico analisando a arquitetura, fábrica de Sistema de Banco de Dados, códigos escritos do projeto para iniciar as otimizações.

Palavras-chave: Tuning, Otimização, Indexação, Inner Join, Cláusula Where.

ABSTRACT

This paper aims to present the Tuning technology, explaining its operation and main characteristics such as architecture, topologies, applications and technical difficulties encountered. This will be a prototype for practical demonstration of the SELECT statement, to show the security performance, speed and integration-based System Manager Database PostgreSQL, when the use of Inner Join and Where clauses, as well as Indices addition, the DBA, will analyze the diagnostic architecture, works of System Database, code written to start the project optimizations.

Keywords: Tuning, Optimization, Indexing, Inner Join, WHERE clause.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SGBD	Sistema Gerenciador de Banco de Dados
SBD	Sistema de Banco de Dados
DDL	Linguagem de Definição de Dados (Data Definition Language)
DML	Linguagem de Manipulação de Dados (Data Manipulation Language)
DBA	Administrador de Banco de Dados
SQL	Structure Query Language
DER	Diagrama de Entidade-Relacionamento

SUMÁRIO

1. INTRODUÇÃO

Atualmente, a demanda por Sistemas de Gerenciamento de Banco de Dados (SGBD) cresce diariamente. A maioria dos fabricantes de SGBD oferece uma ferramenta para que a sintonia (tuning) seja feita de maneira automática. Entretanto, mesmo com tais ferramentas, muitas empresas optam pela abordagem estática (*offline*) na solução do problema apontado, tudo isso é transferido ao Administrador de Banco de Dados (DBA) que tem uma importante decisão sobre as modificações necessárias. É fundamental garantir a segurança, performance e um excelente funcionamento dos dados, de modo que se acontecer algum problema, seja possível recuperar os dados rapidamente evitando grandes perdas.

As aplicações de bancos de dados têm se tornado cada vez mais complexas e variadas. Estas podem ser caracterizadas por manipularem grandes volumes de dados e pela demanda por baixo tempo de resposta das consultas e por alta produtividade das transações. Neste contexto, a sintonia (*tuning*) das estruturas de índice tem se revelado ainda mais importante, influindo diretamente no desempenho dos sistemas de bancos de dados. Tudo isso requer um grande conhecimento dos detalhes de implementação dos SGBDs em geral, características, dados armazenados, como são relacionadas às tabelas e principais consultas etc.

A metodologia Tuning ou tarefa Tuning foi desenvolvida para auxiliar os DBAs, a obterem sucesso na realização “Tuning Performance” em vários pontos já citados acima. É levado em consideração, para a solução de problemas nesta fase do projeto, fazer ajustes durante o processo, em vez de esperar para ajustar depois da implementação do sistema (Time-Out). Isso é chamado de abordagem pró-ativa, na fase de desenvolvimento. Neste ponto uma forma de melhorar a performance do banco de dados em questão, seria através da realocação de memória e ajuste de I/O ou ainda adicionando novos recursos.

Tuning significa “Afinação”, trazendo para a linguagem de informática, otimizações em toda parte do banco de dados, e qualquer modelo relacional. Tuning é interativo, ganho de performance feita por determinada otimização pode influenciar em outros passos para outros ganhos de performance.

Para um bom resultado da metodologia Tuning, pode-se sugerir algumas regras:

- Criação de testes repetitivos;
- Guardar o registro de cada *update* do script;
- Não mude os códigos por adivinhação;
- Pare de realizar o ajuste quando alcançar os objetivos traçados;
- Evite preconceitos.

Lembrando que a tecnologia pode atuar tanto no SGBD como direta e internamente no Banco de Dados. Os usuários devem conhecer detalhadamente as rotinas, em que situações estão as manutenções, e onde se pretende chegar. As diferentes soluções de Tuning estão dentro de aplicações, redes, clientes, bancos de dados, códigos, etc. Problemas de performance ocorrem quando uma função consome muito mais tempo para executar em relação ao tempo permitido. Isso ocorre por razão de um recurso, particular, ser insuficiente ou inadequado. O recurso pode ser um recurso físico, como *buffers* de memória disponíveis para armazenar blocos de dados ou artificial, como um *lock*.

1.1 OBJETIVO

O objetivo do trabalho é conhecer os conceitos da Tecnologia Tuning, bem como de realizar testes de performance, aplicando técnicas mais específicas na área de otimização de Consultas, que acredito ser o maior problema em comum que as empresas enfrentam. Utilizando o Sistema Gerenciador de Banco de Dados PostgreSQL, como alternativa para a utilização na infra-estrutura do banco de dados, proporcionará segurança e agilidade, obtendo sensível redução dos custos operacionais, por ter código aberto e assim reutilizado em todas as áreas comerciais e acadêmicas.

1.2 JUSTIFICATIVA

Atualmente, a maioria das empresas de médio porte não utiliza nenhum planejamento de acesso aos dados. Esta metodologia vem sendo muito bem aproveitada nos ambientes de grande extensão de dados, trazendo uma estatística surpreendente. Os problemas aumentam a cada dia e essa proporção o ser humano também deve crescer em conhecimento e capacidade de validar os ajustes de desempenho, porém isso não tem se findado, já que a falta profissional no mercado de trabalho, uma vez que, para aplicação do planejamento Tuning é fundamental que o DBA tenha um conhecimento abrangente de todas as transações de dados, para que toda degradação de performance tenha os números iguais a zero.

1.3 MOTIVAÇÕES

O interesse da tecnologia Tuning surgiu principalmente por não existir em muitos trabalhos desenvolvidos nessa linha, já que, cada vez que se aplica a metodologia, ela terá um estágio diferente, visto que aborda extensões de soluções através das otimizações apresentadas por problemas de performance. Tuning é uma consultoria bem aplicada, previne custos e apresenta rápidos resultados. Espera-se poder contribuir tanto para a área acadêmica quanto a área profissional, definindo soluções para empresas corporativas.

1.4 ESTRUTURA DO TRABALHO

Este trabalho foi organizado em seis capítulos, sendo o primeiro esta introdução.

No segundo capítulo serão apresentadas as fundamentações teóricas sobre Tuning.

No terceiro capítulo serão apresentados conceitos e noções sobre Sistemas Gerenciadores de Banco de Dados, bem como da Linguagem SQL.

No quarto capítulo será apresentada a proposta de trabalho e como a mesma deve ser aplicada. No quinto capítulo será apresentado um Estudo de Caso.

No sexto capítulo serão apresentadas as considerações finais e as projeções futuras.

2. FUNDAMENTAÇÃO TEÓRICA SOBRE TUNING

Neste capítulo serão abordados conceitos da tecnologia Tuning, passando pela arquitetura, topologia, e algumas vantagens e desvantagens.

2.1 A ARQUITETURA TUNING

Pode-se traduzir “Tuning“, como “ajustar”. Antes de se fazer qualquer alteração no SGBD é fundamental diagnosticar quais são os problemas causados pelo software ou pelo hardware. Muitas vezes o hardware não é adequado ou a aplicação não possui um bom fluxograma, isso evitará que o cliente final, ao testar, o software, reclame alguma deficiência. Por isso, nunca se pode começar a aplicação da Tecnologia Tuning sem entender muito bem a arquitetura do banco de dados que se está usando.

Para se adquirir grande sucesso na performance do Banco de Dados, uma das dicas é ter uma tecnologia bem planejada, mas, mesmo assim, com o tempo pode degradar com o uso, e para isso o Tuning previne qualquer ineficiência no Banco de Dados.

Tuning nada mais é do que ajustar partes das aplicações SGBD, transações do Banco de Dados ou usabilidade das Redes. É totalmente utilizado e recomendado para redução de custos, agilidade, aumento de produtividade e segurança. Uma solução através da arquitetura, como por exemplo, é melhorar a performance pela realocação de memória e ajuste de I/O, garantindo sempre a integridade SGBD. Uma vez que o DBA conheça toda a Arquitetura depois de normalizar os dados, ele pode entretanto necessitar desnormalizar os dados por razões de performance. Segundo Koch e Loney (2002), nenhuma aplicação principal rodará na terceira forma normal. A aderência muito rígida para projetos de tabelas relacionais trarão performance ruim. Um problema ocorrerá com pesquisas que retornam um número muito grande de colunas. Estas colunas são normalmente espalhadas entre várias tabelas, forçando a junção delas durante a pesquisa. Se uma das tabelas da junção

for grande, então a performance de toda a pesquisa poderá sofrer, e assim entre outras formas de desnormalização.

2.2.1 Cliente / Servidor

Uma dica muito importante diz respeito à rede, se ela não for confiável e rápida, será difícil fazer com que a aplicação se comporte de forma adequada; outra dica, que se deve levar em consideração, refere-se ao fluxo de dados que circula pela rede, já que independentemente da velocidade disponível, qualquer redução nesse fluxo trará benefícios para a performance de todas as aplicações. A capacidade da rede é provavelmente o fator de maior peso na performance de uma aplicação distribuída. Por exemplo, se a comunicação entre seus equipamentos se processa a uma velocidade baixa no caso de uma conexão via linha discada, deve-se esperar encontrar problemas de performance, principalmente nos momentos em que for necessário trafegar um volume maior de dados.

Um problema muitas vezes ignorado pelos projetistas de aplicações em ambiente cliente/servidor, é a duração da conexão entre ambos, em vários ambientes de desenvolvimento quem gerencia as conexões é o próprio ambiente de execução (“ run-time”) da linguagem utilizada. Para se estabelecer uma conexão depende muito da complexidade de sua rede, então é necessário determinar o endereço físico do servidor; se a rede for pequena, isto é, muito rápido, mas se for baseada em TCP/IP com centenas de servidores, esta operação pode ser lenta. Eventualmente uma rede mal configurada, ou um componente que não funciona como o esperado pode causar queda de performance nas aplicações. As empresas atualmente quase não investigam sua rede porque julgam ser sofisticadas o que não é verdade, com apenas um componente estando lendo pode sim, prejudicar a performance da rede como um todo. Se as medições de tempo de execução dos testes no cliente e no servidor coincidirem, então não há problema com a rede, mas se o tempo for grande, então o problema é com o projeto ou com uso do banco de dados, mas se ainda o tempo entre um e o outro forem totalmente diferentes, então é altamente provável que exista um problema de performance na rede.

A Figura 1 apresenta toda a tecnologia recomendada para ‘tuning’:

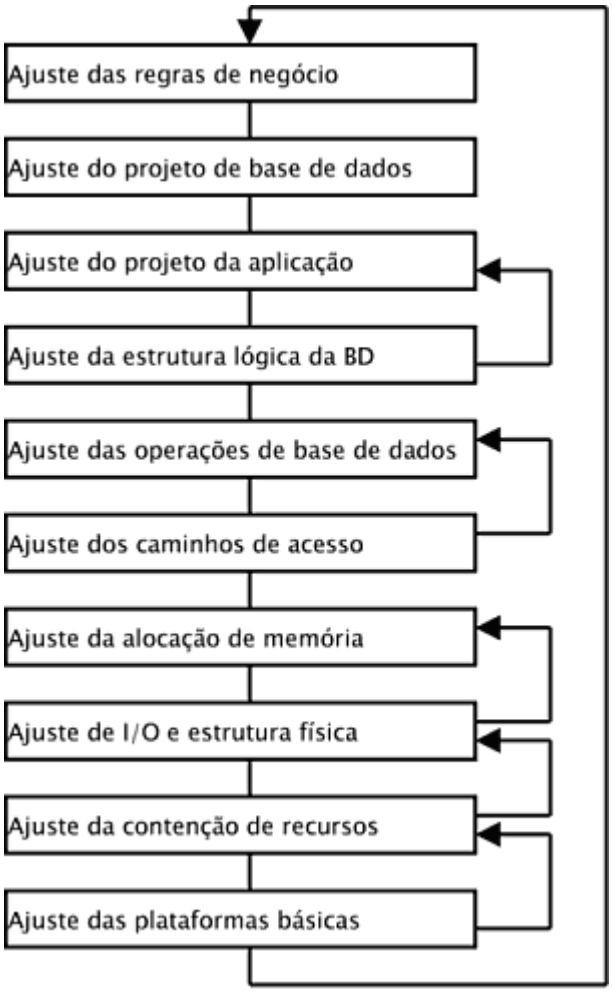


Figura 1. Estrutura da Tecnologia (disponível em http://www.batebyte.pr.gov.br)

A seguir são descritas cada uma das fases recomendadas e apresentadas na Figura 1:

- Ajuste das regras de negócio: Problemas de performance encontrados pelo DBA podem ser causados por problemas no projeto e implementação ou regras de negócio inapropriadas. Por exemplo, a função de negócio de impressão de cheques. O requisito real é pagar em dinheiro às pessoas; o requisito não é necessariamente imprimir pedaços de papel. Poderia ser bastante complicado imprimir milhares de cheques por dia. Seria relativamente mais fácil gravar os depósitos de pagamento em uma fita que seria enviada ao banco para processamento.
- Ajuste do Projeto da Base de Dados: Em fase de projeto, é muito importante declarar quais aplicações serão importantes para o sucesso de performance,

fazer uma análise detalhada dos dados, quais os atributos, chaves estrangeiras, relações entre tabelas, índices, tudo para eliminar as redundância, é feito uma série de caminhos, exceto as chaves primárias que devem ser declaradas apenas uma vez.

- Ajuste do Projeto da Aplicação: É feito pelo analista e projetista, refere-se a um processo de negócio que nada mais é uma aplicação particular do projeto (decisões individuais) como por exemplo salvar os dados em memória cache para evitar a recuperação dos dados várias vezes ao dia.
- Ajuste da Estrutura lógica do Banco de dados: Agora é hora de refinar passo a passo cada parte da estrutura, ajustar os índices, criar novos de acordo com a necessidade da aplicação.
- Ajuste das Operações da Base de Dados: Verifica-se que o otimizador do SGBD e o controle de "*locks*", para entender o processamento da pesquisa, são muito importantes para escrever os comandos SQL efetivamente.
- Ajuste dos Caminhos de Acesso: Esta fase é fundamental dentro do projeto, com certeza a localização física dos dados não é tão importante como seu lugar lógico dentro do projeto da aplicação. A aplicação deve saber onde buscar os dados e retorná-los para a pesquisa, aqui refina-se o caminho de pesquisa através da SQL tornando o banco de modo eficiente. Ex: clusters, hash, índices B-tree, índices bitmap.
- Ajuste de alocação de Memória: Nesta parte, o DBA deverá tomar muito cuidado na hora de alocar a memória para não cometer o terrível erro de alocar mais memória do que realmente a aplicação necessita. Isso causará um grande problema com o Sistema Operacional ocasionando muitos swappings que afetará na queda de performance. Definimos também a estrutura do Servidor PostgreeSQL, como dicionário de dados, library, cachê, buffer cachê, entre outras.
- Ajuste de I/O e estrutura Física: Envolve distribuir dados em discos diferentes para distribuir I/O e evitar contenção de disco. Criar *extents* grandes o suficiente para seus dados e evitar dinâmicas das tabelas.

- Ajuste de Contenção de Recursos: A concorrência de processos de vários usuários também influencia na performance. A espera para que o processo seja disponibilizado deve acontecer um por vez, se acontecer de muitos processos ficarem em andamento acarreta uma grande lentidão, é melhor receber o resultado por escala de modo que não pare, do que receber a resposta de todos de uma só vez.
- Ajuste de Plataformas Básicas: De acordo com a versão do Sistema Gerenciador de Banco de Dados, os parâmetros podem ser diferentes em relação ao Sistema Operacional.

2.2. CONCEITO DE OTIMIZAÇÃO

Otimizar um processo pode significar eliminar um índice ineficiente, implementar novos filtros ou alterar os parâmetros da cláusula *join* das *queries* em execução.

Segundo Date (2004, p. 456), a Otimização representa ao mesmo tempo um desafio e uma oportunidade para sistemas relacionais: um desafio, por que a otimização é uma exigência, caso o sistema espera atingir um desempenho aceitável; uma oportunidade, pois é precisamente um dos pontos fortes da abordagem relacional o fato de que as expressões relacionais estão em um nível semântico suficientemente alto para a que a otimização seja viável em primeiro lugar. Ao contrário, em um sistema não relacional, em que requisições dos usuários são expressas em um nível semântico mais baixo, qualquer “otimização” deve ser feita manualmente pelo usuário. E, se, o usuário tomar uma decisão equivocada não haverá nada de que o sistema possa fazer para melhorar a situação.

Conforme Fernandes (2003, p. 615), a preocupação com a qualidade do trabalho realizado é um hábito que se deve acompanhar o desenvolvedor em todas as etapas do planejamento de um sistema de computador. O refinamento da performance não pode ser feito depois que o sistema estiver em produção. Para obter bons resultados em tempo de resposta, desempenho e segurança, deve-se avaliar a performance a tempo de análise, desenvolvimento e implementação.

2.2.1. Otimizador

O Otimizador é o responsável pela definição do plano de acesso aos dados, ou seja, escolhe a opção mais eficiente para obtenção da informação. O melhor a ser feito ao SGBD é desenvolver, por exemplo, um otimizador de consultas automático, assim ele analisa dentre as disponíveis qual a melhor opção, de modo que formule a consulta. A intenção é fazer com que o otimizador seja mais eficiente do que o DBA. Há várias razões que podem ser citadas:

- O número de valores distintos de cada tipo;
- O número atual de tuplas que aparecem em cada *RelVar* básica;
- O número atual de valores distintos que aparecem em cada atributo de cada *RelVar* básica;
- O número de vezes em que ocorre cada um desses valores em cada um desses e atributos assim por diante.

O otimizador é um programa, por isso é muito mais paciente e eficiente do que um usuário humano. Dentro do ambiente otimizador é possível ser avaliado inúmeras referências e diferentes formas de interpretação, por isso, pode-se compará-lo a um conjunto de conhecimentos e serviços dos melhores programadores, visto que para um ser humano não seria possível guardar tantas informações com profundidade de detalhes. Assim pode-se afirmar que um otimizador precisa de vários ajustes de acordo com cada SGBD e necessidade não há uma regra.

2.2.2. Otimização de Consulta (Visão Geral)

Segundo Silberschatz et al. (2006, p. 383), Otimização de Consulta é o processo de selecionar o plano de avaliação de consulta mais eficiente dentre as muitas estratégias normalmente possíveis para o processamento de determinada consulta, especialmente se esta for complexa.

Não espera que os usuários escrevam suas consultas de modo que possam ser processadas de forma eficiente. Em vez disso, espera-se que o sistema construa um plano de avaliação de consulta que reduza seu custo. Nesse caso a otimização da

consulta entra em ação.

Um aspecto da otimização ocorre no nível da álgebra relacional, em que o sistema tenta encontrar uma expressão que seja equivalente à expressão dada, para que execução seja mais eficiente. Outro aspecto é a seleção de uma estratégia detalhada para o processamento da consulta, como a escolha do algoritmo a ser usada para a execução de uma operação, a escolha dos índices específicos a utilizar, e assim por diante.

A Figura 2 ilustra os quatro estágios do processamento de uma consulta:

1- Converter a consulta para algum formato interno.
2- Converter para a forma canônica.
3- Escolher procedimentos candidatos de baixo nível.
4- Gerar planos de consultas e escolher o mais barato.

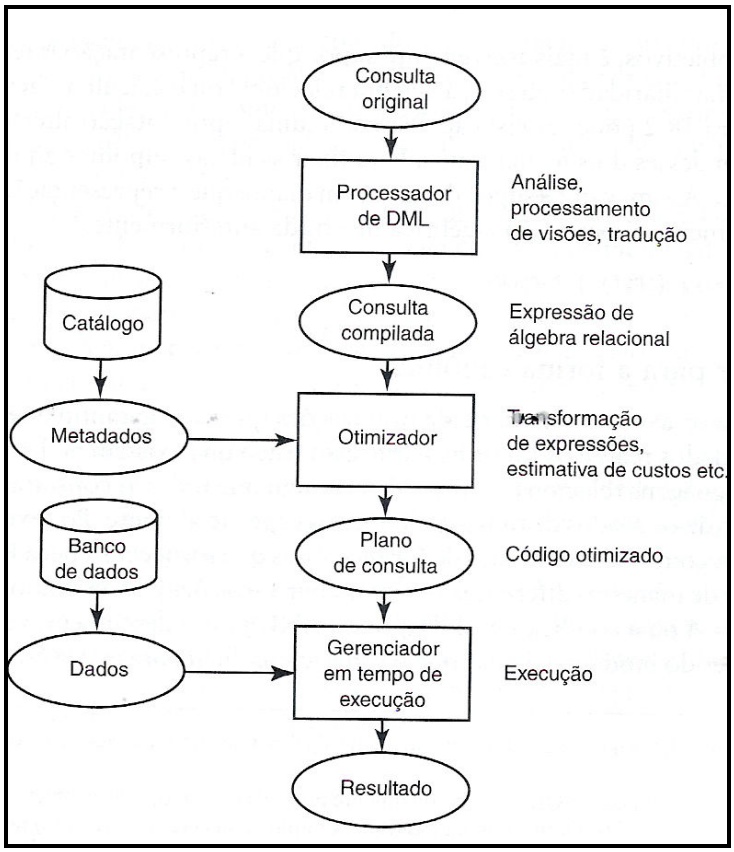


Figura 2. Visão Geral do Processamento de uma Consulta (DATE, C.J, 2004)

2.3 VANTAGENS E DESVANTAGENS DA OTIMIZAÇÃO

A tarefa Tuning é bastante vantajosa, já que permite ser analisada passo a passo, cada diagnostico indicado como ineficiente. Assim que é detectado o problema, a tarefa Tuning pode ser aplicada em qualquer parte do Sistema de Banco de Dados, no Banco de Dados, Sistema Operacional e no Hardware. È evidente que as vantagens de se utilizar a Metodologia são muito maiores em relação as desvantagens, mas elas existem.

2.3.1 Vantagens

Pode-se começar exemplificando a Tarefa Tuning nas consultas, para obter sucesso de performance pode-se criar índices para as tabelas, utilizar as cláusulas WHERE com eficiência, JOINS, ORDER BY ou GROUP BY uma vez que se esta num ambiente de um banco de dados com fluxo grande de informações.

2.3.2 Desvantagens

O Tuning requer tempo, espaço e custo. Por um lado as consultas aumentam a performance do banco de dados, mas por outro, elas diminuem as exclusões, atualizações e inserções no banco, tornando assim muito lenta a parte escrita de manutenção do banco de dados. Isso acontece porque cada vez que é feita alterações no banco de dados isso implica em mudança também no índice, então quanto mais se faz movimentações no banco de dados, um índice terá que ser renovado e, com isso, volta-se a ter um novo problema degradando a performance.

Outro problema relevante diz respeito a um índice, ele ocupa um espaço considerável na memória e pode acontecer de exceder o seu tamanho limite, pode haver mais de um índice em uma determinada tabela, mas na hora de serem armazenados, os índices permanecem no mesmo arquivo. Tudo isso dependerá muito de quantas e quais tabelas estarão sendo usadas e também qual arquitetura está-se adotando para este trabalho.

3. SISTEMAS DE GERENCIAMENTO DE BANCO DE DADOS

Neste capítulo serão abordados conceitos básicos do funcionamento de um SGBD, também de um Banco de Dados e as funções do DBA Tuning.

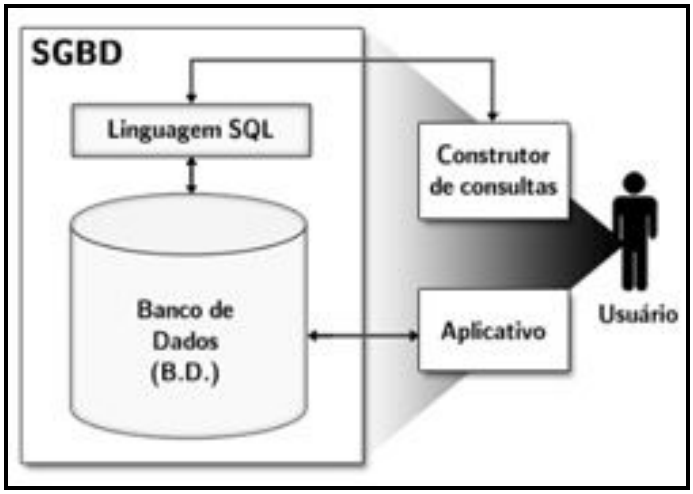


Figura 3. Visão Geral do Sistema de Banco de Dados
(disponível em http://www.socesp.org.br)

A seguir são descritas as funções do SGBD:

- Definição de Dados: O SGBD é uma coleção de programas que permite ao usuário definir, construir e manipular bases de Dados para todas as finalidades.
- Manipulação de Dados: O SGBD também é capaz de lidar com requisições do usuário para buscar, atualizar ou excluir dados existentes no banco de dados, ou ainda acrescentar novos dados ao banco de dados. Para isso ele utiliza a DML (Data Manipulation Language).
- Otimização e Execução: As requisições de DML planejadas ou não, devem ser processadas pelo otimizador, cuja finalidade é determinar o modo eficiente de implementar a requisição.
- Segurança e Integridade de Dados: O SGBD é o responsável por monitorar as requisições feitas pelo usuário na tentativa de violar as restrições de segurança e integridade definidas pelo DBA.

- Recuperação de Dados e Concorrência: Um componente do software chamado de gerenciador de transações ou monitor de TP (Transaction Processing) deverá impor certos controles de recuperação e concorrência.
- Dicionário de Dados: O dicionário de Dados pode ser considerado um banco de dados reserva (somente do sistema e não de acesso ao usuário)

3.1 ÁLGEBRA RELACIONAL

A álgebra relacional é uma linguagem de consulta procedural. Ela consiste em um conjunto de operações que tomam uma ou duas tabelas como entrada e produzem uma nova tabela como resultado. Essas operações recebem o nome de conjuntos. Existem seis operações fundamentais na álgebra relacional, agrupadas em dois tipos: unárias (selecionar, projetar, renomear) e binárias (produto cartesiano, união e diferença de conjuntos), além dessas que são fundamentais existem também, intersecção de conjuntos, ligação natural, dentre outras, que são definidas em termos das operações fundamentais.

Sendo assim pode-se combinar mais de uma expressão algébrica, em uma única operação, um sendo utilizado como entrada para outra operação. A seguir, exemplos de como obter resultados através da álgebra relacional com operações básicas. Começando com a **operação de projeção** que é considerada uma operação unária, será utilizada três tabelas para a demonstração do conceito.

Tabela Cargo

CDCARGO	NUMCARGO	VLSALARIO
C1	Balconista	R$ 450,00
C3	Caixa	R$ 800,00
C7	Gerente	R$ 2.500,00
C2	Vigia	R$ 510,00
C5	Aux. TI	R$ 1.500,00
C4	Aux. Cobrança	R$ 350,00
C6	Vendedor	R$ 510,00
C8	Publicitário	R$ 1.000,00
C9	Assistente	R$ 1.000,00

Tabela Departamento

CDDEPTO	NUMDEPTO	RAMALTEL
D1	Assist. Técnica	2246
D2	Estoque	2589
D3	Administração	2772
D4	Segurança	1810
D5	Vendas	2599
D6	Cobrança	2688
D7	Marketing	2790

Tabela Funcionário

NumReg	NomeFunc	DtAdmissão	Sexo	CdCargo	CdDepto
101	Pedro Seródio	25/03/2000	M	C3	D5
104	Ademir Sampaio	10/03/2002	M	C4	D6
134	Jaqueline Albuquerque	18/03/2003	F	C5	D1
121	Marta de Oliveira	24/05/2004	F	C3	D5
195	Maria de Souza	08/02/2003	F	C1	D5
139	João da Silva	15/12/2007	M	C4	D6
123	Larissa Silva	24/05/2008	F	C7	D3
148	Sergio Nogueira	26/09/2010	M	C6	D5
115	Antonio Augusto	26/09/2003	M	C9	D3
22	Sandra Bonifácio	10/10/2008	F	C8	D7
77	Patricia Braz	23/05/2006	F	C2	D4
56	Aline Camargo	12/05/2009	F	C6	D5
171	Viviane Araújo	30/03/2006	F	C1	D5

Figura 4. Estrutura de Tabelas Funcionário

Em Banco de Dados, a **Operação Projeção** é indicada por π (a letra grega *pi*) que define um conjunto (tabela) em que há um elemento para cada elemento (linha) do conjunto (tabela) de entrada, e a estrutura dos membro do conjunto (tabela) resultante é definida nos argumentos da operação. Pode também ser entendida como uma operação de filtro. **Operador de Projeção** sempre atua somente em uma tabela, seja ela do nosso banco de dados ou uma tabela resultante de outra operação relacional executada, são colunas determinadas, ou seja, especificas.

Então: πNomeFunc – (Funcionário) conjunto de entrada

↳ Argumento da Operação

O resultado da solicitação dessa consulta será:

NomeFunc
Pedro Seródio
Ademir Sampaio
Jaqueline Albuquerque
Marta de Oliveira
Maria de Souza
João da Silva
Larissa Silva
Sergio Nogueira
Antonio Augusto
Sandra Bonifácio
Patricia Braz
Aline Camargo
Viviane Araújo

Tabela 1. Resultado da Seleção da Coluna Funcionário

Ou também fazer da seguinte forma:

Π NOME COLUNA1, NOME COLUNA2,...NOME COLUNA(Nome Tabela)

Neste estado em uma única operação retornarão duas colunas específicas.

NomeFunc	DtAdmissão	Sexo
Pedro Seródio	25/03/2000	M
Ademir Sampaio	10/03/2002	M
Jaqueline Albuquerque	18/03/2003	F
Marta de Oliveira	24/05/2004	F
Maria de Souza	08/02/2003	F
João da Silva	15/12/2007	M
Larissa Silva	24/05/2008	F
Sergio Nogueira	26/09/2010	M
Antonio Augusto	26/09/2003	M
Sandra Bonifácio	10/10/2008	F
Patricia Braz	23/05/2006	F
Aline Camargo	12/05/2009	F
Viviane Araújo	30/03/2006	F

Tabela 2. Operação de Projeção de três colunas

Caso seja preciso retornar apenas os nomes do sexo masculino usa-se então o **Operador de Seleção** que é indicada pelo símbolo δ (letra grega sigma) produz um subconjunto de entrada idêntico ao anterior, mas apenas com elementos do conjunto de entrada que atendem a uma determinada condição chamada de predicado. Um

detalhe importante é que esse subconjunto gerado possui todos os atributos do conjunto de entrada (tabela funcionários)

Ficando assim:

Tabela Funcionário					
NumReg	**NomeFunc**	**DtAdmissão**	**Sexo**	**CdCargo**	**CdDepto**
101	Pedro Seródio	25/03/2000	M	C3	D5
104	Ademir Sampaio	10/03/2002	M	C4	D6
139	João da Silva	15/12/2007	M	C4	D6
148	Sergio Nogueira	26/09/2010	M	C6	D5
115	Antonio Augusto	26/09/2003	M	C9	D3

Tabela 3. Resultado Operação de Seleção

Analogamente é comprovado que retornaram todas as colunas da tabela original, funcionários, das linhas desta tabela que satisfazem a condição, predicado sexo = ‘M’.

Caso seja preciso retornar o nome completo do funcionário, a data de admissão e também o salário, será preciso direcionar a consulta para o que é chamado de **Produto Cartesiano**, que referência mais de uma tabela. Uma vez que, o nome e a data de admissão fazem parte da relação funcionário, o salário existe apenas em cargos.

3.1.1 Produto Cartesiano

O conceito da notação na forma (‘conjunto1 x conjunto2’) o resultado do produto cartesiano de duas tabelas é uma terceira tabela, somando todas as combinações possíveis entre os elementos das tabelas originais. Essa tabela resultante possuirá um número de colunas que é igual à soma das quantidades de colunas das duas tabelas iniciais, e um número de linhas igual ao produto do número de suas linhas.

Assim, no caso acima citado o produto cartesiano de uma tabela A que possua 6 colunas e 13 linhas com uma tabela B onde existem 3 colunas e 9 linhas, a tabela resultante terá 6+3= 9 colunas e 13*9= 117 linhas. Portanto, cada linha dessa tabela corresponderá à concatenação de uma linha da primeira tabela com uma linha da segunda. O produto cartesiano não é muito usado como um fim em si mesmo, ou seja, dificilmente estaremos interessados em saber quais são todas as combinações possíveis entre as linhas de duas tabelas, pois a utilidade prática desse tipo de conhecimento é muito discutível. Por isso, forma primitiva é bastante aproveitada para origem da terceira tabela, geralmente é mais utilizada como maneira de junção.
Ex.: **p** NmFunc, DtAdm, VrSalário (**s** funcionário.CdCargo = cargo.CdCargo (funcionário x cargo))
Então, <nome-da-relação>.<nome-da-coluna>. Usando a operação de projeção fica assim para solucionar este caso: *funcionário.CdCargo = cargo.CdCargo*. Testará todas as possibilidades da tabela funcionário para a tabela cargo. Então tenho:

Tabela Func_Cargo

NumReg	NomeFunc	DtAdmissão	Sexo	CdCargo	CdDepto	CdCargo	NumCargo	VLSalario
101	Pedro Seródio	25/03/2000	M	C3	D5	C3	Caixa	R$ 800,00
104	Ademir Sampaio	10/03/2002	M	C4	D6	C4	Aux. Cobrança	R$ 350,00
134	Jaqueline Albuquerque	18/03/2003	F	C5	D1	C5	Aux. TI	R$ 1.500,00
121	Marta de Oliveira	24/05/2004	F	C3	D5	C3	Caixa	R$ 800,00
195	Maria de Souza	08/02/2003	F	C1	D5	C1	Balconista	R$ 450,00
139	João da Silva	15/12/2007	M	C4	D6	C4	Aux. Cobrança	R$ 350,00
123	Larissa Silva	24/05/2008	F	C7	D3	C7	Gerente	R$ 2.500,00
148	Sergio Nogueira	26/09/2010	M	C6	D5	C6	Vendedor	R$ 510,00
115	Antonio Augusto	26/09/2003	M	C9	D3	C9	Assistente	R$ 1.000,00
22	Sandra Bonifácio	10/10/2008	F	C8	D7	C8	Publicitário	R$ 1.000,00
77	Patricia Braz	23/05/2006	F	C2	D4	C2	Vigia	R$ 510,00
56	Aline Camargo	12/05/2009	F	C6	D5	C6	Vendedor	R$ 510,00
171	Viviane Araújo	30/03/2006	F	C1	D5	C1	Balconista	R$ 450,00

Tabela 4. Tabela Funcionário

3.1.2 Operação Renomear

Nesta operação, uma tabela é utilizada sempre que ela aparece mais de uma vez em uma consulta, como ao contrário dos bancos de dados, o resultado de uma operação em álgebra relacional não tem nome, também utiliza-se a operação *rename* para nomear um resultado. Essa operação é representada pela letra *p* (rho). Por exemplo:

p Cliente2 (Cliente)

Tabela Funcionário		
NumReg	**NomeFunc**	**nasc**
1	Pedro Seródio	25/03/1975
2	Ademir Sampaio	10/03/2002
3	Jaqueline Albuquerque	18/03/2003
4	Marta de Oliveira	24/05/2004
5	Maria de Souza	25/03/1975

Tabela 5. Resultado da Operação Renomear

Então:
Selecionar o cliente para fazer a comparação do campo nascimento e ser projetado o resultado posteriormente trazendo o outro cliente que possui o nome idêntico.

δ nomecliente = ‘ Pedro Serôdio’ (Cliente)

Projetar a coluna desejada:

Π nasc ( )

Concluindo terá:

Π nasc (δ nomecliente = ‘ Pedro Serôdio’ (Cliente))

Chamado de Cliente2, o terceiro passo é renomeado para resultado 2, em seguida o próximo passo seleciona as linhas do resultado2 e as linhas iguais às de Pedro

Serôdio aparecerão como resultado o que será chamado de resultado 3. Posteriormente tem-se a projeção do resultado 3.

(δ Nomecliente = “ Pedro Serôdio”(cliente)) x (p cliente2(cliente))

Cliente2.nasc = nasc = cliente.nascclíente(resultado 2) **Resultado 3**

Π Cliente2.nomecliente. Cliente2.nasc, Cliente2.nascclíente (Resultado 3)

Resultando:

Nasc
25/03/1975

3.1.3 Operação de União (Binária)

Essa operação produz uma terceira tabela onde o resultado são todas as linhas da primeira tabela e também todas as linhas da segunda tabela e os operandos desta tabela devem ser compatíveis, assim, todas as linhas que forem idênticas nas duas tabelas, em uma única vez na tabela resultante. Representação:

Tabela 1 U Tabela 2

Exemplo:

Mostrar o nome de todas as pessoas que tem conta ou empréstimo em uma determinada agência bancária. Aplicando os conceitos anteriores tem-se:

Agência

CodAgencia	NomAgencia	CidadeSgencia
1	Morumbi	São Paulo
2	Zona Leste	ABC
3	Taboão da Serra	São Paulo
4	Zona Sul	ABC
5	Zona Oeste	Osasco
6	Zona Norte	São Carlos

Cliente

CodCliente	NomCliente	RuaCliente	CidadeCliente
1	Pedro Seródio	Rua A	São Paulo
2	Ademir Sampaio	Rua B	ABC
3	Jaqueline Albuquerque	Rua C	São Paulo
4	Marta de Oliveira	Rua B	ABC
5	Maria de Souza	Rua D	Osasco
6	Marta Nunes	Rua E	São Carlos

Conta Corrente

CodAgencia	NomConta	CodCliente	Saldo
1	246589	1	R$ 1.200,00
3	328941	1	R$ 845,00
4	759800	3	R$ 512,00
2	887744	2	R$ 3.000,00
3	446732	4	R$ 1.000,00
2	467766	3	R$ 9.621,00
5	774411	5	R$ 2.000,00
6	468952	6	R$ 2.244,00

Emprestimo

CodAgencia	CodCliente	Num Emp.	Valor
1	1	902230	R$ 500,00
3	1	902231	R$ 1.500,00
4	2	902240	R$ 1.200,00
2	3	902289	R$ 3.000,00
3	1	902233	R$ 850,00
1	3	902299	R$ 700,00
5	3	902212	R$ 400,00
6	6	902244	R$ 880,00
4	3	902245	R$ 1.000,00

Figura 5. Estrutura de tabelas Bancário

Segue: Selecionar os Clientes que têm conta bancária;

ΠNomCliente (δ CodAgencia = “3” ^ContaCorrente = Cliente.CodCliente
(CLIENTE X CONTA)

Selecionar os clientes que têm empréstimo;

ΠNomCliente (δ CodAgencia = “3” ^ContaCorrente = Cliente.CodCliente
(EMPRÉSTIMO X CLIENTE)

Resultado União da agência igual código 3, ficou assim:

Seleção em Cliente x Conta							
CodCliente	**NomCliente**	**RuaCliente**	**CidadeCliente**	**CodAgencia**	**NomConta**	**CodCliente**	**Saldo**
1	Pedro Seródio	Rua A	São Paulo	3	328941	1	R$ 845,00
4	Marta de Oliveira	Rua B	ABC	3	446732	4	R$ 1.000,00

Tabela 6. Seleção Conta x Cliente

Solução do problema: Projeção

NomCliente
Pedro Seródio
Marta de Oliveira

Tabela 7. Resultado de Projeção Cliente x Conta

Agora para a Seleção dos empréstimos da agência com código= 3.

Seleção em Cliente x Empréstimo							
CodCliente	**NomCliente**	**RuaCliente**	**CidadeCliente**	**CodAgencia**	**CodCliente**	**NumEmpest.**	**Saldo**
1	Pedro Seródio	Rua A	São Paulo	3	1	902231	R$ 1500,00
1	Pedro Seródio	Rua A	São Paulo	3	1	902233	R$ 850,00

Tabela 8. Seleção Cliente x Empréstimo

Projetando, tem-se:

NomCliente
Pedro Seródio
Pedro Seródio

Tabela 9. Projeção de resultado Cliente x Empréstimo

Por acaso os resultados foram idênticos.

Para responder à consulta desejada (o nome de todas as pessoas que possuem CONTA OU EMPRÉSTIMO a uma determinada agência 3), realiza-se, então, a união dos dois resultados:

ΠNomCliente (δ CodAgencia = “3” ^ContaCorrente = Cliente.CodCliente
(CLIENTE x CONTA)
U
ΠNomCliente (δ CodAgencia = “3” ^ContaCorrente = Cliente.CodCliente
(EMPRÉSTIMO X CLIENTE)

O Resultado final será:

NomCliente
MARTA DE OLIVEIRA
PEDRO SERÓDIO

Tabela 10. Projeção da União Conta x Empréstimo

OBS: Para que a união seja feita é preciso que as tabelas estejam com os atributos na mesma ordem. Exemplo:

Cliente					Cliente			
CodCliente	**NomCliente**	**RuaCliente**	**CidadeCliente**		**CodCliente**	**RuaCliente**	**NomCliente**	**CidadeCliente**
1	Pedro Seródio	Rua A	São Paulo		1	Rua A	Pedro Seródio	São Paulo
2	Ademir Sampaio	Rua B	ABC		2	Rua B	Ademir Sampaio	ABC
3	Jaqueline Albuquerque	Rua C	São Paulo		3	Rua C	Jaqueline Albuquerque	São Paulo
4	Marta de Oliveira	Rua B	ABC		4	Rua B	Marta de Oliveira	ABC
5	Maria de Souza	Rua D	Osasco		5	Rua D	Maria de Souza	Osasco
6	Marta Nunes	Rua E	São Carlos		6	Rua E	Marta Nunes	São Carlos

Tabela 11. Demonstração de Permissão para Junção de Tabelas

Os atributos não estão na mesma ordem, portanto, não é válida para fazer a união entre as tabelas.

3.1.4 Operação de Diferença (Binária)

Esta operação permite encontrar linhas que estão em uma tabela mas não estão na outra. Representação:

Tabela 1 - Tabela 2

Utilizando o mesmo exemplo anterior, entretanto o intuito é para encontrar todos os clientes que possuam conta, mas que não tenham um empréstimo na agência 3.

ΠNomCliente (δ CodAgencia = “3” ^ContaCorrente = Cliente.CodCliente
(CLIENTE x CONTA)

–

ΠNomCliente (δ CodAgencia = “3” ^ContaCorrente = Cliente.CodCliente
(EMPRÉSTIMO X CLIENTE)

O primeiro resultado será: Os que possuem empréstimo na agência 3.

NomCliente
Pedro Seródio
Pedro Seródio

Tabela 12. Selecão Cliente x Conta

Em seguida, os que possuem conta nesta mesma agência.

NomCliente
MARTA DE OLIVEIRA
PEDRO SERÓDIO

Tabela 13. Rename e Seleção da Agência

Resultado Final, apenas saber quem possui conta, mas não possui empréstimo.

NomCliente
Marta de Oliveira

Tabela 14. Rename e Projeção Empréstimo

Obs: A ordem nesta operação afeta o resultado final.

Tabela 1 **-** Tabela 2

Não é a mesma coisa que dizer que:

Tabela 2 **-** Tabela 1

3.1.5 Operação de Interseção de Tabela

Esta operação traz como resultado uma terceira, verificando outras duas tabelas comparando sem repetições, lembrando que as tabelas selecionadas para essa operação devem ser compatíveis. Representação:

Tabela 1 ∩ Tabela 2

Então:

=(Tabela 1 ∩ Tabela 2) – ((Tabela 1 – Tabela 2) **U (Tabela 2 – Tabela 1))**

Para demonstrar a intersecção das tabelas a fim de exercitar também os operadores anteriores, utilizará então a tabela de conta corrente, empréstimos, clientes e agência. A intersecção é o operador lógico *and*.

Começar por sua vez, determinar quem possui conta na agência especificada, no

caso a agência com código 3.

ComConta ← Π NOMCLIENTE (δ CodAgencia = “3”
^ContaCorrente.CodCliente= Cliente.CodCliente (Cliente x Conta))

O próximo passo deve mostrar quem possui empréstimo nesta agência.

ComEmprestimo ← Π NOMCLIENTE (δ CodAgencia = “3”
^Emprestio.CodCliente= Cliente.CodCliente (Emprestimo x Cliente))

Usa-se a intersecção dos dois resultados com o operador para assinalar, renomear os resultados anteriores.

Π NOMCLIENTE (ComConta ∩ ComEmprestimo)

Observe as duas tabelas:

Seleção em Cliente x Conta							
CodCliente	**NomCliente**	**RuaCliente**	**CidadeCliente**	**CodAgencia**	**NomConta**	**CodCliente**	**Saldo**
1	Pedro Seródio	Rua A	São Paulo	3	328941	1	R$ 845,00
4	Marta de Oliveira	Rua B	ABC	3	446732	4	R$ 1.000,00

Tabela 15- Selecão Cliente x Conta

Resultado Solicitado: NomCliente

NomCliente
Pedro Seródio
Marta de Oliveira

Tabela 16. Resultado Selecão Cliente

Agora projetar os clientes com empréstimo:

Seleção em Cliente x Empréstimo							
CodCliente	**NomCliente**	**RuaCliente**	**CidadeCliente**	**CodAgencia**	**CodCliente**	**NumEmpest.**	**Saldo**
1	Pedro Seródio	Rua A	São Paulo	3	1	902231	R$ 1500,00
1	Pedro Seródio	Rua A	São Paulo	3	1	902233	R$ 850,00

Tabela 17. Selecão Cliente x Empréstimo

Resultado 2:

NomCliente
Pedro Seródio
Pedro Seródio

Tabela 18. Resultado Selecão Cliente

Logo resultado final é:

NomCliente
Pedro Seródio

Tabela 19. Resultado Final Linha Cliente

3.1.6 Operação de Junção

Esta é uma operação que realiza a combinação das linhas de uma tabela com as linhas correspondentes de outra tabela. Será utilizado o produto cartesiano para a seleção das mesmas. É um dos operadores mais utilizados, lembrando que os relacionamentos das colunas no modelo de dados geralmente são chaves estrangeiras, o que facilita muito a escrita das expressões.

Tabela A |X| A.chave1=B.chave2 Tabela B

Em que A.chave1 = B.chave2 é o predicado da seleção, ou seja, uma seleção do produto cartesiano das duas tabelas:

Δ A.chave1 = B.chave2(AxB)

Todas as linhas que estiverem na primeira e na segunda coluna sem exceção, estarão na terceira coluna, que será resultado de um produto cartesiano. Existe ainda uma tipologia de junção que também pode ser chamada de junção natural, que mostra um mesmo resultado só que os valores que se repetem, aparece uma única vez na tabela resultante. Utilizando ainda o mesmo exemplo, agora preciso

saber o nome de todos os clientes que tem empréstimos e a cidade em que vivem. Primeiro passo:

Cliente |X| Cliente.CodCliente = Empréstimo.CodCliente Empréstimo

Como ambas possuem uma coluna em comum, então o melhor a fazer é a junção natural, segue:

CodCliente	NomCliente	RuaCliente	CidadeCliente	CodAgencia	CodCliente	NumEmp.	Valor
1	Pedro Seródio	Rua A	São Paulo	1	1	902230	R$ 500,00
2	Ademir Sampaio	Rua B	ABC	2	1	902231	R$ 1.500,00
3	Jaqueline Albuquerque	Rua C	São Paulo	3	2	902240	R$ 1.200,00
4	Marta de Oliveira	Rua B	ABC	4	3	902289	R$ 3.000,00
5	Maria de Souza	Rua D	Osasco	5	1	902233	R$ 850,00
6	Marta Nunes	Rua E	São Carlos	6	3	902299	R$ 700,00
3	Jaqueline Albuq.	Rua C	São Paulo	3	3	902212	R$ 400,00
6	Marta Nunes	Rua E	São Carlos	6	6	902244	R$ 880,00
3	Jaqueline Albuq.	Rua C	São Paulo	3	3	902245	R$ 1.000,00

Tabela 20. Junção Cliente x Empréstimo

Então projeção das colunas desejadas:

Π NOMCLIENTE, CIDADECLIENTE (Resultado)

Consolidando as duas operações em uma mesma operação:

Π NOMCLIENTE, CIDADECLIENTE (Cliente|X|
CLIENTE.CODCLIENTE=EMPRESTIMO.CODCLIENTE Empréstimo)

Ou ainda com a junção natural:

Π NOMCLIENTE, CIDADECLIENTE (Cliente|X| Empréstimo)

Mais um exemplo: Todos os clientes que tem empréstimos e moram em São Paulo.

Π NOMCLIENTE, CIDADECLIENTE (δ CidadeCliente=”São Paulo” (Cliente|X|
Empréstimo))

A seleção dessa tabela retorna:

CodCliente	NomCliente	RuaCliente	CidadeCliente	CodAgencia	CodCliente	NumEmp.	Valor
1	Pedro Seródio	Rua A	São Paulo	1	1	902230	R$ 500,00
3	Jaqueline Albuquerque	Rua C	São Paulo	3	2	902240	R$ 1.200,00

Tabela 21- Resultado da Junção

Projeção do resultado final:

NomCliente
Pedro Seródio
Jaqueline Albuquerque

Tabela 22- Resultado Final Coluna Cliente

3.1.7 Operação de Divisão

Esta operação representada pelo símbolo "/", serve para consultas como as do tipo para todos. Exemplo encontrar todos os clientes que têm conta em todas as agências de São Paulo.

Primeiro passo: Obter todas as agências que retornam o resultado como São Paulo:

AgenciaSaoPaulo ← ΠXCODAGENCIA(δ cidadeagencia="São Paulo" (Agencia)

Obtenho:

CodAgencia
1
3

Tabela 23. Resultado Seleção Agência São Paulo

Agora obter todos os pares NomCliente, CodAgencia referentes a um cliente possui uma conta em uma agência realizando a junção entre conta e cliente:

ClienteConta ← Π NomCliente, CodAgencia (Conta x Cliente)

NomCliente	CodAgencia
Pedro Seródio	1
Pedro Seródio	3
Jaqueline Albuquerque	4
Ademir Sampaio	2
Marta de Oliveira	3
Jaqueline Albuquerque	2
Maria de Souza	5
Marta Nunes	6

Tabela 24. Resultado Seleção Agência x Cliente

Em seguida preciso encontrar clientes que apareçam em ClienteConta com cada nome de agência em São Paulo.

Π NomCliente⟵ , CodAgencia (Conta x Cliente) /
Π CodAgencia (δ CidadeAgencia = “ São Paulo” (Agencia))

São procurados os conjuntos de linhas em ClienteConta cujos valores dos atributos comuns são iguais a todos os que aparecem em São Paulo.

NomCliente	CodAgencia
Pedro Seródio	1
Pedro Seródio	3
Marta de Oliveira	3

Tabela 25. Resultado Projeção Cliente x Agência São Paulo

Resultado Final será:

NomCliente
Pedro Seródio

Tabela 26. Resultado Projeção Coluna Cliente

Pois, apenas Pedro Serôdia, possui conta em mais de uma agência em São Paulo.

3.2 A LINGUAGEM SQL

SQL (Structured Query Language – Linguagem Estruturada de Pesquisa) teve seus fundamentos no modelo relacional de CODD (1970). Em 1975, foi implementado um protótipo de aplicação dessa nova linguagem. Entre 1976 e 1977, o Sequel foi revisado e ampliado, e teve seu nome alterado para SQL por razões jurídicas. Uma das razões de popularidade dos sistemas relacionais é a sua facilidade de manipulação e entendimento. A linguagem SQL foi desenvolvida especialmente para o ambiente relacional, podendo ser adaptada a qualquer ambiente não relacional.

Foi padronizada pela ANSI (American National Standards Institute, ou Instituto de Padrões Nacionais Americanos) e o mesmo foi aprovada pela ISO (International Standards Institute). A padronização apresenta uma grande vantagem podendo ser utilizada por todos os SGBDs e também por uma série de hardware e software diferentes, ela não implica diretamente na portabilidade das aplicações, porque esta linguagem é apenas para definição e manipulação de banco de dados, não esta preocupada com a estrutura de banco de dados que está sendo adquirido. Como SQL não é uma linguagem de programação completa, as aplicações baseadas nela utilizam outra linguagem de programação, como por exemplo COBOL.

3.2.1. Vantagens e Desvantagens da SQL

Uma das vantagens da padronização da SQL é que ela pode ser utilizada em praticamente todos SGBDs Relacionais, portanto não precisa se preocupar quando for migrar para outro SGBD. Mas por outro lado, a padronização traz para quem desenvolve aplicações soluções padronizadas, não podendo sofrer melhorias ou alterações.

Pode ser utilizado tanto em um computador pessoal quanto em um mainframe ou ainda computadores de trabalho de médio porte. Fácil entendimento do inglês, provê acesso rápido aos dados, redução de custos, já que é só treinar o pessoal para movimentar um ambiente para outro.

Porém a SQL esta longe de ser uma linguagem relacional ideal. Segundo Date em seu livro Relational Database: Selected Writing (Addison – Wesley, 1986), algumas críticas são feitas à linguagem SQL:

- Falta de ortogonalidade nas expressões, funções embutidas, variáveis indicadoras, referência a dados correntes, constante NULL, conjuntos vazios etc;
- Definição formal da linguagem após sua criação;
- Discordância com as linguagens hospedeiras;
- Falta de algumas funções;
- Erros (valores nulos, índices únicos, cláusula From etc);
- Não dá suporte a alguns aspectos do modelo relacional (atribuição de relação, join explícito, domínios, etc).

A SQL é composta de subconjuntos de comandos para executar diferentes tarefas, pode-se citar:

- Uma linguagem de definição de dados (DDL).
- Uma linguagem de manipulação de dados (DML).
- Uma linguagem de segurança de dados (DCL).

A parte **DDL** permite criar e modificar e excluir a estrutura de uma tabela e seus índices; seus principais comandos são:

- CREATE: Cria tabelas, campos e índices num banco de dados.
- DROP: Remove tabelas e índices de um banco de dados.
- ALTER: Altera a estrutura de uma tabela de um banco de dados.

A parte **DML** permite manipular os dados (Inserir, Excluir e Atualizar) bem como executar consultas através da recuperação de subconjuntos de dados para posterior tratamento e seus principais comandos são:

- SELECT: Seleciona um conjunto de registros de uma ou mais tabelas usando um critério específico.
- INSERT: Adiciona dados a uma tabela.
- UPDATE: Atualiza os dados de uma tabela segundo critérios específicos.

- DELETE: Remove registros de uma tabela.

A **DCL** permite a implementação da segurança interna do Banco de Dados. Seus comandos principais são **GRANT** e **REVOKE.**

Mesmo enfrentando alguns problemas e críticas, a linguagem SQL é a mais utilizada atualmente e ficará por muitos anos como preferência de todos. Todo percurso pela linguagem é descrito anteriormente por um modelo de dados.

3.2.2 Definição Básica na SQL

A estrutura básica de uma expressão SQL consiste em três cláusulas: SELECT, WHERE e FROM.
A cláusula SELECT é evidentemente, uma relação. As linguagens de consulta formal são baseadas em noção matemática de uma relação sendo um conjunto. Segue alguns Exemplos:

```
Select nome_agencia
From empréstimo // aqui lista todas as agencias na relação empréstimo.
```

```
Select distinct nome_agencia
From empréstimo // lista o nome_agencia para cada tupla em que aparece a relação empréstimo.
```

```
Select all nome_agencia
From empréstimos // as duplicatas não devem ser removidas.
```

3.2.3 A cláusula Where

A SQL usa os conectivos AND, OR e NOT, em vez dos símbolos matemáticos, ^,v e, na clausula WHERE. Os operandos dos conectivos lógicos podem ser expressões envolvendo, os operadores de comparação <,<=,>,>=,=e <>. A SQL permite usar os operadores de comparação para comparar strings e expressões aritméticas além de tipos especiais, como tipo de data. O operador BETWEEN simplifica a clausula WHERE especificando um valor menor ou igual a algum valor ou igual a outro valor. Considere o mesmo exemplo:

```
Select  numero_empr
From empréstimo
Where nome_agencia = 'Perridge' and quantia > 1200 // encontrando todos os números de empréstimos com este valor.
Select numero_empr
From empréstimo
Where quantia between 10.000 and 20.000 em vez de
```

```
Select numero_empr
From empréstimo
Where quantia between 10.000 and 20.000
```

Em vez de

```
Select numero_empr
From empréstimo
Where quantia<=20.000 and amount>=10.000
```

3.2.4 A cláusula From

A cláusula FROM isolada define um produto cartesiano das relações das tabelas. Ela é exigida em todas as instruções SELECT em que estão sendo recuperados dados de tabelas ou exibições. Esta consulta relata todos os clientes que têm um empréstimo do banco.

Nome_cliente, numero_empréstimo, conta (tomador<>empréstimo)

Para que a consulta seja feita ainda mais completa utiliza-se:

```
Select nome_cliente, tomador.numero_emprestimo, quantia
From tomador, empréstimo
Where tomador.numero_empréstimo= empréstimo.número_empréstimo
```

Note que a SQL usa a notação nome-relação, nome-atributo, como álgebra relacional, para evitar ambiguidade nos casos em que um atributo aparece no esquema mais de uma relação.

As instruções SELECT que não exigem uma cláusula FROM são aquelas que não estão selecionando dados de uma tabela no banco de dados. Essas instruções

SELECT só selecionam dados de variáveis locais ou funções Transact-SQL que não funcionam em uma coluna, por exemplo:

```
SELECT @MyIntVariable
SELECT @@VERSION
SELECT DB_ID('AdventureWorks2008R2')
```

3.2.5 Variáveis de tupla

Uma variável na SQL precisa estar associada a uma determinada relação. Todas as variáveis de uma tupla são definidas pela cláusula FROM por meio da cláusula “as”.

```
Select nome_cliente, T.número_emprestimo, S.quantia
From tomador as T,empréstimo as S
Where T.numero_empréstimo = S.numero_emprestimo
```

Quando são definidas expressões de forma *nome-relação.nome-atributo*, o nome da relação é, na verdade, uma variável de uma tupla definida implicitamente. Esta variável é mais utilizada para comparar duas tuplas na mesma relação.

3.2.6 Ordenação da exibição de Tuplas

A SQL ainda oferece um controle para exibição da tuplas sobre a sua ordem de acordo com cada necessidade. A cláusula ORDER BY faz com que as tuplas no resultado de uma consulta apareçam na ordem classificada. Como padrão a lista ORDER BY lista itens na ordem crescente usamos então ASC e para ordem decrescente DESC.

```
Select *
From empréstimo
Order by quantia desc, número_empréstimo asc.
```

3.2.7 Operações de String

A operação mais utilizada em strings é a correspondência de padrões usando o operador like.

Porcentagem (%) o caracter % e corresponde a qualquer substring.

Sublinhado (_), o caracter _ corresponde a qualquer caractere.

Os padrões também fazem distinção entre maiúsculo e minúsculo.

'Perry%' localiza qualquer string começando com "Perry".

'%idge' localiza qualquer string contendo "idge" como uma substring.

_ _ _ localiza qualquer string com exatamente três caracteres.

_ _ _% localiza qualquer string com pelo menos três caracteres.

Segue exemplo:

```
Select nome_cliente
From cliente
Where rua_cliente like '%Main%'
```

- **Operações em Conjunto**

Uma expressão SELECT-FROM-WHERE aninhada dentro de outra consulta é considerada uma subconsulta.

Aplicações:

- **Membros de Conjuntos**

Verificar se uma tupla é membro ou não de uma relação. O conectivo **IN** testa os membros de um conjunto, no qual este conjunto é a coleção de valores produzidos na cláusula SELECT.

```
select A1, A2, ..., An
from r1, r2, ..., rn
where P in (
select A1, A2, ..., An
from r1, r2, ..., rn
where P
);
```

- **Comparações de Conjuntos**

Permite usar comparações do tipo > some (maior que ao menos uma), <= some, = some, etc...
ou > all (maior que todas), >= all, etc.

```
select A1, A2, ..., An
from r1, r2, ..., rn
where P > all (
select A1, A2, ..., An
from r1, r2, ..., rn
where P
);
```

3.2.8 Cláusula Group By

Em algumas situações aplica-se a função agregada não apenas ao conjunto de tuplas, mas também a um grupo de conjuntos de tuplas. O atributo ou atributos informados pela cláusula GROUP BY são usados para formar grupos. Exemplo:

```
Select nome_agência.avg (saldo)
From conta
Group by nome_agência
```

Esta consulta ordenou em uma tupla o saldo médio das contas dessa agência.

3.2.9 Definição de VIEW

A VIEW simplesmente nos possibilita visualizar os dados. Pode ser considerada uma tabela virtual ou uma consulta armazenada, é aconselhável uma VIEW ser implementada encapsulando uma instrução SELECT, porque toda consulta feita no banco é armazenada em uma memória cachê e para que o banco seja mais rápido criamos a VIEW, em vez de ter que retrabalhar uma instrução.

O processo de manter a VIEW é chamado de manutenção de VIEW, assim as consultas precisam ser pesadas com os custos de armazenamento e o overhead adicional das atualizações. A expansão de VIEW é uma forma de definir o significado das VIEWS em termos de outras VIEWS. O procedimento considera que as definições de VIEW não são recursivas, ou seja, nenhuma VIEW é usada em sua

própria definição, quer diretamente ou indiretamente por outras definições VIEW. Desde que as definições do loop não sejam recursivas esse loop terminará. Veja a seguir como ficará o código:

```
Select *
From (selec nome_cliente
From ((select nome_agência, nome_cliente
From depositante, conta
Where depositante número_conta=conta.numero_conta
Union
(select nome_agência, nome_cliente
From tomador, empréstimo
Where tomador.numero_empréstimo=empréstimo.
Numero_empréstimo))
Where nome_agência = 'Perryridge'
Where nome_cliente = 'John'
```

Nesse momento, não existem mais usos de relações de view, e a expansão termina.

3.2.10 Modificação no Banco de Dados

Uma modificação no banco acontece quando é inserido, excluído ou alterado um determinado dado de qualquer tabela. Não é possível excluir apenas o valor do atributo, o correto é excluir uma tupla inteira. Para inserir algo no banco de dados é necessário especificar uma tupla a ser inserida ou escrever a consulta cujo resultado seja um conjunto de tuplas a ser inserido, obviamente que os tipos que constam nas tuplas devem ser iguais. Já para fazer uma UPDATE (alteração) também podem ser atualizadas por meio de consultas especificas para a tupla que se pretende alterar. Assim tem-se:

```
Delete from conta where saldo < ( select avg (saldo) from conta);
Insert into conta values ('A-9732','Perryride',1200);
Update conta set saldo=saldo*1.05;
```

3.2.11 Transações

Uma transação consiste em uma seqüência de instruções de consulta e\ou atualizações. O padrão SQL especifica que uma transação inicia implicitamente quando uma instrução SQL é executada.

3.2.12 Funções Agregadas

Funções agregadas são aquelas que tomam uma coleção (um conjunto ou multiconjunto) de valores como entrada e retornam um único valor. A SQL oferece funções embutidas como:

- Average: avg
- Minimum: min
- Maximum: sum
- Count: count

Somente a função SUM e AVG precisam ser uma coleção de números, os outros operadores podem operar tanto em coleções numéricas quanto strings.

```
Select avg (saldo)
From conta
Where nome_agência = ‘Perryridge’
```

O resultado dessa consulta retorna apenas um valor, pois a consulta foi feita em apenas uma tupla. Existem casos que precisam eliminar algumas duplicatas antes de calcular uma função agregada. Exemplo: Encontrar o numero de depositantes de agrupados por cada agência, aparecendo uma única vez nos resultados, ficará assim a consulta:

```
Select nome_agencia, count (distinct nome_cliente)
From depositante, conta
Where depositante.numero_conta = conta.numero_conta
Group by nome_agencia;
```

Agora interessa apenas as agências que o saldo médio seja maior do que R$ 1200,00. Essa condição aplica-se a uma única tupla, mas sim em cada grupo formado pela cláusula GROUP BY, sendo assim deverá ser utilizado a cláusula HAVING que após os grupos terem sidos formados, permite funções agregadas ficando assim:

```
Select nome_agencia.avg (saldo)
From conta
Group by nome_agencia
Having avg (saldo) > 1200
```

Para contar o número de tuplas em uma relação a notação é COUNT

```
Select count * from cliente
```

A SQL não permite utilizar DISTINCT com COUNT, é válido usar o distinct com o MAX ou MIN, mesmo se o resultado não mudar. Pode ainda substituir a palavra-chave ALL no lugar do distinct.

3.2.13 Where x Having

Encontrar o saldo médio para cada cliente que mora em Harrison e que tem pelo menos três contas. Segue:

```
Select depositante.nome_cliente, avg (saldo)
From depositante, conta, cliente
Where depositante.numero_conta = conta.numero_conta and
Depositante.nome_cliente = cliente.nome_cliente and
Cidade_cliente = 'Harrison'
Group by depositante.nome_cliente
Having count (distinct depositante.numero_conta)>=3
```

Se a cláusula WHERE aparece junto com a cláusula HAVING, então a SQL executa os grupos do WHERE primeiro, sendo que os grupos que satisfazem o predicado são em grupos pela cláusula GROUP BY. Em seguida aplica a cláusula HAVING, se presente, a cada grupo, ela remove os grupos que não satisfazem o predicado, e faz um SELECT trazendo os grupos restantes para gerar tuplas do resultado da consulta.

3.2.14 DICAS DO USO DA SQL

Para que se obtenha a maior integração dos dados e não tenha nenhum registro em duplicidade deve-se:

- **Utilizar a clausula WHERE:** Limita o número de linhas retornadas pela consulta, já que será retornado um número menor de dados, a condição mais restritiva deve ficar por último na cláusula WHERE.
- **Evitar grandes operações de classificação:** Como por exemplo, operações de classificação envolvendo **ORDER BY, GROUP BY e HAVING**. Subconjuntos de dados são armazenados na memória ou em disco sempre que operações de classificação são realizadas.Utilize apenas se necessário.
- **Evitar usar operadores como NOT, <>, NOT EXISTS, NOT IN, NOT LIKE, IS NULL ou LIKE '%R':** Nesses casos um índice não é útil porque a pesquisa não pode ser limitada, assim, toda linha dever ser avaliada.
- **Produto cartesiano X Join:** O desempenho do Join é maior do que o do produto cartesiano.
- **Evitar o uso do DISTINCT, IN, NOT IN:** Provoca overhead adicional na consulta degradando o desempenho da mesma. Utilize a clausula **EXISTS OU NOT EXIST** no lugar desses operadores**.**
- **Usar índice clustered:** Em consultas delimitadas por intervalo de valores

3.3 OTIMIZADOR DE CONSULTA

Uma vez que a consulta for mal elaborada e executada pode degradar o desempenho de todo o sistema. E cada vez mais precisamos da parte humana atuando na análise de performance no desenvolvimento de aplicações e/ou administração de dados. Ao trabalhar com consultas em bancos de dados deve-se ter uma atenção maior com relação à sua eficiência e uma base de conhecimento para que seja possível criar consultas mais inteligentes, refinadas, objetivando o ganho de performance.

3.3.1 Tipos de Otimizações

Há dois tipos de otimizações, as baseadas em regras (RBO) e as baseadas em tempo (CBO), conforme apresentado a seguir.

3.3.1.1 Rule Based Optimizer – RBO

Otimização baseada em regras é um método que seleciona caminhos de acesso em um rank dado a vários caminhos de acesso, um rank baixo é um acesso melhor do que um rank alto. Ela é processada em dois passos:

- Determinar quais são os possíveis caminhos de acesso;

- Selecionar o caminho de acesso com o rank menor.

Ela não realiza estatísticas sobre os dados, não calcula custos, é dirigida a sintaxe e é suportada por compatibilidade.

Este rank é demonstrado na tabela a seguir:

RANK	CAMINHO DE ACESSO
1	Única linha (ROWID)
2	Única linha (JOIN)
3	Única Linha (hash cluster keu com chama única/primária
4	Única linha (chave primária/única)
5	Cluster Join
6	Chave hash cluster
7	Chave Indexed cluster
8	Índice composto
9	Índice coluna única
10	Bounded range search on indexed columns
11	Unbounded range search on indexed columns
12	Sort-Merge Join
13	Índices de coluna MAX e MIN
14	ORDER BY em colunas índices
15	Leitura completa da tabela

Tabela 27. Rank dos operadores/cláusulas – Inspirada em (GULU, 2002).

3.3.1.2 Cost Based Optimizer – CBO

A Otimização baseia sua escolha em estatísticas armazenadas sobre as tabelas. Quando ela é baseada em tempo é por meio da medição do tempo que cada caminho de acesso demora em efetuar a tarefa. Este tempo é calculado pelo número de leituras lógicas efetuadas, utilização do processador e transmissões na rede. Então fica assim:

-Determina os caminhos de acesso disponíveis;
-Calcula o “tempo” de cada um deles;
-Seleciona o caminho de acesso com o menor tempo.

3.2.2 Entendendo o Plano de Execução

Quando se executa uma operação no banco de dados, seja ela um SELECT, INSERT, ou outra qualquer, o PostgreSQL, assim como outros SGBDs, possui um mecanismo interno chamado planejador (ou otimizador), que reescreve a consulta com a intenção de aperfeiçoar os resultados, gerando um Plano de Execução.

O Plano de Execução ou de Consulta é uma sequência de passos que serão executados pelo SGBD para executar uma consulta, ou seja, quais os tipos de processamento que serão feitos diretamente nos registros ou em estruturas de índices, bem como informações como o tempo de entrada, o tempo de resposta e o total de registros percorridos. O planejador precisa então fazer uso de estatísticas como o número total de registros da tabela, o número de blocos de disco ocupados por cada tabela e se há a presença de índices ou não.

3.4 RECUPERAÇÃO DE DADOS DE VÁRIAS TABELAS (JOINS)

3.4.1 Conceito

Os Joins ou junções são diferentes formas de acessar várias tabelas ao mesmo tempo, registros, testando cruzando informações, combinando desigualdades, para se obter um determinado resultado, são eles: INNER JOIN, LEFT JOIN, RIGTH

OUTER JOIN, FULL OUTER JOIN e CROSS JOIN. Para exemplificar esses conceitos será utilizado o Modelo de Dados de (Machado, 2008).

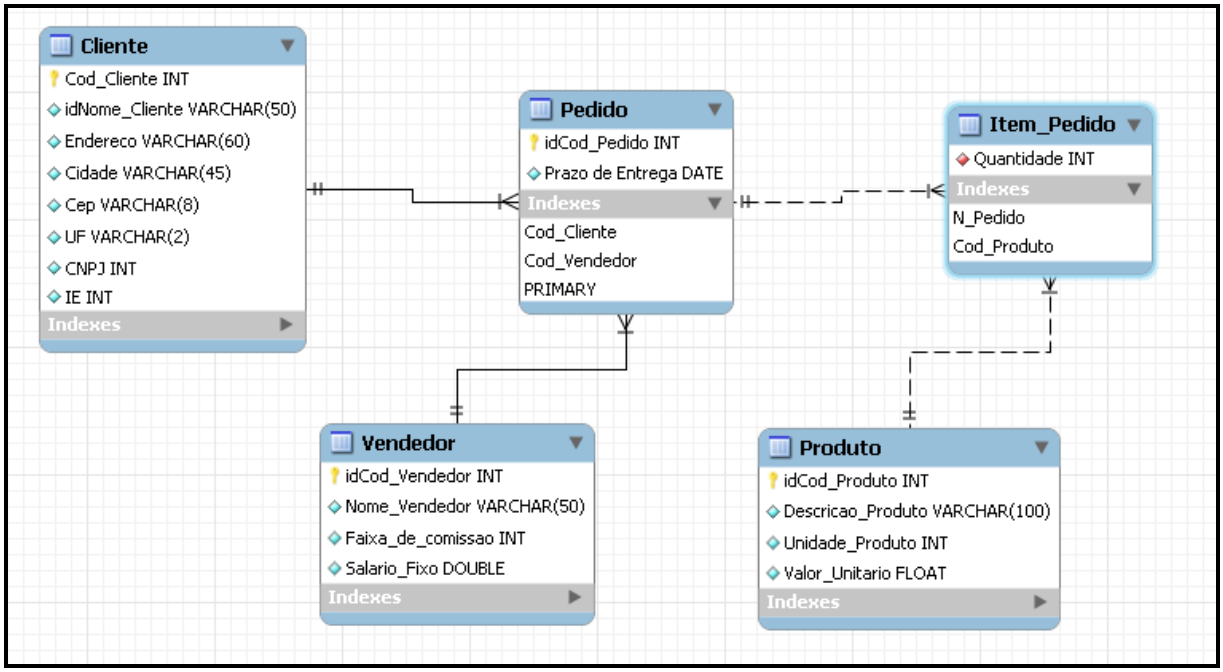


Figura 6. Modelo Físico de Tabelas Vendas– Implantado Workbenk 5.0

Código abaixo foi executado no Postgresql:

```
CREATE TABLE cliente
(
  codigo_cliente serial NOT NULL,
  nome_cliente character varying(50) NOT NULL,
  endereco character varying(100) NOT NULL,
  cidade character varying(50) NOT NULL,
  cep character varying(10),
  uf character varying(2) NOT NULL,
  cnpj character varying(14),
  ie character varying(20),
  CONSTRAINT pkcodigo_cliente PRIMARY KEY (codigo_cliente)
)
WITHOUT OIDS;
ALTER TABLE cliente OWNER TO postgres;
-- Index: "Index_Cep"
-- DROP INDEX "Index_Cep";
CREATE INDEX "Index_Cep"
ON cliente  USING btree  (cep);
-- Index: "Index_Cidade"
-- DROP INDEX "Index_Cidade";
CREATE INDEX "Index_Cidade"
ON cliente USING btree (cidade);

CREATE TABLE CLIENTE
```

```
REATE TABLE item_pedido
(
numero_pedido integer NOT NULL,
codigo_produto integer NOT NULL,
CONSTRAINT "Codigo_Produto" FOREIGN KEY (codigo_produto)
REFERENCES produto (codigo_produto) MATCH SIMPL
ON UPDATE NO ACTION ON DELETE NO ACTION
CONSTRAINT "Numero_Pedido" FOREIGN KEY (numero_pedido)
REFERENCES pedido (numero_pedido) MATCH SIMPLE
ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITHOUT OIDS;
-- Index: "fki_Codigo_Produto"
-- DROP INDEX "fki_Codigo_Produto";
CREATE INDEX "fki_Codigo_Produto"
ON item_pedido USING btree  (codigo_produto);
-- Index: "fki_Numero_Pedido"
-- DROP INDEX "fki_Numero_Pedido";
CREATE INDEX "fki_Numero_Pedido"
ON item_pedido USING btree  (numero_pedido);

CREATE TABLE pedido
(
 numero_pedido serial NOT NULL,
 prazo_entrega date,
 codigo_cliente integer,
 codigo_vendedor integer,
 CONSTRAINT pknumero_pedido PRIMARY KEY (numero_pedido),
 CONSTRAINT fkpedido_codigo_cliente FOREIGN KEY (codigo_cliente)
 REFERENCES cliente (codigo_cliente) MATCH SIMPLE
 ON UPDATE NO ACTION ON DELETE NO ACTION,
 CONSTRAINT fkpedido_codigo_vendedor FOREIGN KEY (codigo_vendedor)
 REFERENCES vendedor (codigo_vendedor) MATCH SIMPLE
 ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITHOUT OIDS;
ALTER TABLE pedido OWNER TO postgres;

CREATE TABLE produto
(
  codigo_produto serial NOT NULL,
  descricao_produto character varying(50) NOT NULL,
  unidade_produto character varying(10),
  valor_unitario numeric,
  CONSTRAINT pkcodigo_produto PRIMARY KEY (codigo_produto)
)
WITHOUT OIDS;
ALTER TABLE produto OWNER TO postgres;
```

```
CREATE TABLE vendedor
(
  codigo_vendedor serial NOT NULL,
  nome_vendedor character varying(50) NOT NULL,
  salario_fixo numeric,
  faixa_de_comissao numeric,
  CONSTRAINT pkcodigo_vendedor PRIMARY KEY (codigo_vendedor)
)
WITHOUT OIDS;
ALTER TABLE vendedor OWNER TO postgres;
-- Index: "Index_Salario"
-- DROP INDEX "Index_Salario";

CREATE INDEX "Index_Salario"
  ON vendedor  USING btree  (salario_fixo);

CREATE TABLE vendedor
(
  codigo_vendedor serial NOT NULL,
  nome_vendedor character varying(50) NOT NULL,
  salario_fixo numeric,
  faixa_de_comissao numeric,
  CONSTRAINT pkcodigo_vendedor PRIMARY KEY (codigo_vendedor)
)
WITHOUT OIDS;
ALTER TABLE vendedor OWNER TO postgres;
-- Index: "Index_Salario"
-- DROP INDEX "Index_Salario";
CREATE INDEX "Index_Salario"
  ON vendedor  USING btree  (salario_fixo);
```

3.4.2 INNER JOIN

A operação inner join é utilizada em duas ou mais tabelas, quando quero saber algo exato, a condição join restringe e qualifica a junção dos dados entre as tabelas. Por exemplo, se desejo saber por quem os pagamentos de minha empresa foram feitos e se houver algum departamento que não efetuou nenhum pagamento no resultado aquele que não efetuou nenhum aparecerá como null e não serão retornados, apenas constará os nomes de quem efetuou algum pagamento dentro do mês. Outro exemplo, agora utilizando as tabelas acima, serão transcritas para o Banco Postresql para demonstração dos resultados, desejo saber quais pedidos foram feitos e no caso que houver retorne o nome de cada cliente.

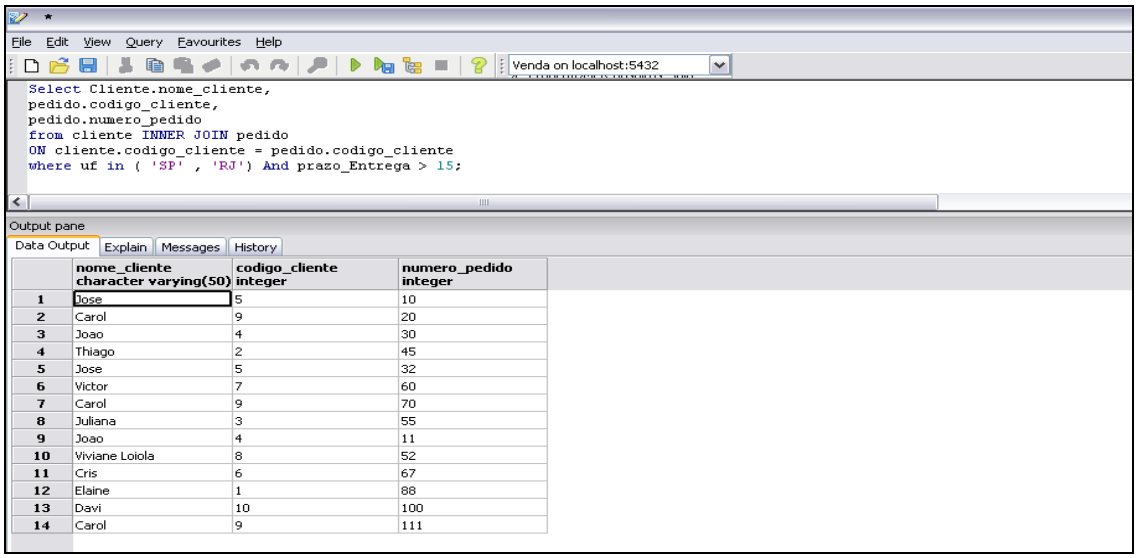
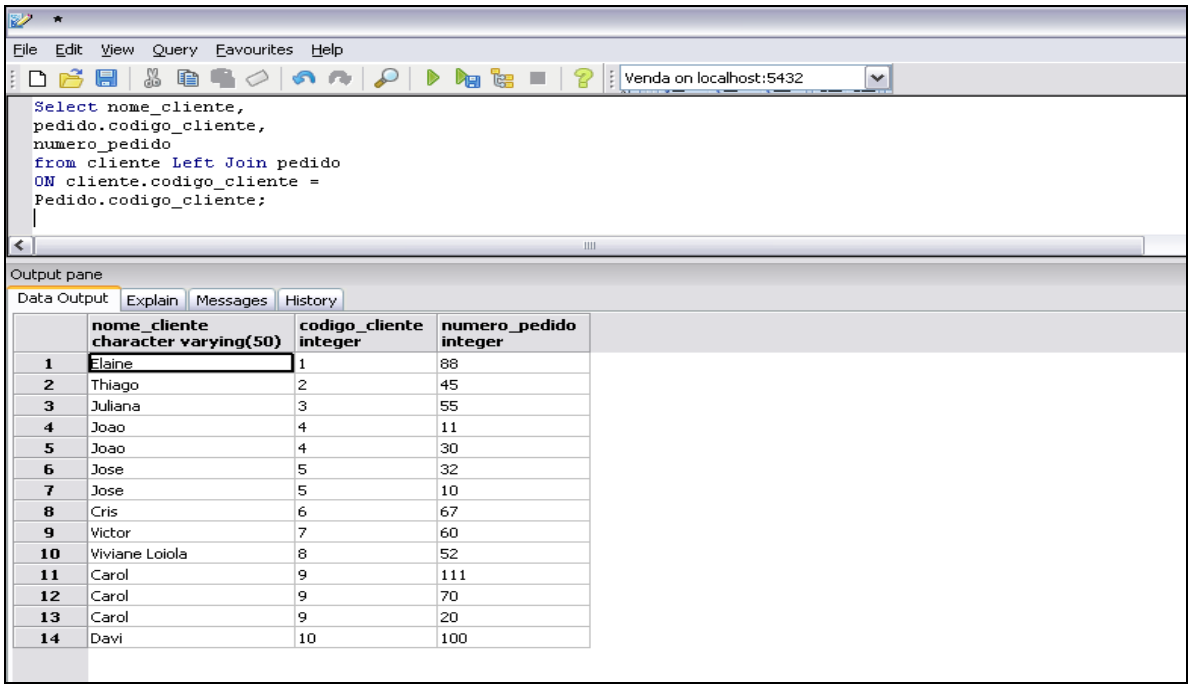


Figura 7. Resultado da Query Inner Join

3.4.3 Left Join

O Left Join diferencia do Inner Join porque no mesmo exemplo anterior os clientes da coluna mais a esquerda mesmo que na coluna mais a direita não conste nenhum item, neste caso mesmo aqueles clientes que não efetuaram nenhum pedido seria retornado como resultado. A consulta ficaria assim:

```
SELECT nome_cliente,
            pedido.codigo_cliente,
            numero_pedido,
FROM cliente LEFT JOIN pedido
ON cliente.codigo_cliente=
     Pedido.codigo_cliente
```



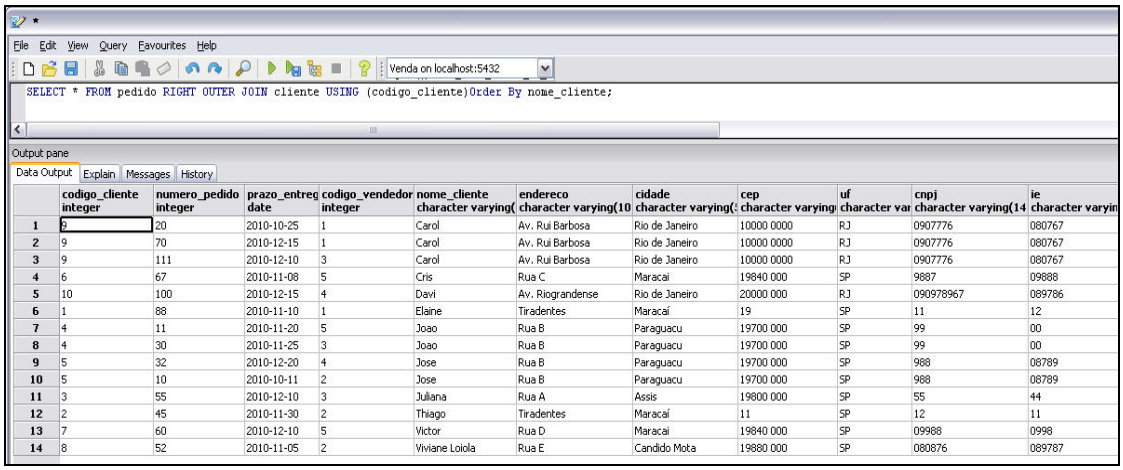
	nome_cliente character varying(50)	codigo_cliente integer	numero_pedido integer
1	Elaine	1	88
2	Thiago	2	45
3	Juliana	3	55
4	Joao	4	11
5	Joao	4	30
6	Jose	5	32
7	Jose	5	10
8	Cris	6	67
9	Victor	7	60
10	Viviane Loiola	8	52
11	Carol	9	111
12	Carol	9	70
13	Carol	9	20
14	Davi	10	100

Figura 8. Resultado da Query Left Join

3.4.4 RIGTH Outer Join

Esta operação somente pode ser executada entre duas tabelas.São incluídas todas as linhas da tabela do segundo nome da tabela da expressão (tabelas mais a direita da expressão). A única diferença entre o **LEFT OUTER JOIN** e o **RIGHT OUTER JOIN** é só a indicação de qual é o lado forte do relacionamento, ou seja, em qual lado da cláusula está a tabela que pode não ter valores na tabela filha. Não é utilizado com tanta freqüência pelos bancos de dados.

SELECT * FROM pedido RIGHT OUTER JOIN
cliente USING (codigo_cliente) Order By nome_cliente;



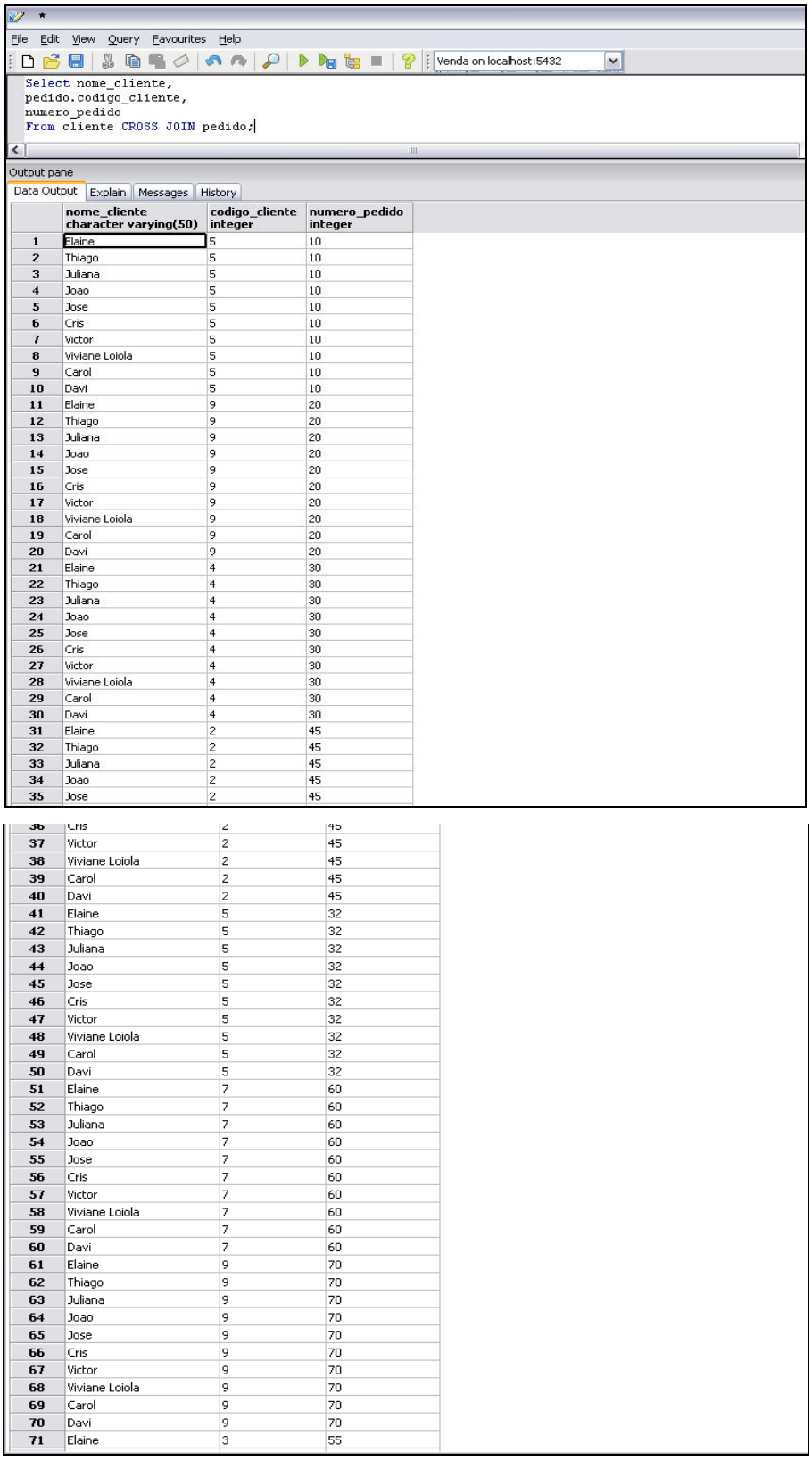
	codigo_cliente integer	numero_pedido integer	prazo_entreg date	codigo_vendedor integer	nome_cliente character varying(	endereco character varying(10	cidade character varying(	cep character varying	uf character var	cnpj character varying(14	ie character varyin
1	9	20	2010-10-25	1	Carol	Av. Rui Barbosa	Rio de Janeiro	10000 0000	RJ	0907776	080767
2	9	70	2010-12-15	1	Carol	Av. Rui Barbosa	Rio de Janeiro	10000 0000	RJ	0907776	080767
3	9	111	2010-12-10	3	Carol	Av. Rui Barbosa	Rio de Janeiro	10000 0000	RJ	0907776	080767
4	6	67	2010-11-08	5	Cris	Rua C	Maracai	19840 000	SP	9887	09888
5	10	100	2010-12-15	4	Davi	Av. Riograndense	Rio de Janeiro	20000 000	RJ	090978967	089786
6	1	88	2010-11-10	1	Elaine	Tiradentes	Maracaí	19	SP	11	12
7	4	11	2010-11-20	5	Joao	Rua B	Paraguacu	19700 000	SP	99	00
8	4	30	2010-11-25	3	Joao	Rua B	Paraguacu	19700 000	SP	99	00
9	5	32	2010-12-20	4	Jose	Rua B	Paraguacu	19700 000	SP	988	08789
10	5	10	2010-10-11	2	Jose	Rua B	Paraguacu	19700 000	SP	988	08789
11	3	55	2010-12-10	3	Juliana	Rua A	Assis	19800 000	SP	55	44
12	2	45	2010-11-30	2	Thiago	Tiradentes	Maracaí	11	SP	12	11
13	7	60	2010-12-10	5	Victor	Rua D	Maracai	19840 000	SP	09988	0998
14	8	52	2010-11-05	2	Viviane Loiola	Rua E	Candido Mota	19880 000	SP	080876	089787

Figura 9- Resultado da Query Outer Join

3.4.5 Cross Join

Esta operação retorna todas as linhas das tabelas a serem consultadas. Não tem muito proveito porque dessa forma fica difícil fazer quaisquer referência cruzada. Aqui retorna todas as linhas de todas as tabelas que estão dentro da Query. Exemplo:

SELECT nome_cliente,
pedido.codigo_cliente,
numero_pedido
FROM cliente **CROSS JOIN** pedido



	nome_cliente character varying(50)	codigo_cliente integer	numero_pedido integer
1	Elaine	5	10
2	Thiago	5	10
3	Juliana	5	10
4	Joao	5	10
5	Jose	5	10
6	Cris	5	10
7	Victor	5	10
8	Viviane Loiola	5	10
9	Carol	5	10
10	Davi	5	10
11	Elaine	9	20
12	Thiago	9	20
13	Juliana	9	20
14	Joao	9	20
15	Jose	9	20
16	Cris	9	20
17	Victor	9	20
18	Viviane Loiola	9	20
19	Carol	9	20
20	Davi	9	20
21	Elaine	4	30
22	Thiago	4	30
23	Juliana	4	30
24	Joao	4	30
25	Jose	4	30
26	Cris	4	30
27	Victor	4	30
28	Viviane Loiola	4	30
29	Carol	4	30
30	Davi	4	30
31	Elaine	2	45
32	Thiago	2	45
33	Juliana	2	45
34	Joao	2	45
35	Jose	2	45
36	Cris	2	45
37	Victor	2	45
38	Viviane Loiola	2	45
39	Carol	2	45
40	Davi	2	45
41	Elaine	5	32
42	Thiago	5	32
43	Juliana	5	32
44	Joao	5	32
45	Jose	5	32
46	Cris	5	32
47	Victor	5	32
48	Viviane Loiola	5	32
49	Carol	5	32
50	Davi	5	32
51	Elaine	7	60
52	Thiago	7	60
53	Juliana	7	60
54	Joao	7	60
55	Jose	7	60
56	Cris	7	60
57	Victor	7	60
58	Viviane Loiola	7	60
59	Carol	7	60
60	Davi	7	60
61	Elaine	9	70
62	Thiago	9	70
63	Juliana	9	70
64	Joao	9	70
65	Jose	9	70
66	Cris	9	70
67	Victor	9	70
68	Viviane Loiola	9	70
69	Carol	9	70
70	Davi	9	70
71	Elaine	3	55

Figura 10. Resultado da Query em Cross Join

3.5 INDEXAÇÃO

Índice é uma estrutura que permite o rápido acesso às linhas de uma tabela com base nos valores de uma ou mais colunas. Nada mais é do que uma tabela do banco de dados onde são armazenados os ponteiros, arrumados de forma ascendente ou descendente.

O SGBD utiliza através do ponteiro localiza a linha com o valor desejado, tudo isso é feito transparente para o usuário. A criação de índices dependerá muito do projeto do banco de dados e das necessidades de pesquisas formuladas pelos usuários, é recomendável a utilização de índices quando a tupla for pesquisada com frequência, quando as tuplas são pesquisadas por um intervalo de valores, quando as tuplas regularmente forem utilizadas joins e também por tuplas que são utilizadas em cláusulas WHERE.

Os índices não deverão ser utilizados quando a consulta pedir como resultado um volume maior de linhas, por ser um índice não grupado, pode ocupar muito espaço e não ser utilizado para o que ele foi chamado. E ainda não deve ser utilizado quando o índice cluster agrupado (que serve para pequenos grupos dentro da mesma tupla) e se o número que foi pedido na consulta for pequeno então devemos utilizar o índice não agrupado. Devem ser levados em conta, os seguintes aspectos antes de acionar o processo:

- Tipos de acesso: os tipos de acesso que são aceitos com eficiência. Os tipos de acesso podem incluir a localização de registros com um valor de atributo especificado e a localização de registros cujos valores de atributos se encontrem em um intervalo especificado.
- Tempo de acesso: significa avaliar o tempo gasto para encontrar determinado item de dados, ou conjunto de itens, usando a técnica em questão.
- Tempo de Inserção: o tempo de gasto para inserir um novo item de dados, isso vale dizer encontrar o local atual para inserir o novo item de dados além do tempo gasto para atualizar a estrutura do índice.
- Tempo de Exclusão: leva-se em consideração o tempo gasto da busca da consulta para depois cronometrar a exclusão e ainda também atualizar o índice.

- Tempo Adicional: Neste caso observamos o tempo da resposta da consulta pelo índice aplicado, desde que a quantidade de espaço seja moderada e esteja acontecendo um desempenho melhor, valerá a pena manter o índice.

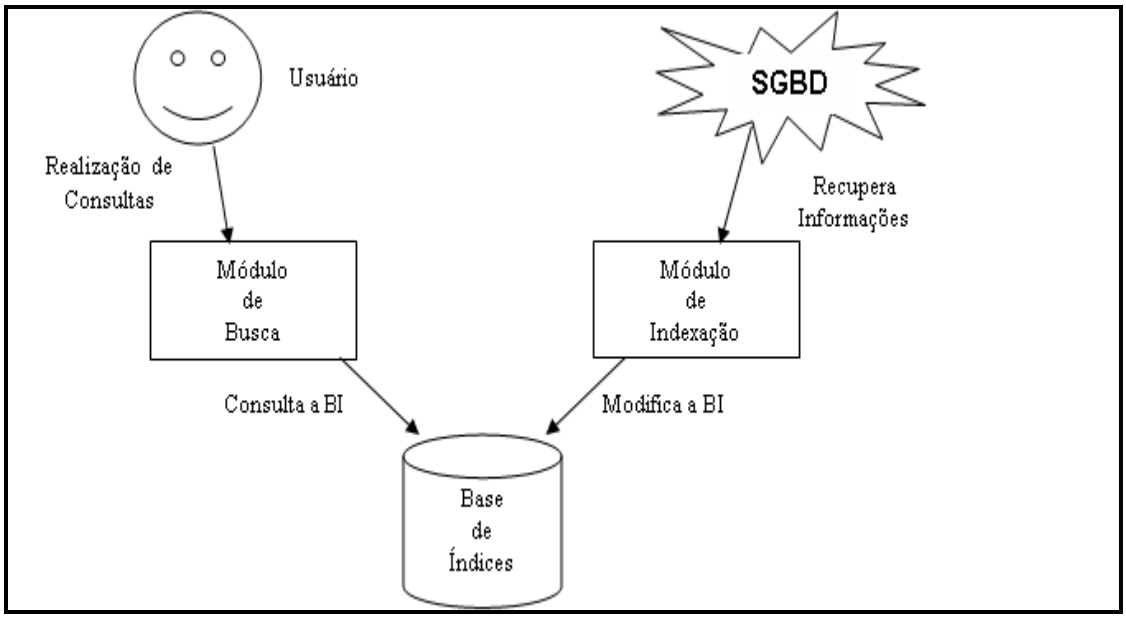


Figura 11. Arquitetura de engenho de busca.

O gráfico mostra o funcionamento da estrutura do Índice, na prática de uma busca as principais tarefas: poder recuperar informações do SGBD e disponibilizá-las para os usuários realizarem consultas sobre elas. Dentro dessa realidade, podemos destacar dois módulos principais: o **Módulo de Indexação**, que recupera as informações, e o **Módulo de Busca**, que as disponibiliza para consultas. Esses dois módulos realizam operações sobre a **Base de Índices (BI)** ou simplesmente índice, que contém o conjunto de documentos coletados pelo Módulo de Indexação, para que sejam consultados através de uma ordenação (Where) do Módulo de Busca. Ex:

```
CREATE [UNIQUE] INDEX <nome do índice>
ON <nome da tabela> (coluna(s)>);
```

A cláusula UNIQUE é opcional e define que para aquela coluna não existirão valores duplicados, ou seja, todos os dados armazenados na coluna serão únicos.A junção do índice unique e da especificação NOT NULL para uma coluna define a chave

primária da tabela, quanto ao aspecto lógico, pois uma chave primária não pode ser nula, neste caso somente árvore B pode ser do tipo Unique. Exemplo com várias colunas ficará assim:

```
CREATE INDEX idx_clientes_ps ON clientes (codigo, nome);
```

3.5.1 Atualização de Índice

Cada vez que um registro é inserido ou excluído, todos os índices terão que ser atualizados.

3.5.2 Inserção

Primeiro o sistema de banco de dados realiza a pesquisa usando o valor de chave de busca que aparece no registro a ser inserido. Em seguida é analisado:

- Se o valor da chave de busca não aparece no índice, o sistema insere no índice e associa a chave mais apropriada;
- Se o registro do índice armazenar ponteiros para todos os registros com o mesmo valor de chave de busca, o sistema acrescenta para o novo registro no registro de índice.
- Se o registro de índice armazenar um ponteiro somente no primeiro registro com o valor da chave de busca, o sistema então coloca o registro sendo inserido após os outros registros com os mesmos valores de chave de busca.

Chama-se de índice esparso quando o índice é armazenado, em blocos, se o sistema criar um novo bloco ele insere no índice o primeiro valor da chave de busca e ainda se o registro feito tiver o menor valor de chave de busca em seu bloco, o sistema atualiza a entrada de índice apontando para o bloco inserido.

3.5.3 Exclusão

Antes de iniciar esta ação, o sistema deverá primeiro fazer uma busca do registro a ser excluído, caso ele seja único, o sistema retira do índice o registro correspondente. Senão, o registro de índice armazena o ponteiro para todos os registros com o mesmo valor da chave do índice, o sistema exclui do registro de índice o ponteiro para o registro excluído. Se o índice não tiver um registro de índice com o valor da chave de busca do registro a ser excluído, nada é modificado no índice. Se ele for único no índice, o sistema substitui o registro de índice correspondente por um registro de índice para o próximo valor da chave de busca. Exemplo:

```
DROP INDEX <nome do índice>;
```

3.5.4 Índice de Clustering

É denominad de índice primário, indica a ordem sequencial do arquivo, isso significa que, para cada entrada no índice há um valor de chave que ocorre no registro de dados, a entrada aponta para o primeiro registro que contém aquele valor de chave. Chama-se de índice secundário, cuja chave aponta para todos os registros de dados com tal valor. O índice clustering é aquele que organiza grupos do mesmo tipo. Por exemplo, pode-se organizar o índice pelas funções, pelas colunas, pelas expressões, por ordem, etc. Nesta fase adota-se uma forma de estrutura para formar o índice.

3.5.5 Tipos de Índices

O PostgreSQL suporta atualmente quatro tipos de índices: **B-tree (árvore B), R-tree (árvore R), Hash e GiST.**

B-tree: é o tipo padrão (assumido quando nenhum índice é indicado). São índices que podem tratar consultas de igualdade e de faixa, em dados que podem ser classificados. Indicado para consultas com os operadores: <, <=, =, >=, >.

R-tree: tipo mais adequado a dados espaciais. Adequado para consultas com os operadores: <<, &<, &>, >>, @, ~=, &&.

Hash: indicados para consultas com comparações de igualdade simples.

GiST: é um índice especializado para alguns tipos de dados. Caso precisar utilizá-lo para caracter simples char ou valores inteiros.

Somente os tipos B-tree e GiST suportam índices com várias colunas. Índices com mais de um campo somente será utilizado se as cláusulas com os campos indexados forem ligados por AND. Um índice com mais de 03 atributos dificilmente será utilizado. São utilizadas para melhorar a performance do índice os comandos: FOREIGN KEY, ORDER BY, WHERE, ON, GROUP BY e HAVING. Exemplo:

Uma tabela contendo os CEPs do Brasil, com 633.401 registros.
Esta tabela sem nenhum índice executa a consulta abaixo:
SELECT * FROM cep_tabela WHERE cep = '60420440';
Em **7691 ms**.
Agora com o ÍNDICE:
ALTER TABLE cep_tabela ADD CONSTRAINT cep_pk PRIMARY KEY (cep);
A mesma consulta gasta apenas **10 ms**.
Analisando esse exemplo, verifica-se uma diferença gigantesca quando do uso de índices para realização de pesquisas.

3.6 MINERAÇÃO DE DADOS

Há uma grande diferença ente técnicas de mineração de dados e tarefa de mineração de dados. A tarefa consiste em encontrar algo fora da rotina, como por exemplo, um valor excessivo do cartão de crédito (um diferencial na tabela). Já a técnica consiste em formalizar métodos para garantir que se chegará nos padrões necessários no momento. A Figura 12 ilustra a descoberta de conhecimento em Banco de Dados KDD (*Knowledge Discovery Database*) (AMO, 2005).

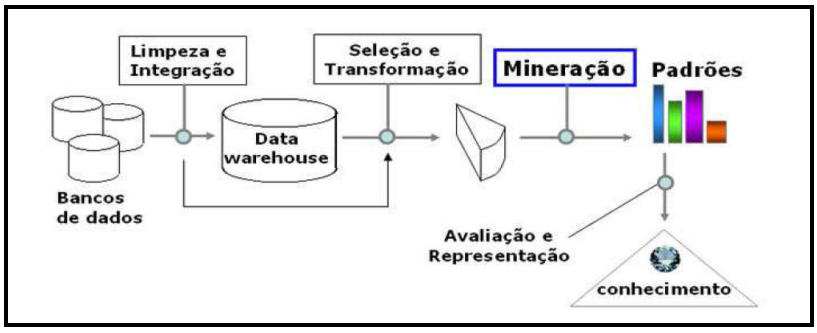


Figura 12. Descoberta de Conhecimento, retirada de Amo, 2005.

1- Banco de Dados: São selecionados os dados que serão utilizados.
2- Limpeza e Integração: Verificação de consistência entre as informações, correção de erros, preenchimento de valores desconhecidos, eliminação de informações redundantes, eliminação dos valores não pertencentes à aplicação.
3- Seleção e Transformação: São selecionados os atributos que serão precisos para fazer as comparações e chegar ao que será necessário para demonstrar uma estatística eficiente por exemplo. Logo são transformados no formato ideal pelos algoritmos através de uma operação de agregação.
4- Mineração: Consiste no processo de técnicas inteligentes com a finalidade de extrair os padrões do nosso interesse.
5- Avaliação e Pós Processamento: São identificados através dos testes feitos pelos critérios do usuário.

Visualização dos Resultados: O usuário tem a visualização concreta do que realmente desejou através da mineração. Exemplo prático através de uma função:

SE (Salário >= 2000) E (Idade > 25)
ENTÃO Cliente = "*Empréstimo concedido*"
SE (Salário < 1000) E (Idade < 25)
ENTÃO Cliente = "*Empréstimo negado*"

A mineração de Dados é aplicada através de um algoritmo eficiente.

4. PROPOSTA DE TRABALHO

Para o desenvolvimento do presente trabalho, será utilizado um modelo de dados, projetado para Administração de Cirurgias, retirado de (Machado, 2008). Serão transpostas as tabelas, através da criação de códigos SQL apenas olhando para o modelo físico no SGBD Postgresql 8.2, versão Free, assim como o modelo físico também será transcrito através da ferramenta MySQL Work Benck 5.0 Em seguida, serão feitos algumas inclusões de dados para povoar essas tabelas, no intuito de tornar possível os testes de performance das Querys.

Como ferramenta, será adotada o Explain, para visualizar o caminho de performance das consultas. Conforme o algoritmo vai mostrando, os comandos serão modificados no sentido de melhorar a qualidade da consulta, se funciona para poucos dados será igual para muitos dados, o SGBD Postgresql é muito moderno e eficiente, mas o conceito funciona para qualquer outro Banco de Dados.

A Figura 13 ilustra a proposta de forma geral.

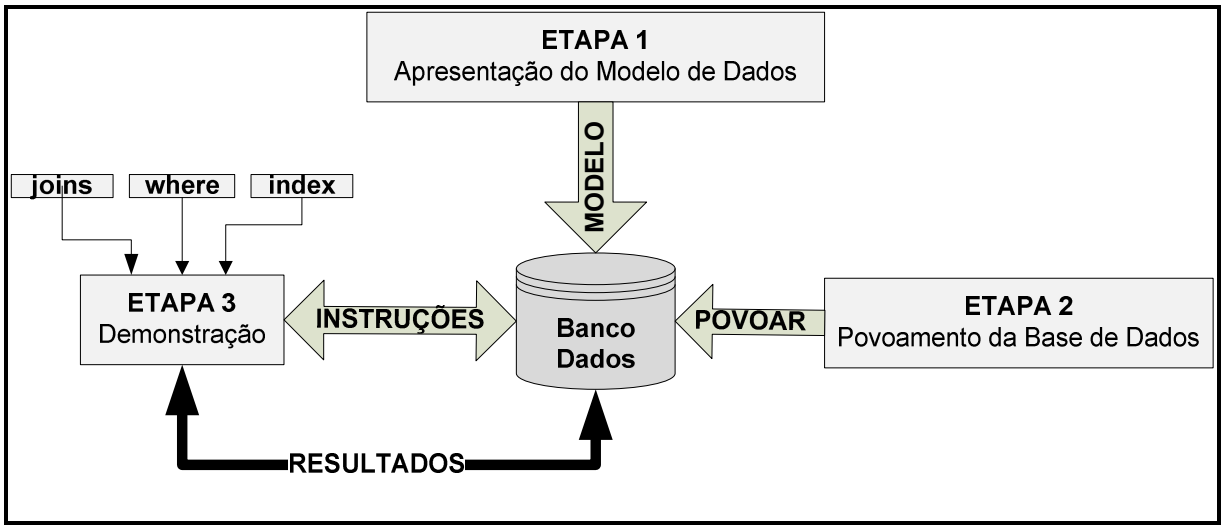


Figura 13. Proposta de Trabalho.

A seguir, serão descritas cada uma das etapas apresentadas na proposta de trabalho.

4.1 ETAPA 1: APRESENTAÇÃO DO MODELO DE DADOS

A primeira etapa da proposta tem por objetivo estudar as relações entre as tabelas, bem como entender tais relações (ligações) e o objetivo para que elas foram criadas. No mundo real em um hospital, existem diversas salas cirúrgicas e possuem recursos para grupos de especialidades médicas, sendo apropriadas para cirurgias de uma especialidade. Existem salas que prestam somente cirurgias no hospital e seus pacientes agendam cirurgias conforme disponibilidade das salas, informando as datas, hora inicial e hora final prevista, junto à operadora do sistema.

4.2 ETAPA 2: POVOAMENTO DA BASE DE DADOS

A segunda etapa tem por finalidade o armazenamento de dados em todas as tabelas, por completo (em todas as colunas) que conterão uma quantidade suficiente de registros para ser avaliado o tempo de resposta junto ao Banco de Dados. Dessa forma independente da quantidade a avaliação da query será a mesma. O importante a ser avaliado é que o tempo de execução será considerado até milésimos de segundos.

4.3 ETAPA 3: DEMONSTRAÇÃO E RESULTADOS

A terceira etapa, pretende-se demonstrar Views, através da ferramenta do Postgresql Free, Explain. Uma ferramenta completa que permite através de um módulo, executar os caminhos feitos pela query e visualizar os pontos que poderão ser melhorados. Serão testados os índices, quando deve ser utilizado ou não, também serão testadas várias QUERYS utilizando os comandos básicos junto à cláusula WHERE, HAVING, GROUP BY, todos possíveis com a ajuda da instrução SELECT. Será feito um planejamento para medir os resultados de diferentes consultas no banco de dados com a utilização de Índices quando a tupla precisar ser utilizada constantemente para obter um resultado com sucesso, entendendo que a consulta foi executada da melhor forma possível.

5. ESTUDO DE CASO

5.1 ETAPA 1: APRESENTAÇÃO DO MODELO DE DADOS

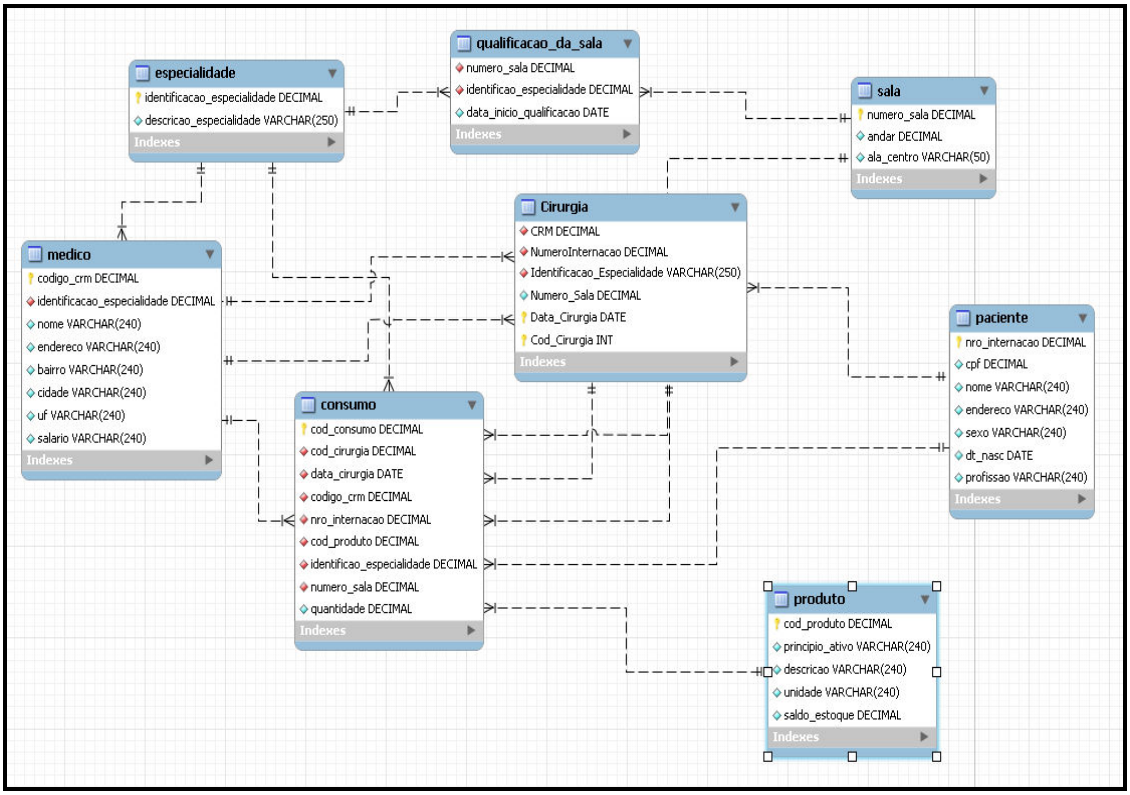


Figura 14. Modelo Físico “Administração de Cirurgias”, Machado, 2008.

5.2 ETAPA 2: POVOAMENTO DA BASE DE DADOS

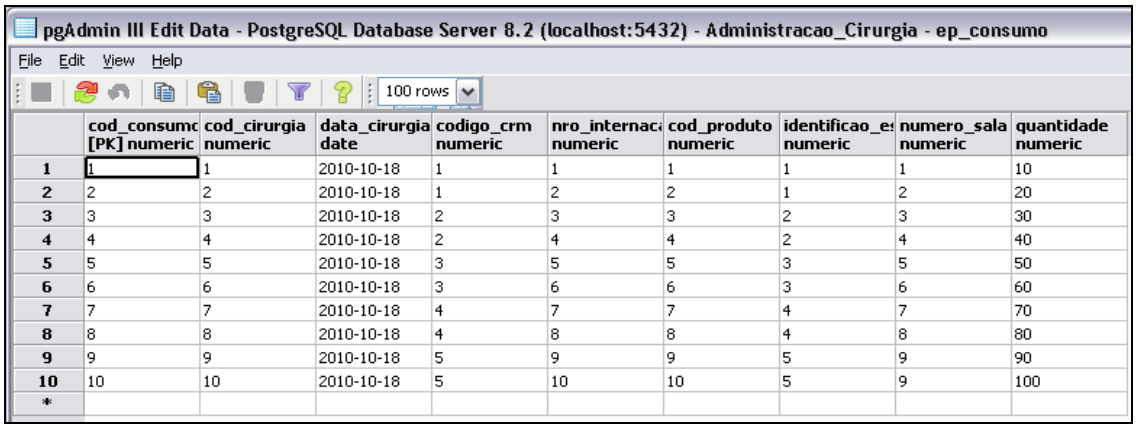
pgAdmin III Edit Data - PostgreSQL Database Server 8.2 (localhost:5432) - Administracao_Cirurgia - ep_cirurgia

File Edit View Help

100 rows

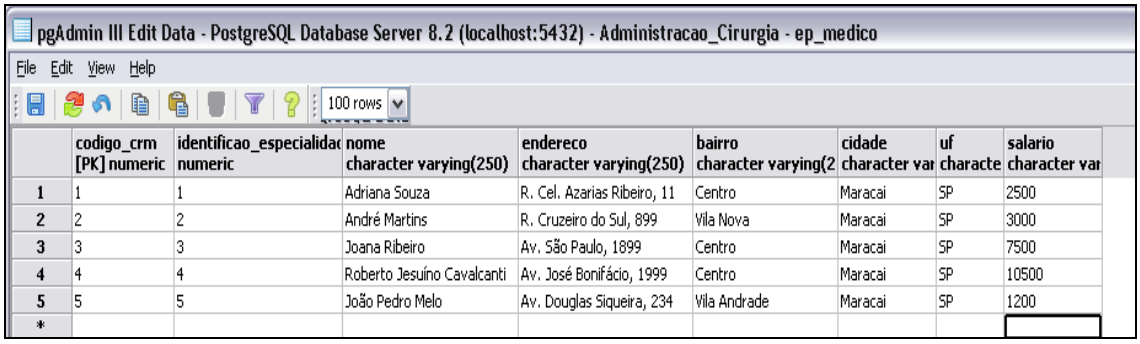
	cod_cirurgia [PK] numeric	codigo_crm numeric	nro_internacao numeric	identificao_especialidade numeric	numero_sala numeric	data_cirurgia [PK] date
1	1	1	1	1	1	2010-10-18
2	2	1	2	1	2	2010-10-18
3	3	2	3	2	3	2010-10-18
4	4	2	4	2	4	2010-10-18
5	5	3	5	3	5	2010-10-18
6	6	3	6	3	6	2010-10-18
7	7	4	7	4	7	2010-10-18
8	8	4	8	4	8	2010-10-18
9	9	5	9	5	9	2010-10-18
10	10	5	10	5	9	2010-10-18
*						

Figura 15. Dados da Tabela Cirurgia



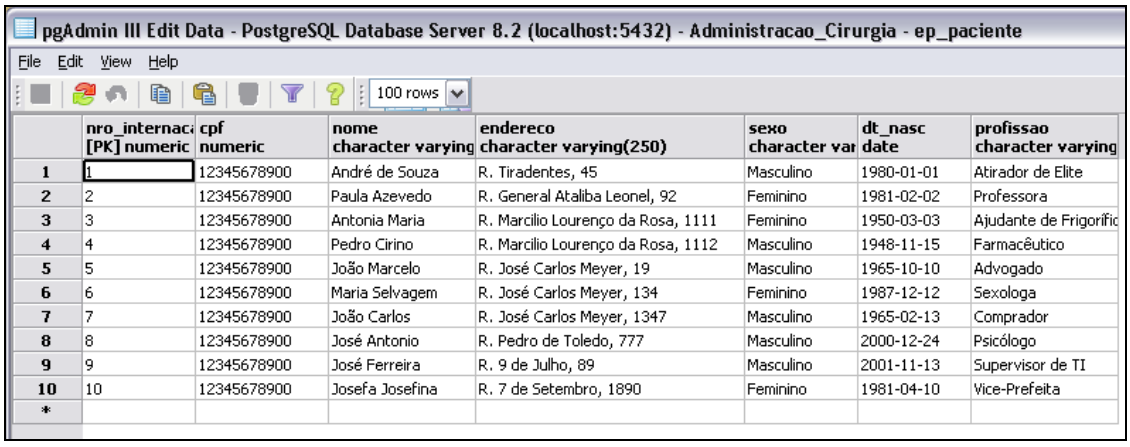
	cod_consumo [PK] numeric	cod_cirurgia numeric	data_cirurgia date	codigo_crm numeric	nro_internac numeric	cod_produto numeric	identificao_es numeric	numero_sala numeric	quantidade numeric
1	1	1	2010-10-18	1	1	1	1	1	10
2	2	2	2010-10-18	1	2	2	1	2	20
3	3	3	2010-10-18	2	3	3	2	3	30
4	4	4	2010-10-18	2	4	4	2	4	40
5	5	5	2010-10-18	3	5	5	3	5	50
6	6	6	2010-10-18	3	6	6	3	6	60
7	7	7	2010-10-18	4	7	7	4	7	70
8	8	8	2010-10-18	4	8	8	4	8	80
9	9	9	2010-10-18	5	9	9	5	9	90
10	10	10	2010-10-18	5	10	10	5	9	100
*									

Figura 16. Dados da Tabela Consumo



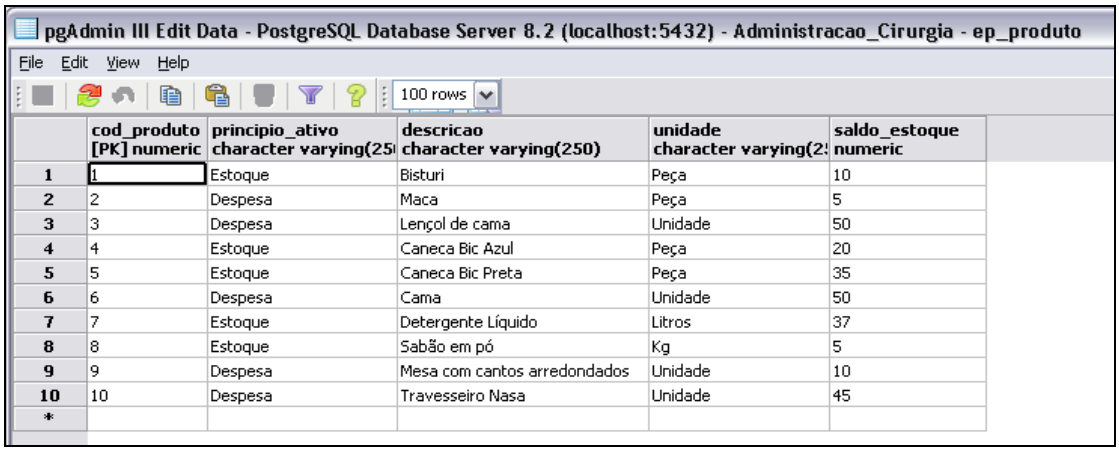
	codigo_crm [PK] numeric	identificao_especialidad numeric	nome character varying(250)	endereco character varying(250)	bairro character varying(2	cidade character var	uf characte	salario character var
1	1	1	Adriana Souza	R. Cel. Azarias Ribeiro, 11	Centro	Maracai	SP	2500
2	2	2	André Martins	R. Cruzeiro do Sul, 899	Vila Nova	Maracai	SP	3000
3	3	3	Joana Ribeiro	Av. São Paulo, 1899	Centro	Maracai	SP	7500
4	4	4	Roberto Jesuíno Cavalcanti	Av. José Bonifácio, 1999	Centro	Maracai	SP	10500
5	5	5	João Pedro Melo	Av. Douglas Siqueira, 234	Vila Andrade	Maracai	SP	1200
*								

Figura 17. Dados da Tabela Médico



	nro_internac [PK] numeric	cpf numeric	nome character varying	endereco character varying(250)	sexo character var	dt_nasc date	profissao character varying
1	1	12345678900	André de Souza	R. Tiradentes, 45	Masculino	1980-01-01	Atirador de Elite
2	2	12345678900	Paula Azevedo	R. General Ataliba Leonel, 92	Feminino	1981-02-02	Professora
3	3	12345678900	Antonia Maria	R. Marcilio Lourenço da Rosa, 1111	Feminino	1950-03-03	Ajudante de Frigorific
4	4	12345678900	Pedro Cirino	R. Marcilio Lourenço da Rosa, 1112	Masculino	1948-11-15	Farmacêutico
5	5	12345678900	João Marcelo	R. José Carlos Meyer, 19	Masculino	1965-10-10	Advogado
6	6	12345678900	Maria Selvagem	R. José Carlos Meyer, 134	Feminino	1987-12-12	Sexologa
7	7	12345678900	João Carlos	R. José Carlos Meyer, 1347	Masculino	1965-02-13	Comprador
8	8	12345678900	José Antonio	R. Pedro de Toledo, 777	Masculino	2000-12-24	Psicólogo
9	9	12345678900	José Ferreira	R. 9 de Julho, 89	Masculino	2001-11-13	Supervisor de TI
10	10	12345678900	Josefa Josefina	R. 7 de Setembro, 1890	Feminino	1981-04-10	Vice-Prefeita
*							

Figura 18. Dados da tabela Paciente



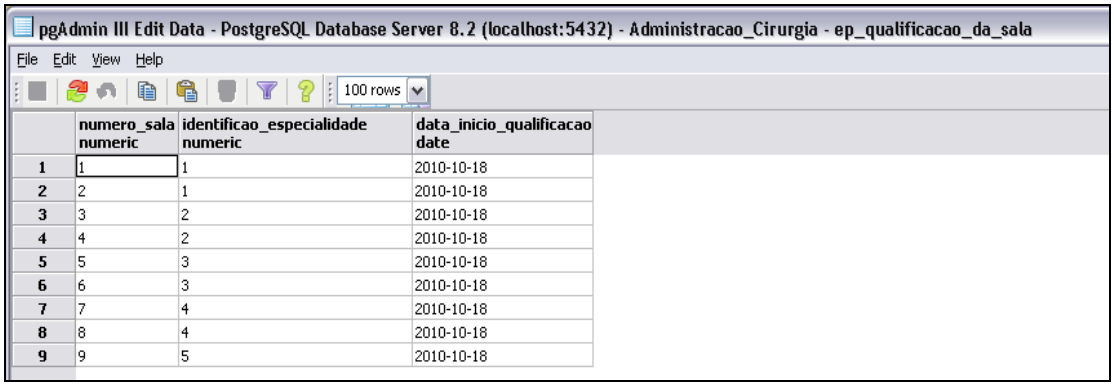
pgAdmin III Edit Data - PostgreSQL Database Server 8.2 (localhost:5432) - Administracao_Cirurgia - ep_produto

File Edit View Help

100 rows

	cod_produto [PK] numeric	principio_ativo character varying(25	descricao character varying(250)	unidade character varying(2	saldo_estoque numeric
1	1	Estoque	Bisturi	Peça	10
2	2	Despesa	Maca	Peça	5
3	3	Despesa	Lençol de cama	Unidade	50
4	4	Estoque	Caneca Bic Azul	Peça	20
5	5	Estoque	Caneca Bic Preta	Peça	35
6	6	Despesa	Cama	Unidade	50
7	7	Estoque	Detergente Líquido	Litros	37
8	8	Estoque	Sabão em pó	Kg	5
9	9	Despesa	Mesa com cantos arredondados	Unidade	10
10	10	Despesa	Travesseiro Nasa	Unidade	45
*					

Figura 19. Dados da tabela Produto



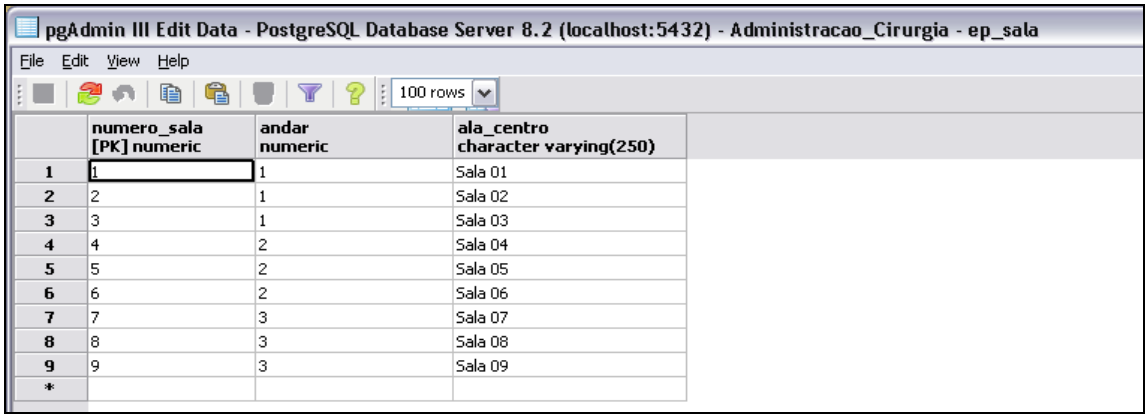
pgAdmin III Edit Data - PostgreSQL Database Server 8.2 (localhost:5432) - Administracao_Cirurgia - ep_qualificacao_da_sala

File Edit View Help

100 rows

	numero_sala numeric	identificao_especialidade numeric	data_inicio_qualificacao date
1	1	1	2010-10-18
2	2	1	2010-10-18
3	3	2	2010-10-18
4	4	2	2010-10-18
5	5	3	2010-10-18
6	6	3	2010-10-18
7	7	4	2010-10-18
8	8	4	2010-10-18
9	9	5	2010-10-18

Figura 20. Dados da tabela Qualificação da Sala



pgAdmin III Edit Data - PostgreSQL Database Server 8.2 (localhost:5432) - Administracao_Cirurgia - ep_sala

File Edit View Help

100 rows

	numero_sala [PK] numeric	andar numeric	ala_centro character varying(250)
1	1	1	Sala 01
2	2	1	Sala 02
3	3	1	Sala 03
4	4	2	Sala 04
5	5	2	Sala 05
6	6	2	Sala 06
7	7	3	Sala 07
8	8	3	Sala 08
9	9	3	Sala 09
*			

Figura 21. Dados da tabela Sala

5.3 ETAPA 3: DEMONSTRAÇÃO E RESULTADOS

Realizando consulta com uso do INNER JOIN que obrigatoriamente faz usar o ON para informar os campos a serem ligados.

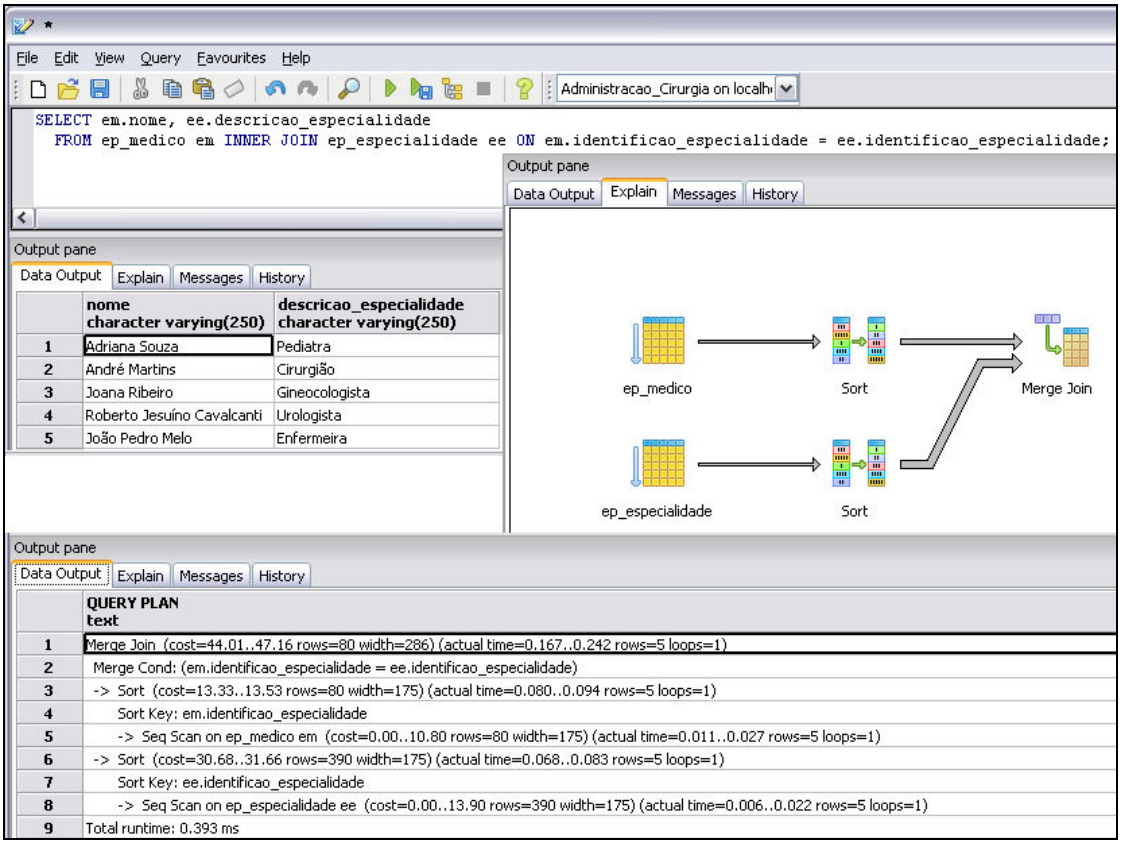


Figura 22. Análise do Inner Join x From

From X Join, retorna todos os nomes da tabela médico e também todas as profissões da tabela especialidade. Logo no gráfico pode se ver que em primeiro lugar o banco teve que percorrer todas as linhas de uma tabela comparando com as linhas da segunda tabela, em seguida em algum momento o algoritmo tem que organizar essas linhas de maneira que cada nome corresponda verdadeiramente a sua especialidade, por último para retornar o resultado da query. Na parte QUERY PLAN da Figura 22, tem-se a análise feita que mostra uma simulação do custo da utilização dos comandos além do que ele percorreu toda a tabela várias vezes, degradando a resposta da query.

Agora será feito o mesmo teste, porém com o uso da cláusula JOIN, sem precisar informar os atributos para amarração como no passo anterior e retornando o mesmo resultado.

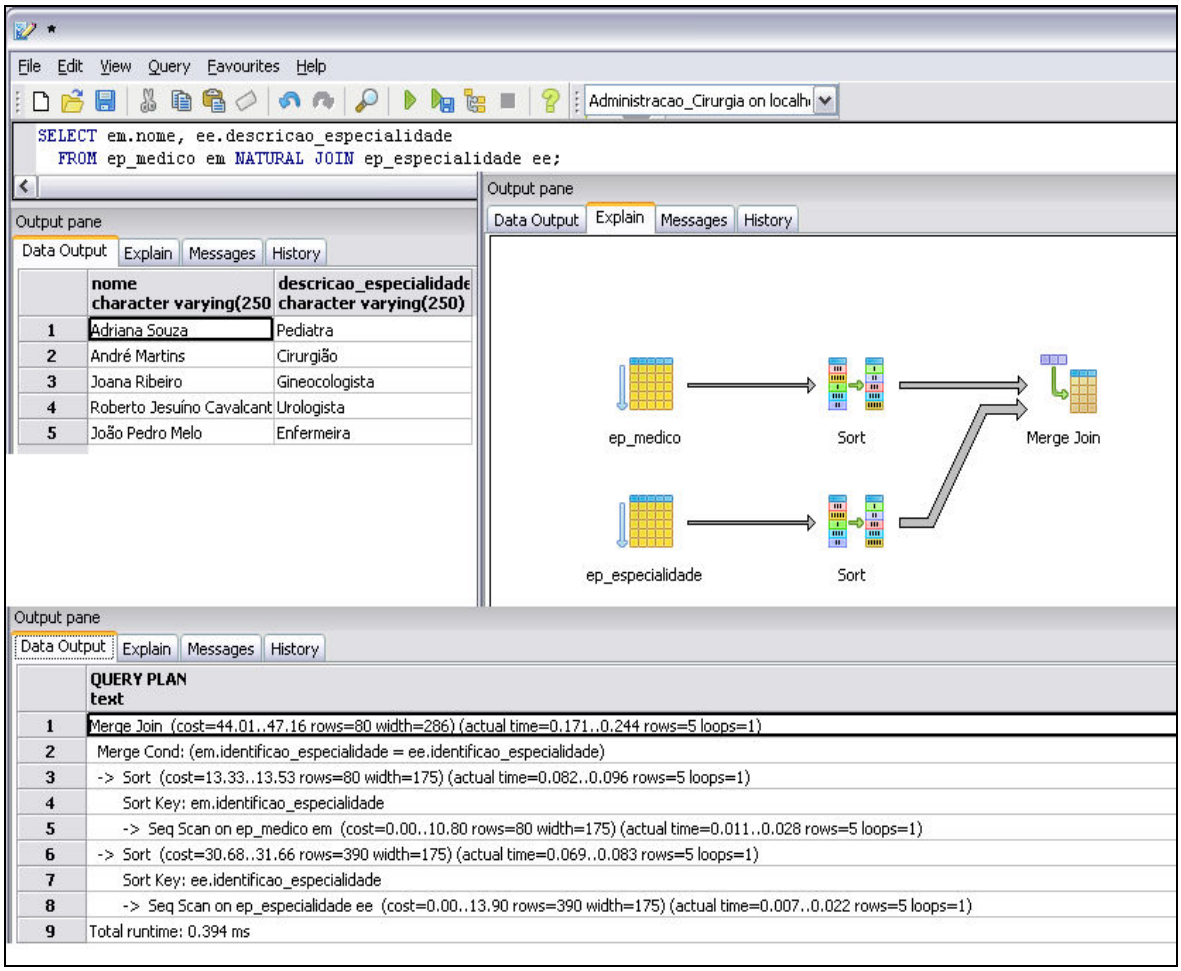


Figura 23. Análise do Inner Join Natural

Mesmo alterando a forma da consulta, o analisador, por intermédio do algoritmo, buscou através da comparação os atributos compatíveis, mas os resultados foram os mesmos, bastante queda de performance.

Outro teste, com as mesmas tabelas, agora utilizando o comando Cross Join, que retorna um produto cartesiano (terceira tabela).

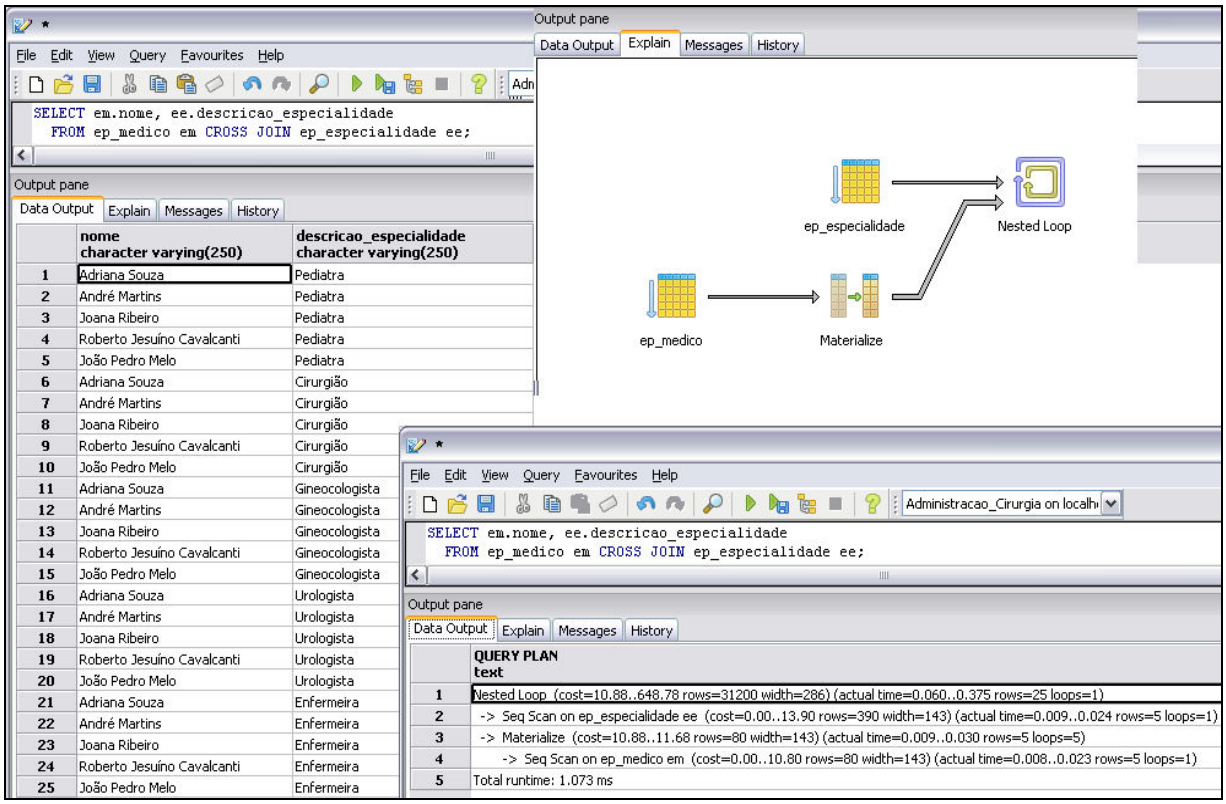


Figura 24. Cross Join (Produto Cartesiano)

De acordo com os resultados obtidos não faz sentido fazer esse tipo de consulta a menos que seja uma grande necessidade. No produto cartesiano é preciso comparar cada linha de uma tabela com a coluna da segunda tabela, conforme as comparações mostram igualdade isso é colocado em uma terceira tabela em algum momento isso tem que ser organizado dentro de um loop para mostrar o resultado total de todas as linhas inseridas na tabela. Na simulação de custo é bastante alta a utilização desse comando.

Para solucionar este caso a maneira correta de se fazer um join eficiente seria além de apelidar as tabelas usar a cláusula Where que é bastante clara na definição, de forma que a maneira como ela é escrita o analisador sintático retorna com sucesso.

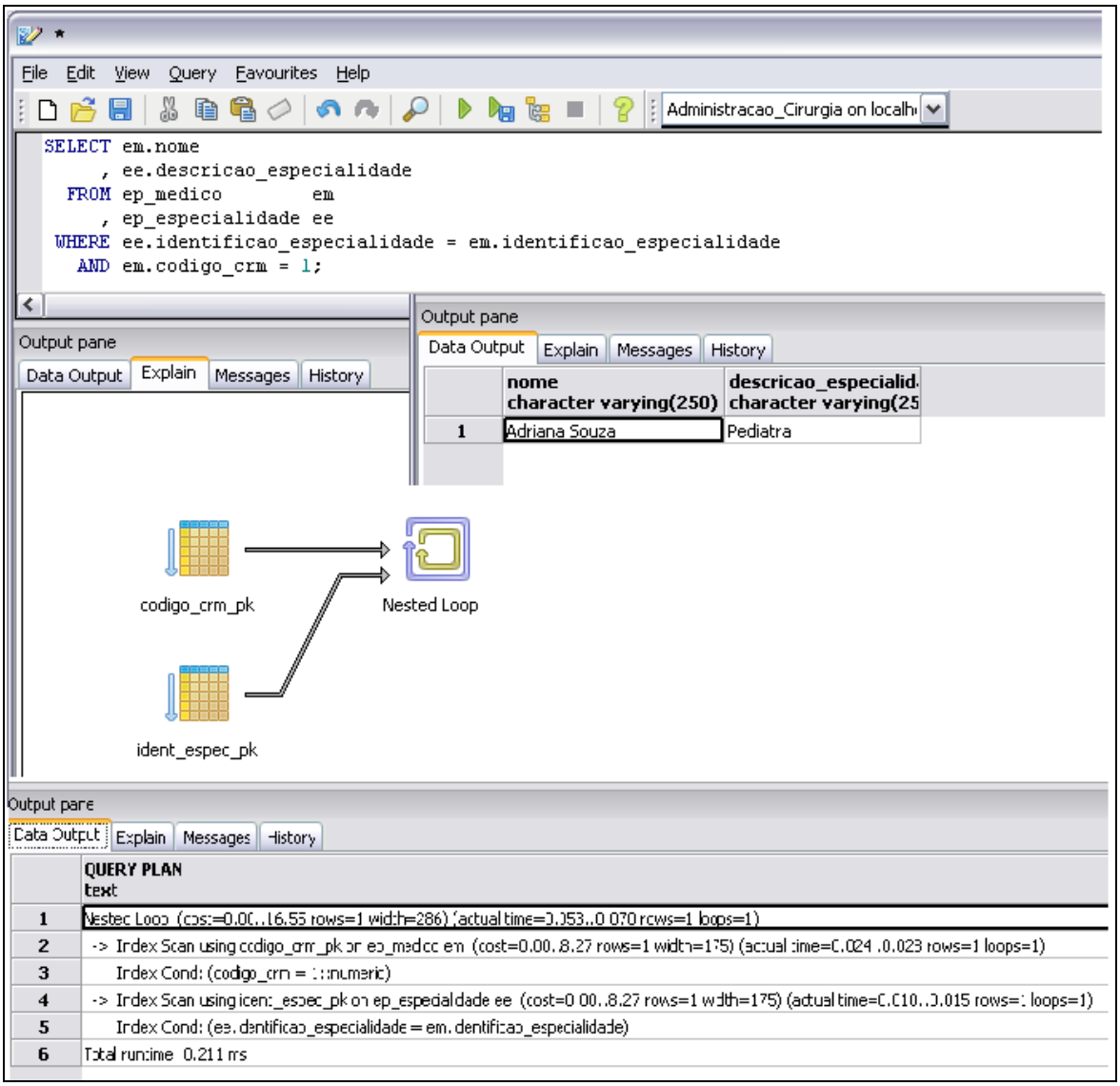


Figura 25. Join otimizando o nome da tabela e usando o From x Where

Join x Where muito utilizado e traz melhores resultados. Observe o gráfico do otimizador explain foi utilizado o índice através da chave primária em ambas as tabelas fazendo um loop até encontrar a linha desejada.

Suponha que fosse possível o mesmo médico atender dois pacientes distintos no mesmo dia e ao mesmo tempo em duas salas diferentes, só para testar essa busca teria que percorrer todas as tabelas para me dar o resultado de quantas vezes isso aconteceu e quais os nomes dos médicos, nome do paciente, numero da sala, etc.

Exemplo ficaria assim:

```
SELECT em.codigo_crm                  as

"CODIGO_CRM"
    , em.nome                         as

"NOME_MEDICO"
    , ee.descricao_especialidade      as

"ESPECIALIDADE"
    , ep.nome                         as

"NOME_PACIENTE"
    , ep.cpf                          as        "CPF"
    , ep.dt_nasc                      as        "DT_NASC"
    , es.numero_sala                  as

"NUMERO_SALA"
    , es.andar                        as        "ANDAR"
    , es.ala_centro                   as

"ALA_CENTRO"
    , eqs.data_inicio_qualificacao        as

"DATA_INICIO_QUALIFICACAO"
    , eco.data_cirurgia               as

"DATA_CIRURGIA"
    , eco.quantidade                  as

"QUANTIDADE"
    , epo.cod_produto                 as

"COD_PRODUTO"
    , epo.principio_ativo             as

"PRINCIPIO_ATIVO"
    , epo.descricao                   as        "DESCRICAO"
    , epo.unidade                     as        "UNIDADE"
    , epo.saldo_estoque               as

"SALDO_ESTOQUE"
  FROM ep_produto           epo
    , ep_consumo          eco
    , ep_qualificacao_da_sala eqs
    , ep_sala             es
    , ep_paciente         ep
    , ep_cirurgia         ec
    , ep_especialidade    ee
    , ep_medico           em
 WHERE em.codigo_crm          IN (1, 2, 3, 4, 5)
```

```
  AND em.identificao_especialidade = ee.identificao_especialidade
  AND em.identificao_especialidade = ec.identificao_especialidade
  AND em.codigo_crm                = ec.codigo_crm
  AND ec.nro_internacao            = ep.nro_internacao
  AND ec.numero_sala               = es.numero_sala
  AND es.numero_sala               = eqs.numero_sala
  AND ec.identificao_especialidade = eqs.identificao_especialidade
  AND em.codigo_crm                = eco.codigo_crm
  AND ee.identificao_especialidade = eco.identificao_especialidade
  AND ec.cod_cirurgia              = eco.cod_cirurgia
  AND ep.nro_internacao            = eco.nro_internacao
  AND es.numero_sala               = eco.numero_sala
  AND eco.cod_produto              = epo.cod_produto
ORDER BY 1;
```

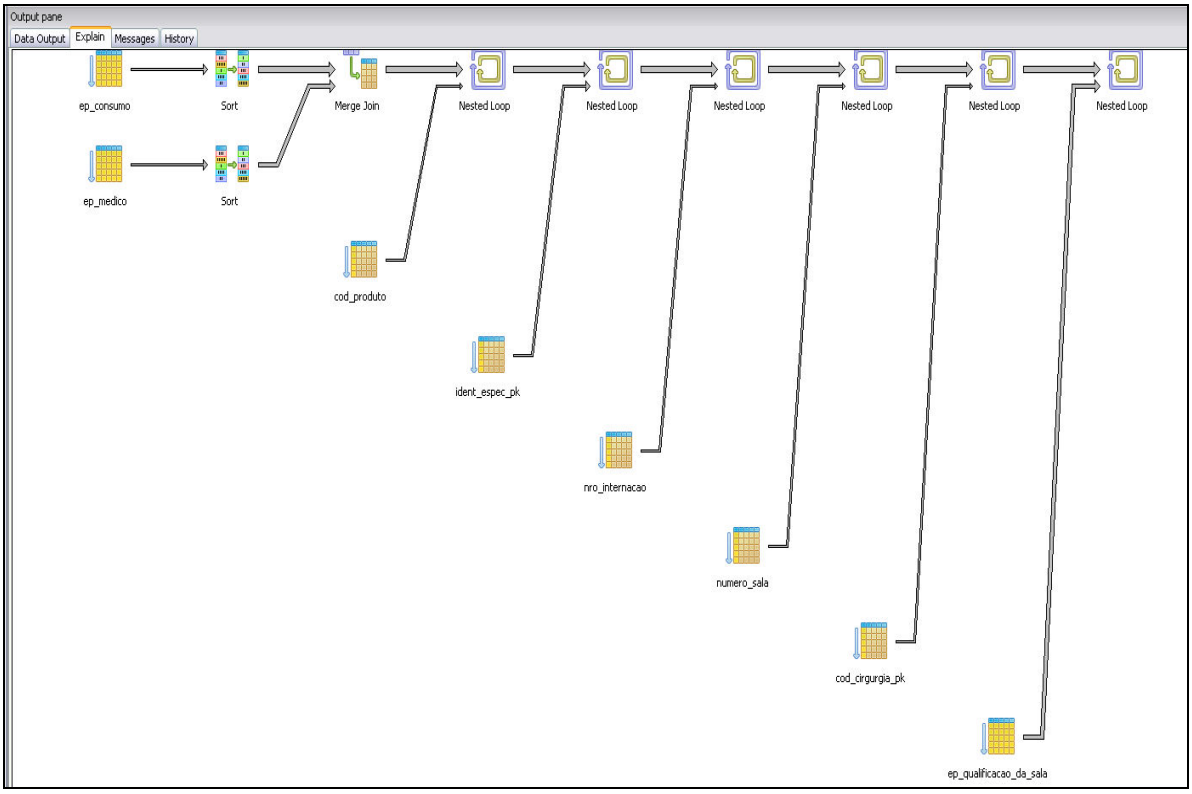
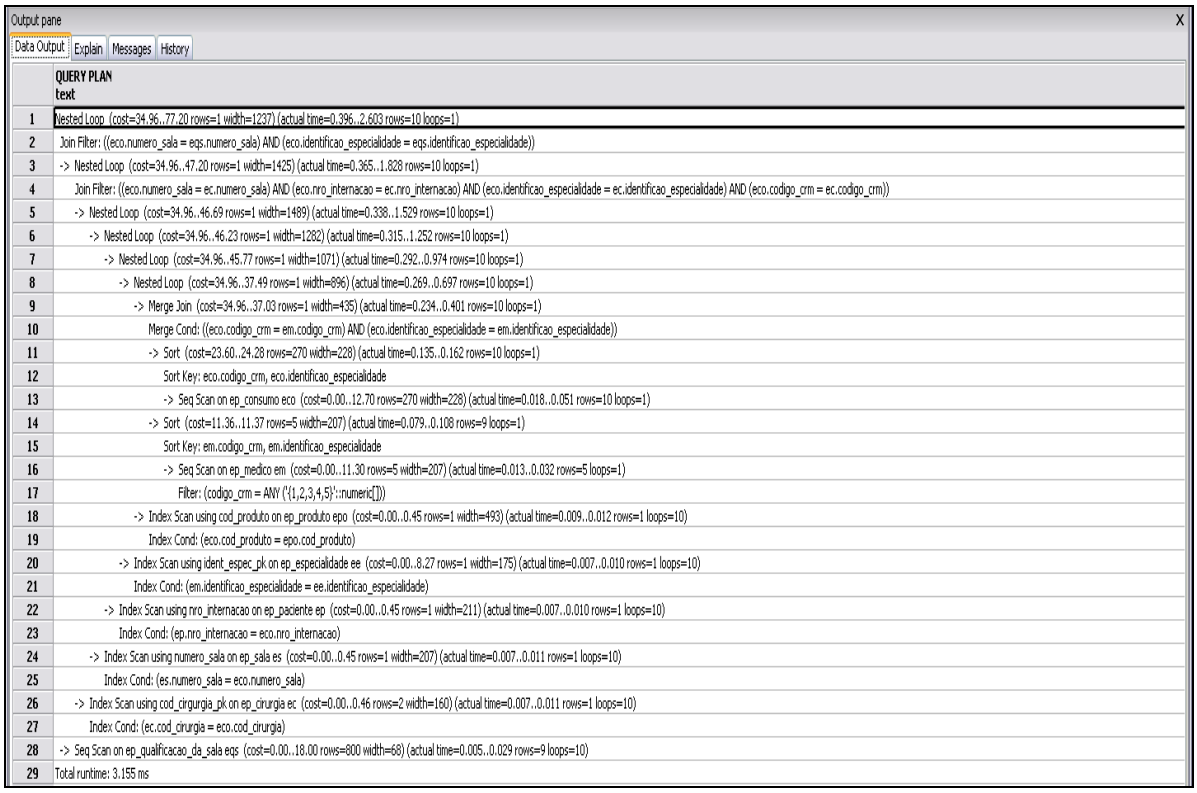


Figura 26. Explain Teste percorrendo todas as tabelas

Veja que mesmo com mais comandos, mais dados e mais tabelas, as ações foram analisadas da mesma forma que nas anteriores, para solucionar este caso, precisa forçar a tabela a usar o índice que será criado nesse momento, porque as duas primeiras tabelas não fizeram a utilização e também a ultima tabela.



Output pane

Data Output | Explain | Messages | History

	QUERY PLAN text
1	Nested Loop (cost=34.96..77.20 rows=1 width=1237) (actual time=0.396..2.603 rows=10 loops=1)
2	Join Filter: ((eco.numero_sala = eqs.numero_sala) AND (eco.identificao_especialidade = eqs.identificao_especialidade))
3	-> Nested Loop (cost=34.96..47.20 rows=1 width=1425) (actual time=0.365..1.828 rows=10 loops=1)
4	Join Filter: ((eco.numero_sala = ec.numero_sala) AND (eco.nro_internacao = ec.nro_internacao) AND (eco.identificao_especialidade = ec.identificao_especialidade) AND (eco.codigo_crm = ec.codigo_crm))
5	-> Nested Loop (cost=34.96..46.69 rows=1 width=1489) (actual time=0.338..1.529 rows=10 loops=1)
6	-> Nested Loop (cost=34.96..46.23 rows=1 width=1282) (actual time=0.315..1.252 rows=10 loops=1)
7	-> Nested Loop (cost=34.96..45.77 rows=1 width=1071) (actual time=0.292..0.974 rows=10 loops=1)
8	-> Nested Loop (cost=34.96..37.49 rows=1 width=896) (actual time=0.269..0.697 rows=10 loops=1)
9	-> Merge Join (cost=34.96..37.03 rows=1 width=435) (actual time=0.234..0.401 rows=10 loops=1)
10	Merge Cond: ((eco.codigo_crm = em.codigo_crm) AND (eco.identificao_especialidade = em.identificao_especialidade))
11	-> Sort (cost=23.60..24.28 rows=270 width=228) (actual time=0.135..0.162 rows=10 loops=1)
12	Sort Key: eco.codigo_crm, eco.identificao_especialidade
13	-> Seq Scan on ep_consumo eco (cost=0.00..12.70 rows=270 width=228) (actual time=0.018..0.051 rows=10 loops=1)
14	-> Sort (cost=11.36..11.37 rows=5 width=207) (actual time=0.079..0.108 rows=9 loops=1)
15	Sort Key: em.codigo_crm, em.identificao_especialidade
16	-> Seq Scan on ep_medico em (cost=0.00..11.30 rows=5 width=207) (actual time=0.013..0.032 rows=5 loops=1)
17	Filter: (codigo_crm = ANY ('{1,2,3,4,5}'::numeric[]))
18	-> Index Scan using cod_produto on ep_produto epo (cost=0.00..0.45 rows=1 width=493) (actual time=0.009..0.012 rows=1 loops=10)
19	Index Cond: (eco.cod_produto = epo.cod_produto)
20	-> Index Scan using ident_espec_pk on ep_especialidade ee (cost=0.00..8.27 rows=1 width=175) (actual time=0.007..0.010 rows=1 loops=10)
21	Index Cond: (em.identificao_especialidade = ee.identificao_especialidade)
22	-> Index Scan using nro_internacao on ep_paciente ep (cost=0.00..0.45 rows=1 width=211) (actual time=0.007..0.010 rows=1 loops=10)
23	Index Cond: (ep.nro_internacao = eco.nro_internacao)
24	-> Index Scan using numero_sala on ep_sala es (cost=0.00..0.45 rows=1 width=207) (actual time=0.007..0.011 rows=1 loops=10)
25	Index Cond: (es.numero_sala = eco.numero_sala)
26	-> Index Scan using cod_cirgurgia_pk on ep_cirurgia ec (cost=0.00..0.46 rows=2 width=160) (actual time=0.007..0.011 rows=1 loops=10)
27	Index Cond: (ec.cod_cirurgia = eco.cod_cirurgia)
28	-> Seq Scan on ep_qualificacao_da_sala eqs (cost=0.00..18.00 rows=800 width=68) (actual time=0.005..0.029 rows=9 loops=10)
29	Total runtime: 3.155 ms

Figura 27. Analisador de Custos

Quando enfim o trabalho do SGBD se torna maior, a visualização do custo é muito mais clara, há uma diferença muito grande quando o índice é utilizado e quando não é utilizado, mas não ficou totalmente com a performance perfeita, me trará os resultados, mas neste momento os testes terão que mudar, para otimizar ainda mais o trabalho do banco de dados. Tentei aqui criar um índice para a tabela consumo e para tabela médico, mesmo que criando vários índices além da chave primária o resultado gráfico ainda ficou o mesmo.

Os testes abaixo não tem solução, a única maneira de obter o resultado é percorrendo toda a tabela que em grandes bancos de dados pode demorar um pouco, além do que o custo sobe rapidamente, esse tipo de consulta deve ser evitada.

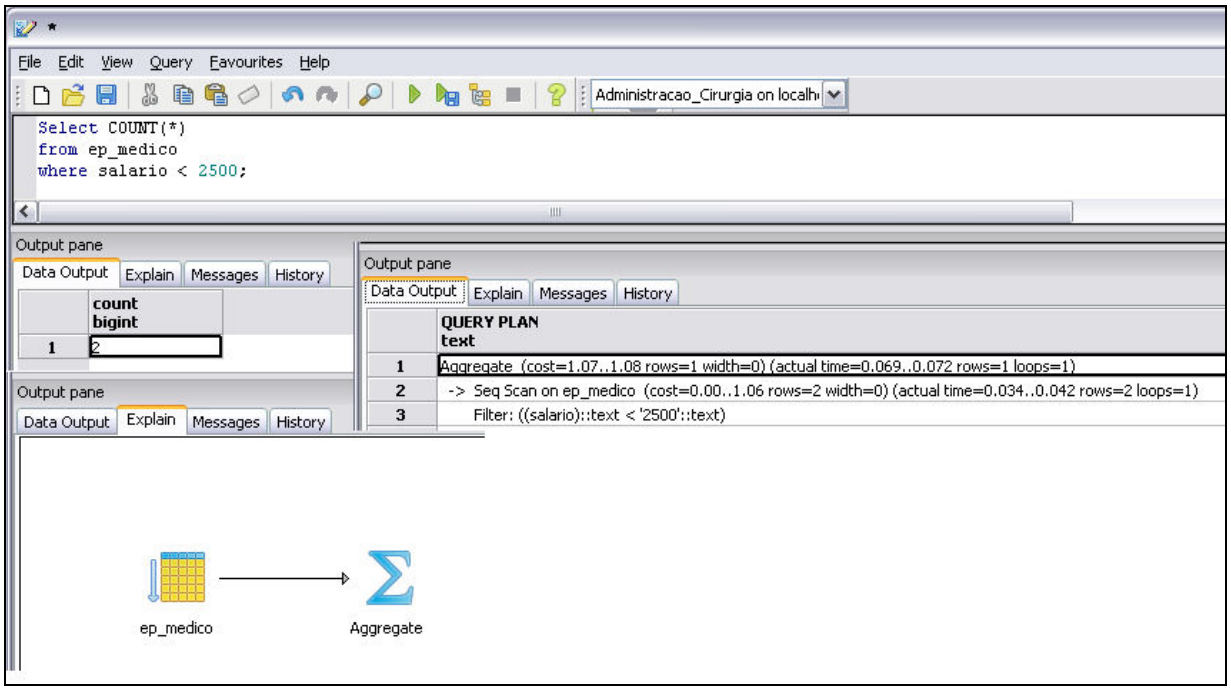


Figura 28. Figura Select Count tabela Médico

Percorreu toda a tabela médico e comparando sempre a linha próxima com a anterior e guardando o resultado de acordo com a condição imposta, até chegar ao fim.

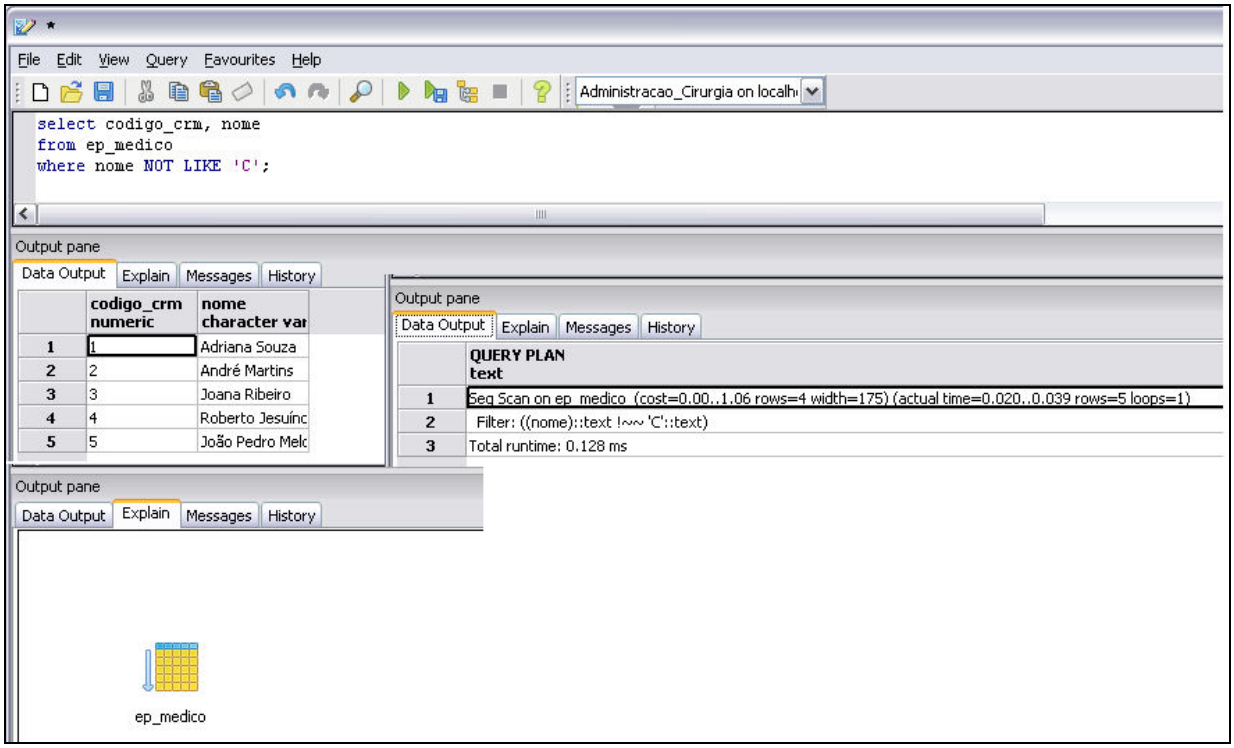


Figura 29. Figura Select Not Like tabela Médico

Seq Scan percorreu toda a tabela comparando linha a linha para obter todos os médicos que não começam com a Letra C.

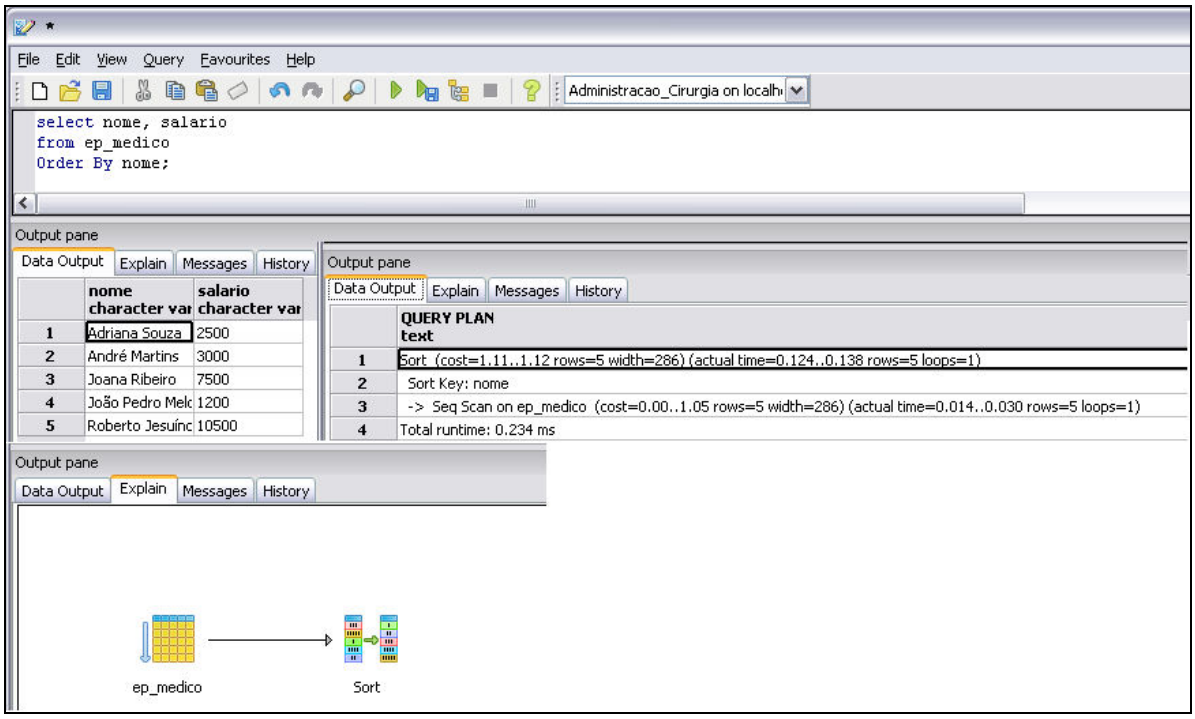


Figura 30. Figura Exemplo Order By tabela Médico

Aqui mesmo caso, percorreu toda a tabela para ordenar em ordem crescente.

6. CONSIDERAÇÕES FINAIS

É muito importante ressaltar que a tecnologia Tuning é utilizada em qualquer aplicação, não exclusivamente para o banco de dados. Detalhando um pouco mais sobre este trabalho, o tuning pode resolver, por exemplo, a transmissão dos pacotes da rede do banco de dados com grande extensão, se problema estiver no Hardware, estouro de memória, a tecnologia consegue solucionar por meio da realocação de memória ou se o problema for com o Sistema Operacional que esta influenciando na performance do banco a tecnologia tuning administra os processos e evita a queda de performance, registrando os resultados, até chegar no funcionamento do tempo desejado, melhorando o desempenho das transações como um todo. Porém aqui foi abordado ajustes do código SQL que é o caminho básico para que possa seguir com os testes, mas não basta conhecer apenas, mas saber como aplicá-la de forma correta, porque o DBA tem grande poder de decisão. Uma vez analisado de forma detalhista, bem estruturada, uma grande parte dos problemas já são solucionados, mesmo nesta primeira parte de análise que é bem mais demorada, porém quando é implantado bem analisado, só ocorrerá queda de performance se o banco de dados for mal conduzido, mas de qualquer forma é bem menor do que se o banco não for bem estruturado. Claro que a escolha da utilização do SGBD é a primeira etapa a ser avaliada, bem como o uso de uma boa metodologia, mas não ter um hardware e um software compatível o sucesso não ocorrerá.

Atualmente os Bancos de Dados como o Postgresql, Free, tem eficientes algoritmos para avaliação de performance que é pouco utilizado pelos desenvolvedores, mas se todos os testes seguissem a risca as suas avaliações, o banco de dados nunca perderia a performance.

De maneira geral as descobertas que muitas vezes estavam implícitas, mas nunca tinha atentado para tantos detalhes e principalmente de forma sequencial, e uma vez que entendi a lógica despertava mais a vontade de continuar descobrindo. Gostei muito de utilizar o SGBD Postgresql é bem fácil de manusear, entender, é bastante eficiente e prático, rápido, visual.

Corrigir os erros lógicos, testar os comandos que me forçaram a guardar a forma que eles faziam acontecer para que a cada testa com comandos diferentes, utilizava

o anterior até que consegui aplicar toda a parte básica da SQL, que não tinha a noção que eram tantos e que as formas de testes são quase infinitas. Daí, é a hora de seguir sim a tecnologia, principalmente outras experiências de projetos diferentes ajudam e muito a criar nosso próprio projeto, já que se não tivesse um plano de aplicação das tarefas não teria conseguido mostrar, os erros e nem o sucesso das querys, seguir as técnicas foram fundamentais para a conclusão, porque mesmo sendo uma aplicação pequena, são as mesmas funcionalidades para um banco de grande porte, a diferença está apenas nos dados de retorno de resultado que serão muito maiores.

Com relação aos testes, ficou claro que a escolha do SGBD e também as diferentes organizações de consultas levando em consideração o tipo dos atributos e também os inserts supostos influenciam muito nas respostas da query. O momento dos testes são minuciosos, pois a forma da escrita da SQL altera pouco ou outras vezes muito no caminho traçado pelo otimizador, é um processo demorado, por isso acredito que muitos desenvolvedores de softwares comerciais não utilizam essa tecnologia por causa do tempo gasto, como o SGBD é muito eficiente para o que eles precisam, não se preocupam muito com a performance da consulta.

Alguns testes demonstrados é certo que não há solução principalmente os que estão relacionados a agregação, fazer uma média, soma, Max, Min, etc. Outros testes relacionados as junções de tabelas funcionaram e outros não. Mas o que realmente funcionou de maneira precisa, foram os testes bem organizados, testados uma função por vez subseqüente até completar os loops.

A query plan é fundamental para medir os milésimos de segundos representativos demonstrando os acertos da mudança feita na otimização do código SQL e também as medições de custo para a empresa. Através das implementações é importante aprender bem todos os conceitos desde o primeiro ano, pois achei que nunca fosse precisar de várias disciplinas interligadas.

Trabalhos Futuros

Depois de todos os testes realizados algumas consultas obtiveram solução e outras aplicando a tecnologia não foram solucionadas. Para alcançar total sucesso de desempenho deverá ser estudado à fundo os conceitos do otimizador explain gráfico e também como o SGBD administra os algoritmos criados para o seu funcionamento, só assim poderá então solucionar totalmente a queda de performance, e também o custo, não restando nada mais a ser solucionado. Pode ainda virar um modulo da ferramenta open source para a próxima versão.

REFERÊNCIAS

AMO, Sandra. **Ténicas de Mineração de Dados 2005.** Trabalho de Conclusão de Curso. Faculdade de Computação. Universidade Federal de Uberlândia - Uberlândia-MG.

CARNEIRO, MOREIRA, FREITAS. 2006. Alessandro Pinto, Julinão Lucas, André Luis Castro. 2006. **Técnicas de Otimização de Banco de Dados Um Estudo Comparativo: MySQL e PostgreSQL**. Trabalho de Graduação. Centro de Ciências Computacionais – FURG – Universidade Federal do Rio Grande- RS, disponível em< http//www.seminfo.com.br> acessado em 11 de abril de 2010.

DATE, C.J.; **Introdução a Sistema de Banco de Dados**, Tradução de Daniel Vieira 8ª Edição Americana - – Rio de Janeiro: Elsevier, 2004.

LOZANO. Fernando. 2003. **Introdução ao Tuning do SGBD PostgreSQL. Consultor Independente** – Editor Adjunto – Revista Java Magazine- Profº Estácio de Sá e UniAbeu. Disponível em www.lozano.eti.br acessado em 12 de abril de 2010.

SILVA, Rafael Domingos. **Sql Tuning:Conceitos**.2005. Trabalho de Conclusão de Curso. Coordenadoria de Informática. FEMA – Fundação Educacional do Município de Assis. Assis-SP.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. **Sistemas de Banco de Dados.** Tradução de Daniel Vieira. 5ª edição – Rio de Janeiro: Elsevier, 2006.

MACHADO, Felipe Nery Rodrigues, **Banco de Dados (PROJETO E IMPLEMENTAÇÃO)** São Paulo – Editora Èrica Ltda – 2008 – 2 ª Edição - 2 ª Reimpressão.

MAYER, Roberto Carlos, **Otimizando a Performance de Bancos de Dados Relacionais,** Rio de Janeiro - Axcel Books do Brasil Editora Ltda – 2001.

TELLES, Marcelo Josué. 2006. **Uma abordagem prática de Tuning em Banco de Dados usando Postgresql.** UFRGS- Instituto de Informática Universidade Federal do Rio Grande do Sul- Porto Alegre – RS, disponível em <http//www.marcelo.kinghost.net/UFRGS/dB/documentSBC.pdf >acesso em 11 de Abril de 2010.

MELLO, Rafael Bevilacqua. 2009. **Migração entre Sistemas Gerenciadores de Banco de dados -** Trabalho de Conclusão de Curso. Coordenadoria de Informática. FEMA – Fundação Educacional do Município de Assis. Assis-SP.

MORELLI EDUARDO, MONTEIRO JOSÉ MARIA, ALMEIDA ANA CAROLINA, LIFSCHITZ SÉRGIO. 2006. **Reindexação Automática em SGBDs Relacionais -** Departamento de Informática - PUC-Rio disponível http://www.inf.pucrio.br/~postgresql/conteudo/publicationsfiles/MorMonAlmLifsbbd09 cameraReady_SBBD.pdf

LONEY KEVIN , KOCK GEORGE - Oracle 9i – A Referência Completa- 2002 - 1304 páginas.

http://www.batebyte.pr.gov.br/modules/conteudo/conteudo.php?conteudo=1592 acessado em junho de 2010.

http://pgdocptbr.sourceforge.net/pg82/tutorial-transactions.html acesso em junho de 2010.

www.oracle.com.br acessado em abril de 2010.

http://www.interplan.com.br/propaganda_tuning_oracle.htm acesso em abril de 2010.

http://www.projetos.unijui.tche.br/erbd2009/arquivo/51990.pdf. Acesso abril de 2010.

http://www.shammas.eng.br/acad/sitesalunos0606/rpt/tun_met.html acesso em abril de 2010.

http://www.sql.server.performance.com acesso em abril de 2010.

http://pt.wikibooks.org/wiki/PostgreSQL_Pr%C3%A1tico/DDL/%C3%8Dndices,_Tipos_de_Dados_e_Integridade_Referencial acessado em outubro de 2010.

http://www.excelenciaemgestao.org/Portals/2/documents/cneg5/anais/T8_0203_0548.pdf acessado em outubro de 2010.

http://www.socesp.org.br acessado em outubro de 2010.

Index scan

Foi feito uma varredura no índice da tabela

Table scan:

Foi feito um table scan na tabela

Aggregate, Hash Aggregate, Group Aggregate:

Usado com count, sum, min, max, avg, sttdev, variance. Ao ser usado com o GROUP BY, se o result set for pequeno, ele será carregado num hash e a agregação será feira nesse hash(Hash Aggregate).

Caso o resultset seja muito grande, será usado o Group Aggregate, sem o hash.

Append:

Usado em cláusulas UNION e em consultas envolvendo herança.

Bitmap Heap Scan:

É usado um bitmap (mapa de bits). Usado em colunas de baixa cardinalidade.

Bitmap Index Scan:

Hash:

Denota uma função hash executada na tabela (para ser usada posteriormente em um hash join ou hash left join)

Hash join, Hash Left Join [2]:

Aplicar o algoritmo de hash join em um inner join de duas relações funciona da seguinte maneira: primeiro, é preparada uma hash para a relação menor, aplicando uma função hash para cada linha , na columa a ser usada no join.Então, a relação maior é escaneada, em busca das linhas relevantes, procurando pela hash table. Usado com INNER JOIN, OUTER JOIN.

Limit:

Cláusula de limitação

Materialize:

Um sub-plano foi materializado em um arquivo temporário

Merge Join:

O Merge Join é um operador binário.Os lados esquerdo e direito são os fluxo de dados externo e interno, respectivamente. Ambos fluxos devem ser organizados de acordo com a chave do merge-join. Primeiro, uma linha do fluxo exterior é lido. Isso inicializa os valores das chaves do merge join. Então, linhas do fluxo interno são lidas até que uma linha tenha uma valor que case ou seja maior (exceto se a coluna chave é descendente) seja encontrada. Se as chaves da junção casam, então a coluna qualificadora é passada adiante para processamento adicional, e uma próxima chamada ao operador merge-join continua lendo do fluxo de dados atual.Se os novos valores são maiores que a chave de comparação, então esses valores são usados como a chave de junção, enquanto a traz linhas dos outros fluxos de dados.Esse processo continua até um dos fluxo de dados terminar.

Nested Loop (ou Nested Loop Left Join):

(Loop aninhado [1]) Se você faz o join em duas tabelas, e o otimizador do PosrtgreSQL escolhe um join do tipo Nested-Loop, então uma tabela será acessada como a tabela externa e a outra como a tabela interna. O PostgreSQL irá escanear a tabela externa linha a linha e para cada linha na tabela externa, ele irá escanear a tabela interna, procurando pelas linhas equivalentes. Usando com INNER JOIN e LEFT OUTER JOIN.

Function:

Resultado de uma função

Sort:

Ordenação. Usado explicitamente com a cláusula 'ORDER BY', porém é usado implicitamente ao tratar de Unique, Sort-Merge Joins e outros operadores.

Subconsulta:

Indica o uso de uma subconsulta.

Tid Scan:

Usado o Tuple ID da coluna, somente quando a cláusula 'ctid=' aparece na query. Muito raro, e muito rápido.(praticamente apenas consultas ao cátalogo do sistema)

Unique:

Usado em DISTINCT e Union. Não muda a ordenação, apenas elimina as linhas duplicadas. O grupo de entrada deve estar ordenado (forçará um SORT, se necessário)