

Otimizando consultas de dados com particionamento de tabela

Abstract

This article presents the table partitioning technique that will be applied to the financial table that belongs to the public company payroll system. This table has constant monthly growth as it receives all financial information from employees after closing the payroll, it is considered a table of historical data. In addition, it receives many queries to generate digital files for banks, Court of Auditors, Social Security, Internal Revenue Service, Internal Audit, print financial history and other reports. The payroll system was developed in Delphi 7 and has PostgreSQL as a Database Management System to store payroll and HR data. Therefore, the purpose of this article is to demonstrate the table partitioning technique in order to improve the response time of data queries.

Key words

Partitioning, Table, PostgreSQL, Tuning.

Eliano M Nascimento, Formado pela Universidade São Judas Tadeu em Tecnologia em Processamento de dados, é servidor público no cargo de Analista de Sistemas, atuando como administrador de Banco de dados.

I. INTRODUÇÃO

1. Contexto

Atualmente se exigem mais rapidez, eficiência no processamento e na disponibilidade dos dados. Estes pontos são cruciais para Sistema Gerenciador de Banco de Dados (SGBD) que tem como principal responsabilidade realizar consultas, inserções, atualizações, exclusões e disponibilidade dos dados com a máxima performance. No banco de dados com pouca informação esses aspectos não são tão perceptivos, mas nas bases de dados na grandeza do giga e terabytes é uma realidade. O crescente volume de dados gerados pelas empresas e governos estimulam cada vez mais as necessidades das empresas e profissionais de Tecnologia da Informação (TI) buscarem soluções para melhorar o desempenho dos SGBD. Este trabalho demonstra a técnica de particionamento de tabela como uma destas soluções que são aplicadas no banco de dados para melhorar o desempenho das consultas, inserções, remoções e atualizações dos dados.

2. Objetivo

O objetivo deste trabalho é aplicar a técnica de particionamento de tabela para melhorar o desempenho das consultas, inserções, remoções e atualizações dos dados em tabelas com grande volume de informações armazenadas.

3. Motivação

A motivação para usar a técnica de particionamento de tabela usando o PostgreSQL é a simplicidade de administrá-lo, por ser de código aberto e possuir características superiores a outros sistemas gerenciador de banco de dados de distribuição gratuita. Além disso, ele é o principal sistema de gerenciado de banco de dados adotado pela instituição e possui uma tabela histórica com volume grande de dados e crescimento constante.

4. Materiais

Este trabalho foi realizado utilizando os seguintes recursos tecnológicos:

- Sistema Gerenciador de Banco de Dados PostgreSQL 9.2 de 32 bit para sistema operacional Windows. O PostgreSQL é um sistema de gerenciamento de banco de dados objeto-relacional (SGBDOR) de código aberto baseado no POSTGRES, Versão 4.2, desenvolvido na Universidade da Califórnia, em Berkeley. Ele suporta grande parte do padrão SQL e oferece muitas características

modernas como: consultas complexas, chaves estrangeiras, integridade transacional e controle de concorrência. E por causa da licença liberal, o PostgreSQL pode ser usado, modificado e distribuído por qualquer pessoa gratuitamente para qualquer finalidade, seja privada, comercial ou acadêmica.

- Ferramenta DBDesigner 4;
- Ferramentas auxiliares: Excel, Bloco de notas;
- Computador Servidor IBM x3200, processador de 2.4 Ghz, RAM de 2GB e 4 HD de 120GB;
- Sistema Operacional Windows 2003 server.

Este trabalho utilizou para testes um Banco de Dados com as seguintes características:

- PostgreSQL versão 9.2 de 32 bits.
- Base de dados de folha de pagamento com 50 mil funcionários.
- Possuindo dezoito tabelas importantes para demonstrar a estrutura relacional.

5. Metodologia do Trabalho

Este trabalho foi realizado nas seguintes etapas:

- Na primeira etapa foram instalados os programas necessários no servidor 2, destino, e gerado backup total da base de dados no servidor 1, origem, para criar o banco de dados de trabalho. No SGBD de trabalho foi carregada a copia da base de dados RH, após este processo foi copiado os dados da tabela tffinanceira e também das tabelas com relação de vínculos com a tffinanceira através de script no formato para posterior importação já com a tabela particionada. Também foram observados os relacionamentos, índices aplicados nesta tabela para mantê-los na nova tabela.
- A segunda foi aplicada o particionamento, os relacionamentos e índices na tabela tffinanceira. Após este procedimento foi executado um teste para verificar se estava distribuindo os dados nas suas respectivas partições.
- Na terceira e ultima parte foi importado todos os dados nas suas respectivas tabelas e realizadas consultas para medir o desempenho.

II. REVISÃO BIBLIOGRAFICA

1. A tecnologia de particionamento de tabelas

O método de particionamento de tabela segue o mesmo conceito de particionamento de disco rígido (HD) físico de um computador. Aplicando o particionamento poderá dividi-lo logicamente em várias partes ou unidade lógicas, tornando estas unidades lógicas como se fossem discos rígidos independentes.

O particionamento de tabelas segue este raciocínio, onde é aplicada a técnica de particionamento para dividir tabela fisicamente com grande volume de dados em tabelas com menores quantidades de dados usando uma coluna denominada chave de particionamento como referencia para distribuição dos dados conforme critério adotado, como por exemplo: ano, mês, região e período, com isso tornando o acesso aos dados mais rápido e seu gerenciamento mais fácil.

Uma tabela particionada pode ter N partições, essas partições funcionam como tabelas normais (sem estar particionada), sendo que cada partição pode ter sua própria configuração de tablespace e índice, a única diferença que podem considerar entre uma tabela normal e uma tabela particionada é que a tabela particionada será sempre dependente da tabela que a originou e a tabela normal funciona de forma independente de qualquer outra tabela. Esta diferença para a aplicação acessar os dados é transparente o comportamento e estruturas da tabela normal e particionada são iguais.

Para ilustrar este conceito estão abaixo às figuras 1 e 2. Na primeira a tabela pedido foi particionada usando o ano da venda na coluna Dt_Venda como chave de particionamento para distribuir os dados. Nesta chave poderia também aplicar o critério de seleção no último dígito do ano para distribuir os dados nas partições. No primeiro critério a quantidade de tabela particionada cresce infinitamente ou até o limite suportado pelo SGBD, já o segundo limitaria até 10 (0 a 9) partições no máximo.

Na figura 2 mostra o particionamento da mesma tabela usando como chave de particionamento a coluna Cod_Regiao para dividir a tabela principal em cinco partes, limitando as em quantidade de regiões as quais irão armazenar os dados das vendas nas suas respectivas partições. Estes dois simplórios exemplos de seleção de dados para armazenamento, apresentam alguns detalhes que devem ser levados em consideração conforme o negócio da empresa antes de aplicar o particionamento em tabela, para evitar transtornos no futuro.

Num_Pedido	Valor	Dt_Venda	Cod_Regiao	Regiao
00001	5579.00	23/01/2010	1	Sul
00002	3508.89	18/03/2010	2	Sudeste
00003	4897.57	30/05/2010	3	Centro_Oeste
00004	5579.00	22/06/2010	4	Nordeste
00005	786.98	05/10/2010	5	Norte
00006	3508.89	12/01/2011	2	Sudeste
00007	4897.57	19/02/2011	3	Centro_Oeste
00008	9876.09	22/06/2011	5	Norte
00009	1768.90	09/09/2011	1	Sul
00010	687.99	27/12/2011	4	Nordeste
00011	342.56	11/01/2012	2	Sudeste
00012	987.34	29/04/2012	4	Nordeste
00013	123.18	15/07/2012	3	Centro_Oeste
00014	1234.76	02/08/2012	5	Norte
00015	289.05	17/11/2012	1	Sul

Num_Pedido	Valor	Dt_Venda	Cod_Regiao	Regiao
00001	5579.00	23/01/2010	1	Sul
00002	3508.89	18/03/2010	2	Sudeste
00003	4897.57	30/05/2010	3	Centro_Oeste
00004	5579.00	22/06/2010	4	Nordeste
00005	786.98	05/10/2010	5	Norte

Num_Pedido	Valor	Dt_Venda	Cod_Regiao	Regiao
00006	3508.89	12/01/2011	2	Sudeste
00007	4897.57	19/02/2011	3	Centro_Oeste
00008	9876.09	22/06/2011	5	Norte
00009	1768.90	09/09/2011	1	Sul
00010	687.99	27/12/2011	4	Nordeste

Num_Pedido	Valor	Dt_Venda	Cod_Regiao	Regiao
00011	342.56	11/01/2012	2	Sudeste
00012	987.34	29/04/2012	4	Nordeste
00013	123.18	15/07/2012	3	Centro_Oeste
00014	1234.76	02/08/2012	5	Norte
00015	289.05	17/11/2012	1	Sul

Figura 1- Particionamento por Dt_Venda

Num_Pedido	Valor	Dt_Venda	Cod_Regiao	Regiao
00001	5579.00	23/01/2010	1	Sul
00002	3508.89	18/03/2010	2	Sudeste
00003	4897.57	30/05/2010	3	Centro_Oeste
00004	5579.00	22/06/2010	4	Nordeste
00005	786.98	05/10/2010	5	Norte
00006	3508.89	12/01/2011	2	Sudeste
00007	4897.57	19/02/2011	3	Centro_Oeste
00008	9876.09	22/06/2011	5	Norte
00009	1768.90	09/09/2011	1	Sul
00010	687.99	27/12/2011	4	Nordeste
00011	342.56	11/01/2012	2	Sudeste
00012	987.34	29/04/2012	4	Nordeste
00013	123.18	15/07/2012	3	Centro_Oeste
00014	1234.76	02/08/2012	5	Norte
00015	289.05	17/11/2012	1	Sul

Num_Pedido	Valor	Dt_Venda	Cod_Regiao	Regiao
00001	5579.00	23/01/2010	1	Sul
00009	1768.90	09/09/2011	1	Sul
00015	289.05	17/11/2012	1	Sul

Num_Pedido	Valor	Dt_Venda	Cod_Regiao	Regiao
00002	3508.89	18/03/2010	2	Sudeste
00006	3508.89	12/01/2011	2	Sudeste
00011	342.56	11/01/2012	2	Sudeste

Num_Pedido	Valor	Dt_Venda	Cod_Regiao	Regiao
00003	4897.57	30/05/2010	3	Centro_Oeste
00007	4897.57	19/02/2011	3	Centro_Oeste
00013	123.18	15/07/2012	3	Centro_Oeste

Num_Pedido	Valor	Dt_Venda	Cod_Regiao	Regiao
00004	5579.00	22/06/2010	4	Nordeste
00010	687.99	27/12/2011	4	Nordeste
00012	987.34	29/04/2012	4	Nordeste

Num_Pedido	Valor	Dt_Venda	Cod_Regiao	Regiao
00005	786.98	05/10/2010	5	Norte
00008	9876.09	22/06/2011	5	Norte
00014	1234.76	02/08/2012	5	Norte

Figura 2 - Particionamento por Cod_Regiao

As vantagens do uso de particionamento de tabelas são mais perceptíveis em grandes ambientes de banco de dados, onde existe a necessidade de acesso aos dados mais rápidos, não somente por esta causa, mas também para outras funcionalidades dentro do banco, como: backup, balanceamento de I/O, melhor desempenho de instruções SQL, redução de contenção (disputa entre os processos) e disponibilidade dos dados.

A finalidade deste trabalho é demonstra que o particionamento de tabela pode melhorar a performance da consulta. Serão abordados na próxima seção para melhor entendimento os métodos de particionamento de tabelas dos principais SGBDs mais utilizados no mercado.

II. REVISÃO BIBLIOGRAFICA

2. Particionamento de tabelas nos principais SGBD

2.1 Oracle

A Oracle implementou pela primeira vez o método de particionamento de tabelas na versão 8.0 do seu SGBD e com lançamentos de novas versões foram incorporados novos métodos. Para melhor mais ainda o desempenho de acesso aos dados deve armazenar as partições em tablespace distintas, e se possível em disco diferente. Com isso, a facilidade de realização de manutenção fica mais simples, como por exemplo, backup, gerenciamento e disponibilidade dos dados caso uma tablespace fique inativa não compromete o acesso aos dados das demais tablespace. Os principais métodos estão relacionados abaixo.

- Particionamento por Faixa (Range Partitioning)

Este método foi o primeiro que a Oracle incorporou no seu sistema de banco de dados na versão 8.0. Ele é feito com base em faixa de dados de uma determinada coluna da tabela escolhida para ser a chave de particionamento para distribuir os dados para as partições. Este método como mostra a figura 3[1] é muito utilizado para distribuir os dados, como exemplo, de uma tabela de vendas por ano, trimestre e mês.

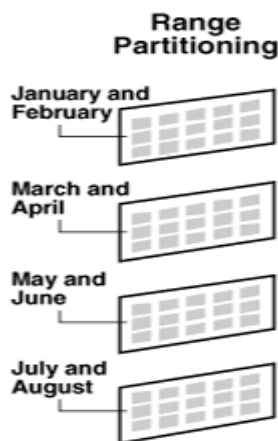


Figura 3- Particionamento por faixa
Fonte:(Documentação Oracle)

- Particionamento por Hash (Hash Partitioning)

Incorporado na versão 8.1 do Oracle o algoritmo hash é aplicado na coluna chave de particionamento da tabela para determinar a partição em que se encontra determinada linha, como por exemplo, Podem ser criadas várias partições com base na identificação do cliente e deixar o Oracle distribuir as linhas entre os espaços da tabela com base nos resultados da passagem da coluna de partição como um parâmetro de um algoritmo de hash e usar esse resultado para determinar onde armazenar a linha. Ao contrário dos métodos faixa e lista de distribuição de dados, o hash figura 4[1] não oferece nenhum mapeamento lógico entre os dados e as partições.

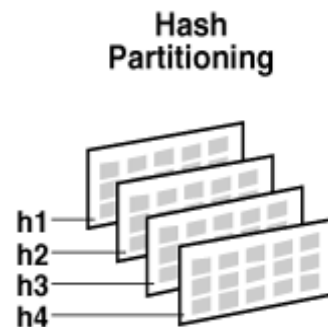


Figura 4- Particionamento por Hash
Fonte:(Documentação Oracle)

- Particionamento por Lista (List Partitioning)

Introduzido no 9i, preencheu uma lacuna que estava faltando ao possibilitar que uma tabela pudesse ser dividida com base em valores distintos, tais como códigos de estado, códigos de país ou de outra chave com valores diferentes. A distribuição de dados é definida por uma lista de valores de chave de particionamento, como por exemplo a figura 5[1], para uma coluna de regiões como chave de particionamento, a partição "East Sales Region" pode conter os valores "New York", "Virginia", "Florida" e outras que pertence a região East). Uma partição "DEFAULT" especial pode ser definida para conter todos os valores de uma chave de particionamento que não foram explicitamente definidos por nenhuma das listas.

II. REVISÃO BIBLIOGRAFICA



Figura 5 – Particionamento por lista
Fonte:(Documentação Oracle)

Estes três métodos citados acima podem ser utilizados para particionar uma tabela como tabela particionada simples ou composta:

- **No particionamento simples:** É escolhida na tabela normal uma coluna com chave de particionamento para distribuir os dados em um nível, como foi exemplificado anteriormente na tabela Pedido e representado na ilustração da figura 1.
- **No particionamento composto:** Utiliza-se a combinação de dois métodos de distribuição de dados para definir uma tabela particionada composta. A tabela é particionada pelo primeiro método de distribuição de dados, com por exemplo faixa, depois cada partição individual é subdividida em subpartições, aplicando um segundo método de distribuição de dados, por exemplo hash. A figura abaixo ilustra este conceito de particionamento.

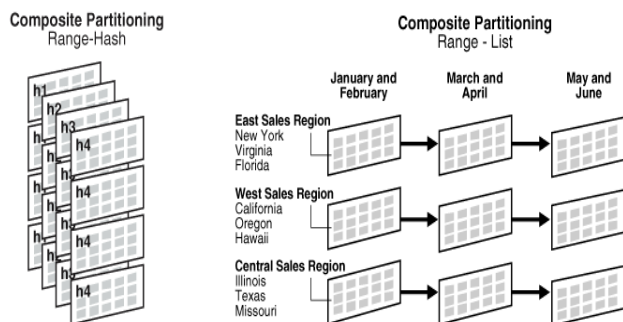


Figura 6- Particionamento composto
Fonte:(Documentação Oracle)

Entre os sistemas de banco de dados, o Oracle é o que mais explorou a técnica de particionamento de tabelas. Nas versões 10 e 11 foram incorporadas extensões ou aprimoramento das técnicas de particionamento de tabelas das versões anteriores, como, o particionamento por intervalos (Interval Partitioning) do Oracle versão 11g que é o aprimoramento da técnica por intervalo com uso do recurso automático para criar novas partições a partir do surgimento de dados novos, facilitando muito o gerenciamento da tabela particionada. É importante destacar que a chave de particionamento da tabela para o particionamento por intervalo deve ser do tipo date ou number.

a. MS SQL Server

A tecnologia de particionamento de tabela no MS SQL Server está disponível na versão 2005 Enterprise e nas versões 2008 R2: Enterprise, Datacenter, Developer e posteriores. Nesta família de Sistema de Gerenciador de Banco de dados o particionamento de tabelas está relacionado com Partition Schema, Partition Function e filegroups o conhecimento sobre estes pontos é aconselhável para melhor compreender a técnica de particionamento de tabela:

- **Filegroups:** É o primeiro a ser criado para conter as partes da tabela particionada. Não esquecendo que as partições podem ser armazenadas em um único filegroup ou para cada partição da tabela um filegroup.
- **Partition Function:** Define o número de partições que será dividida a tabela normal em partes e distribui os dados para serem armazenados nos filegroups, baseado na coluna chave de particionamento do tipo numérico ou data que deve ser marcada explicitamente como PERSISTED.
- **Partition Schema:** Faz o mapeamento físico das partições e sua localização na base de dados.

Segue abaixo, figura 7, um esquema de fluxo para criar tabelas particionadas no MS SQL Server e um código em SQL dividindo a tabela em três partes e mapeando os filegroups.

II. REVISÃO BIBLIOGRAFICA

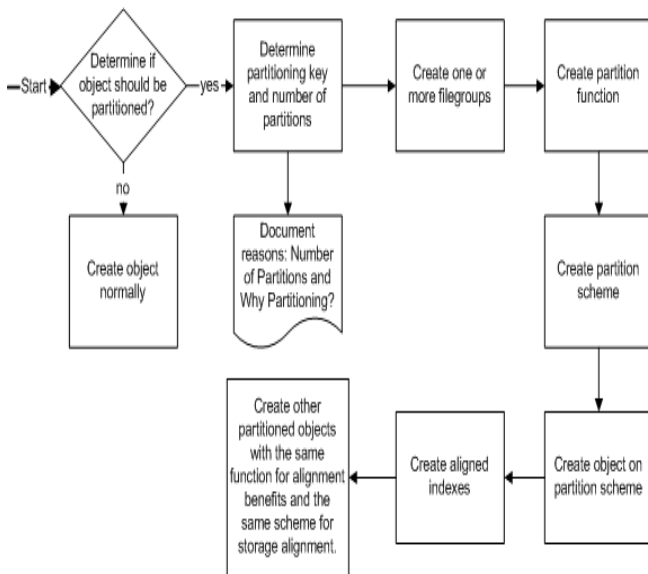


Figura 7: Etapa para criar tabela particionada
Fonte (Documentação MS SQL Server)

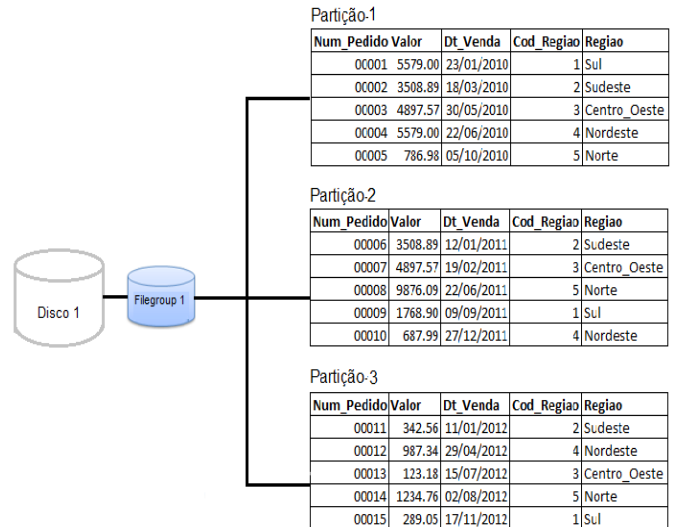


Figura 8: Armazenando o filegroup e partições em um disco

```

CREATE PARTITION FUNCTION nome_funcao (int)
AS RANGE LEFT FOR VALUES (1, 100, 1000);
GO
CREATE PARTITION SCHEME nome_esquema
AS PARTITION nome_funcao
TO (test0fg, test1fg, test2fg, test3fg);
  
```

Como citado anteriormente, no Oracle para alcançar melhor desempenho no particionamento é aconselhável colocar cada partição em tablespace distinta e se possível em disco independente. O MS SQL Server também segue este mesmo conceito, armazenando as partições em filegroups distintos e em disco diferente seria o ideal para ter melhor desempenho, as figuras 8, 9 e 10 representam estes cenários. Aplicado esta divisão em sistema de processamento em paralelo as tabelas particionadas terão acessos simultâneos, elevando mais ainda o desempenho de acesso aos dados.

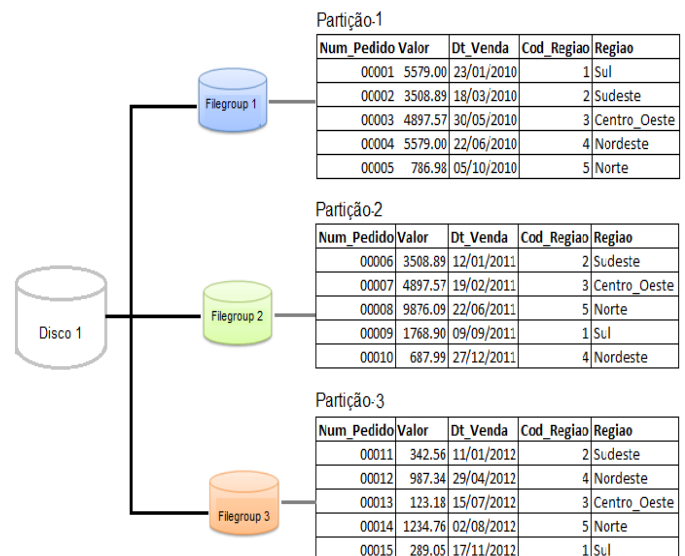


Figura 9: Armazenando os filegroups e partições em um disco

II. REVISÃO BIBLIOGRAFICA

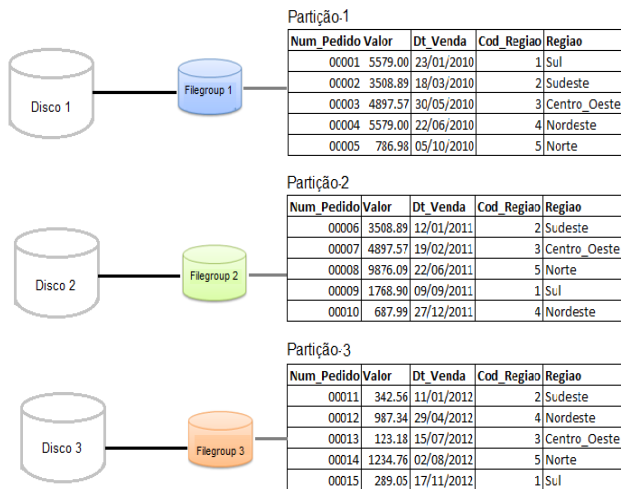


Figura 10: Armazenando filegroup e partição em cada disco

No SQL Server são dois os tipos de particionamento de tabelas:

- 1- **Particionamento Horizontal:** É a divisão da tabela normal em tabela particionada com o mesmo número de colunas, mas com menos linhas armazenadas com base na chave de partição. Por exemplo, a tabela Pedido foi dividida em três partes com seleção dos dados na coluna data da venda e armazenando os por ano em cada tabela particionada, a figura 8 retrata este cenário. Qualquer consulta ao buscar dados de um determinado ano fará referência somente à tabela apropriada. Particionar tabela de forma que as consultas façam referência ao menor número possível de tabelas é o ideal. Caso contrário, consultas com UNION em excesso, usadas para mesclar as tabelas logicamente no momento da consulta, podem afetar o desempenho da seleção dos dados.
- 2- **Particionamento Vertical:** Divide a tabela normal em várias tabelas que contêm menos colunas. Existem dois tipos de particionamento vertical a normalização e divisão de linhas:
 - **Normalização :** É o processo padrão do banco de dados para remover colunas redundantes de uma tabela e colocá-las em tabelas secundárias, vinculadas à tabela primária pela relação da chave primária e chave estrangeira.

- **O particionamento de linhas:** Divide a tabela original verticalmente em tabelas com menos colunas. Cada linha lógica em uma tabela dividida coincide com a mesma linha lógica das outras tabelas conforme é identificado por uma coluna UNIQUE KEY idêntica, em todas as tabelas particionadas. Por exemplo, a junção da linha com ID 154 de cada tabela dividida recria a linha original.

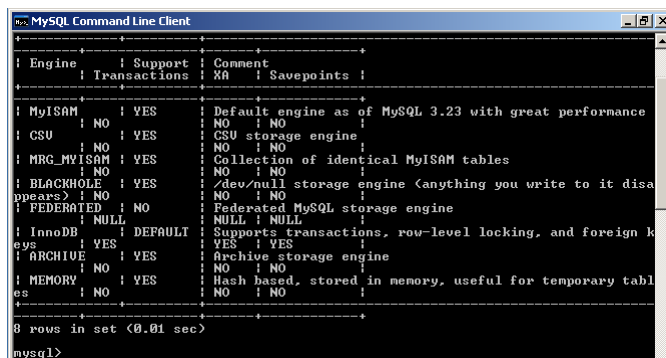
Como o particionamento horizontal, a divisão vertical deixa a consulta pesquisar menos dados. Isso aumenta o desempenho de consulta, por exemplo, uma tabela que contém quinze colunas das quais somente as nove primeiras são geralmente referenciadas, pode se beneficiar da divisão das seis últimas colunas em uma tabela separada. O particionamento vertical deve ser realizado com cautela, porque a análise de dados de várias partições requer consultas que unem as tabelas. O particionamento vertical também pode comprometer o desempenho, se as partições forem muito grandes.

Estas duas técnicas de particionamento pode serem aplicadas juntas em uma mesma tabela. É claro que deve ter muita cautela e análise da situação para adotar as duas técnicas juntas.

b. MySQL

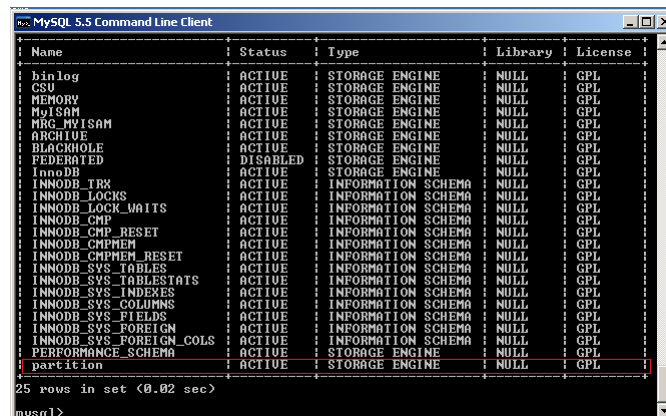
É um sistema de banco de dados de código aberto e gratuito hoje pertencendo a Oracle, apresenta quase todas as funcionalidades dos grandes bancos de dados. Foi implementado a partir da versão 5.1 o recurso de particionamento de tabelas com diferentes tipos de particionamento, possibilitando o subparticionamento e distribuindo as partes da tabela em diferentes discos. Para trabalhar com particionamento de tabelas no MySQL é importante saber se os recursos, engines, estão disponíveis e ativos na versão que estar utilizando. Abaixo a figura 11 e 12 mostra uma lista de recursos do MySQL5.5 disponíveis, usando os comandos “show engines” e “show plugins”.

II. REVISÃO BIBLIOGRAFICA



Engine	Support Transactions	Comment
MyISAM	YES	Default engine as of MySQL 3.23 with great performance
CSV	NO	NO
MRG_MYISAM	YES	Collection of identical MyISAM tables
BLACKHOLE	YES	/dev/null storage engine (anything you write to it disappears)
FEDERATED	NO	Federated MySQL storage engine
InnoDB	DEFAULT	Supports transactions, row-level locking, and foreign keys
ARCHIVE	YES	Archive storage engine
MEMORY	YES	Hash based, stored in memory, useful for temporary tables

Figura 11: Recursos (Engine) do MySQL 5.5



Name	Status	Type	Library	License
binlog	ACTIVE	STORAGE ENGINE	NONE	GPL
CSV	ACTIVE	STORAGE ENGINE	NONE	GPL
MEMORY	ACTIVE	STORAGE ENGINE	NONE	GPL
MyISAM	ACTIVE	STORAGE ENGINE	NONE	GPL
MRG_MYISAM	ACTIVE	STORAGE ENGINE	NONE	GPL
ARCHIVE	ACTIVE	STORAGE ENGINE	NONE	GPL
BLACKHOLE	ACTIVE	STORAGE ENGINE	NONE	GPL
FEDERATED	DISABLED	STORAGE ENGINE	NONE	GPL
InnoDB	ACTIVE	STORAGE ENGINE	NONE	GPL
INNODB TRX	ACTIVE	INFORMATION SCHEMA	NONE	GPL
INNODB LOCKS	ACTIVE	INFORMATION SCHEMA	NONE	GPL
INNODB LOCK_WAITS	ACTIVE	INFORMATION SCHEMA	NONE	GPL
INNODB_CMP	ACTIVE	INFORMATION SCHEMA	NONE	GPL
INNODB_CMP RESET	ACTIVE	INFORMATION SCHEMA	NONE	GPL
INNODB_CMPMEM	ACTIVE	INFORMATION SCHEMA	NONE	GPL
INNODB_CMPMEM_RESET	ACTIVE	INFORMATION SCHEMA	NONE	GPL
INNODB SYS TABLES	ACTIVE	INFORMATION SCHEMA	NONE	GPL
INNODB SYS INDEXES	ACTIVE	INFORMATION SCHEMA	NONE	GPL
INNODB SYS COLUMNS	ACTIVE	INFORMATION SCHEMA	NONE	GPL
INNODB SYS FIELDS	ACTIVE	INFORMATION SCHEMA	NONE	GPL
INNODB SYS FOREIGN COLS	ACTIVE	INFORMATION SCHEMA	NONE	GPL
PERFORMANCE_SCHEMA	ACTIVE	STORAGE ENGINE	NONE	GPL
partition	ACTIVE	STORAGE ENGINE	NONE	GPL

Figura 12: Recursos de particionamento do MySQL 5.5

Estes recursos foram implementados e incorporados no decorrer do lançamento das novas versões do Sistema de Banco de dados, possibilitando o uso de determinada técnica de particionamento. A partir da versão 5.5 do MySQL é possível determinar a coluna chave de particionamento por data ou strings, nas versões anteriores estava limitado ao particionamento da tabela apenas pela coluna do tipo de dado numérico.

As técnicas de particionamento de tabela no MySQL - Range, List, Hash, Key – são as mesmas que foram citadas anteriormente, possibilitando também o uso da combinação de duas técnicas para particionar a tabela, mas com uma estruturação de código mais simples. Abaixo segue os códigos em SQL aplicando estas técnicas na tabela pedido figura 1.

• **Particionamento por Range (faixa):** Esta técnica é usada para dividir tabelas por faixa em uma série de maneiras, dependendo da necessidade do negócio da empresa. A figura 13 abaixo, mostra o particionamento da tabela pedido em quatro partes usando a coluna de chave de particionamento

dt_venda para distribuir os dados das vendas por ano nas respectivas partições. Sendo que a cada novo ano será necessário acrescentar nova partição.

```
CREATE TABLE Pedido (
    num_pedido INT NOT NULL,
    Valor DECIMAL(8,2),
    dt_venda DATE NOT NULL,
    cod_regiao INT NOT NULL,
    regiao varchar(30) NOT NULL
)
PARTITION BY RANGE (dt_venda) (
    PARTITION p0 VALUES LESS THAN (2011-01-01),
    PARTITION p1 VALUES LESS THAN (2012-01-01),
    PARTITION p2 VALUES LESS THAN (2013-01-01),
    PARTITION p3 VALUES LESS THAN (2014-01-01)
);
```

Figura 13: Particionamento de tabela por Range

• **Particionamento por List (Lista):** Este método de particionamento é muito parecido com o Range (faixa). A principal diferença entre estes dois tipos de particionamento é que, na técnica de lista de particionamento, cada partição é definida e selecionada com base na associação de um valor da coluna de um conjunto de listas de valores, em vez de um conjunto de intervalos de valores com é feita na Range. A figura 14 abaixo, mostra que os dados armazenados na partição p1 foram selecionados com base na lista1 que possui as regiões 1 e 2, a segunda lista seleciona os dados para armazenar em p2 onde estão as regiões 3, 4 e 5.

```
CREATE TABLE Pedido (
    num_pedido INT NOT NULL,
    Valor DECIMAL(8,2),
    dt_venda DATE NOT NULL,
    cod_regiao INT NOT NULL,
    regiao varchar(30) NOT NULL
)
PARTITION BY LIST (cod_regiao) (
    PARTITION p1 VALUES IN(1,2),
    PARTITION p2 VALUES IN(3,4,5)
);
```

Figura 14: Particionamento de tabela por List

II. REVISÃO BIBLIOGRAFICA

- **Particionamento por Hash:** Esta técnica usa o algoritmo hash para distribuir os dados nas partições. É a mais simples para implementar, usando apenas duas linhas após a estrutura da tabela. A primeira linha determina a chave de particionamento para distribuir os dados nas respectivas partições e a segunda determina a quantidade de partes que será dividida à tabela. Este cenário pode ser visto na figura 15 abaixo.

```
CREATE TABLE Pedido (
    num_pedido INT NOT NULL,
    Valor DECIMAL(8,2),
    dt_venda DATE NOT NULL,
    cod_regiao INT NOT NULL,
    regiao varchar(30) NOT NULL
)
PARTITION BY HASH (cod_regiao)
PARTIÇÕES 5;
```

Figura 15: Particionamento de tabela por Hash

- **Particionamento por Key (chave):** O particionamento por chave é baseado automaticamente na chave primária da tabela normal para distribuir os dados nas partições. Se não existir chave primária, mas existir uma chave única então à chave única é usada como chave de particionamento e dever ser especificada igual à chave primária como NOT NULL. Este método também aceita a chave de particionamento string. Estes cenários estão representados abaixo na figura 16, 17 e 18.

```
CREATE TABLE Pedido (
    num_pedido INT NOT NULL,
    valor DECIMAL(8,2),
    dt_venda DATE NOT NULL,
    cod_regiao INT NOT NULL PRIMARY KEY,
    região VARCHAR(30) NOT NULL
)
PARTITION BY KEY ( )
Partições 5;
```

Figura 16: Particionamento usando a chave primária

```
CREATE TABLE Pedido (
    num_pedido INT NOT NULL,
    valor DECIMAL(8,2) NOT NULL,
    dt_venda DATE NOT NULL,
    cod_regiao INT NOT NULL,
    região VARCHAR(30) NOT NULL,
    UNIQUE KEY (cod_regiao)
)
PARTITION BY KEY ( )
Partições 5;
```

Figura 17: Particionamento usando a chave única

```
CREATE TABLE Pedido (
    num_pedido INT NOT NULL,
    valor DECIMAL(8,2) NOT NULL,
    dt_venda DATE NOT NULL,
    cod_regiao INT NOT NULL,
    região VARCHAR(30) NOT NULL,
    UNIQUE KEY (região)
)
PARTITION BY KEY (região)
Partições 5;
```

Figura 18: Particionamento usando string como chave

c. PostgreSQL

A função de particionamento de tabelas no sistema gerenciado de banco de dados PostgreSQL esta disponível a partir da versão 8.3 e para usá-la até a versão 8.4 é preciso ativar o parâmetro *constraint_exclusion* para *on*. Nas versões posteriores este parâmetro já vem ativo. O particionamento neste sistema adota o conceito de herança, a partir de uma tabela norma (padrão) denominada de tabela pai são geradas as tabelas filhas (Partições). E existem dois métodos para particionar tabelas: Range e List, os quais já foram

II. REVISÃO BIBLIOGRAFICA

demonstrados como funcionam anteriormente nas figuras 3 e 4, ilustram esta situação. A grande diferença entre os métodos de particionamento de um SGDB para outro estar na implementação dos códigos, porque cada sistema de banco de dados define o próprio código, alguns com linhas a mais que outros, como é de praxe em desenvolvimento de programas, a lógica de programação e fundamental para se ter código conciso. No PostgreSQL após criar a tabela normal e as partes é preciso definir as *constraints* para evitar que um registro seja adicionado fora de sua devida partição e depois criar uma função que direcione a inserção dos dados para as suas respectivas partições. Para maior entendimento segue abaixo exemplo.

```
CREATE TABLE Pedido(  
  num_pedido INT NOT NULL,  
  valor NUMERIC(8,2),  
  dt_venda DATE NOT NULL,  
  cod_regiao INT NOT NULL,  
  região VARCHAR(30) NOT NULL  
  CONSTRAINT Pedido_pkey PRIMARY KEY (cod_regiao)  
);  
CREATE TABLE Pedido_part_01() INHERITS(Pedido);  
CREATE TABLE Pedido_part_02() INHERITS(Pedido);  
  
ALTER TABLE Pedido_part_01 ADD CHECK (cod_regiao BETWEEN 1 AND 3);  
ALTER TABLE Pedido_part_02 ADD CHECK (cod_regiao BETWEEN 4 AND 5);  
  
CREATE OR REPLACE FUNCTION inserir_pedido()  
RETURNS TRIGGER AS $$  
BEGIN  
  IF (NEW.cod_regiao BETWEEN 1 AND 3) THEN  
    INSERT INTO Pedido_part_01 VALUES (NEW.num_pedido,  
    NEW.valor,NEW.dt_venda,NEW.cod_regiao,NEW.regiao);  
  ELSIF (NEW.cod_regiao BETWEEN 4 AND 5) THEN  
    INSERT INTO Pedido_part_02 VALUES (NEW.num_pedido,  
    NEW.valor,NEW.dt_venda,NEW.cod_regiao,NEW.regiao);  
  END IF;  
  RETURN NULL;  
END;  
$$ LANGUAGE PLPGSQL;  
  
CREATE TRIGGER Pedido_particionamento  
BEFORE INSERT ON Pedido FOR EACH  
ROW EXECUTE PROCEDURE inserir_pedido();
```

Figura 19: Particionamento de tabela no PostgreSQL

O código para implementar o particionamento no PostgreSQL é o mais extenso entre os quatro sistemas de banco de dados apresentados neste trabalho. Exigindo desenvolver função, trigger e constante, além da tabela normal. Analisando os métodos de particionamento de tabelas destes Sistemas de Gerenciamento de Banco de dados, a maneira de implementação é a grande diferença, já que as técnicas adotadas são as mesmas.

III. DESENVOLVIMENTO

1. A base de dados

O banco de dados utilizado neste trabalho pertence a um sistema de folha de pagamento com 50 mil funcionários e 10 anos de vida. Ele armazena mensalmente mais de 300 mil registros na tabela *tffinaceira* a qual recebe 60% das consultas submetida à base de dados. Esta tabela guarda a situação financeira dos funcionários após o cálculo da folha de pagamento. As principais tabelas que tem relacionamento com a tabela *tffinaceira*, a qual é objeto deste trabalho para particionamento, estão apresentadas no Modelo Relacional (MR) na figura 20 e na lista apresentada na figura 21 para melhor entendimento.

Lista de relações		
Nome	Tipo	Dono
tanotacao	tabela	postgres
tbanco	tabela	postgres
tcargo	tabela	postgres
tccusto	tabela	postgres
tdependente	tabela	postgres
tempresa	tabela	postgres
tcargo	tabela	postgres
tevento	tabela	postgres
teventocalculo	tabela	postgres
teventofuncionario	tabela	postgres
teventofuncionario_sequevento_seq	sequência	postgres
tffinaceira	tabela	postgres
tfolha	tabela	postgres
tfolhames	tabela	postgres
tfolhasuplementar	tabela	postgres
tfuncao	tabela	postgres
tfuncionario	tabela	postgres
tgrauparentesco	tabela	postgres
tlotacao	tabela	postgres
tniveiscargo	tabela	postgres
tnomeacao	tabela	postgres

Figura 21: Tabelas pertencentes à base RH

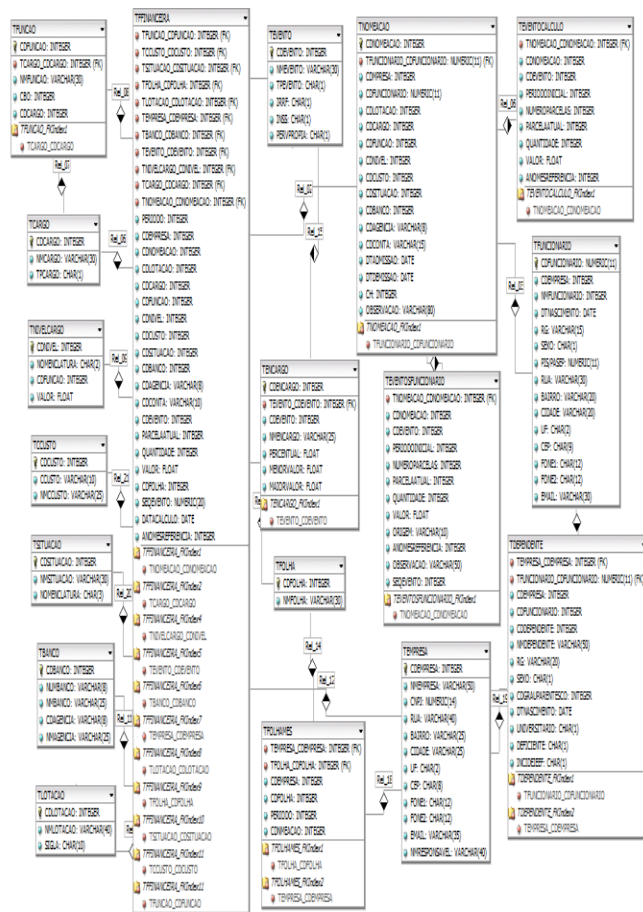


Figura 20: Modelo Relacional base folha

Neste modelo foram omitidos algumas tabelas e campos para melhor visualização, mas sem afetar o objetivo principal que é demonstrar as estruturas de relacionamentos das demais tabelas com a tabela *tffinaceira*.

2. Preparando a base de dados

A preparação da base envolvem as instalações dos programas e ativação de algumas funções necessárias para deixar a máquina destino, servidor 2, compatível, tanto em software com em hardware, com a máquina origem da base de dados RH, servidor 1. Os programas instalados na máquina servidor 2 foram: Sistema operacional Windows 2003 e Sistema de Gerenciamento de Banco de dados PostgreSQL 9.2. Após as instalações dos programas no servidor 2 foi realizado o backup da base RH no servidor 1 para ser restaurada no servidor 2. Estes procedimentos estão detalhados nas subseções abaixo.

a. Backup

Na máquina, servidor 1, com a base RH onde contém a tabela *tffinaceira* foi realizado um backup completo usando um usuário com permissão para ler e gravar os dados ou melhor permissão total. O backup foi realizado usando o prompt de comandos do sistema operacional, acessando os diretórios até chegar no diretório *bin* do PostgreSQL 9.2,

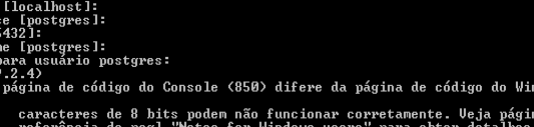
```

C:\Administrador: C:\Windows\system32\cmd.exe - pg_dump -U postgres -C -f C:\backup\RH_completo.dump
02/04/2013 00:27 68.608 pg_recovexlog.exe
02/04/2013 00:27 57.324 pg_replica.exe
02/04/2013 00:30 57.324 pg_repregress_cpq.exe
02/04/2013 00:27 65.536 pg_resetxlog.exe
02/04/2013 00:27 144.896 pg_restore.exe
02/04/2013 00:28 30.288 pg_standby.exe
02/04/2013 00:29 39.936 pg_test_fsync.exe
02/04/2013 00:29 20.480 pg_test_fining.exe
02/04/2013 00:29 97.280 pg_upgrade.exe
02/04/2013 00:27 4.525.568 postgres.exe
02/04/2013 00:27 352.256 psql.exe
02/04/2013 00:28 50.368 reindexdb.exe
01/08/2012 00:01 265.216 sslskey32.dll
02/04/2013 00:36 1.640.720 stackbuilder.exe
02/04/2013 00:28 40.448 vacuumdb.exe
02/04/2013 00:29 25.088 vacuumd.exe
22/05/2012 04:46 120.320 wxbase28u_net_vc_custom.dll
22/05/2012 04:43 1.154.048 wxbase28u_vc_custom.dll
22/05/2012 04:46 118.272 wxbase28u_xml_vc_custom.dll
22/05/2012 04:45 683.520 wxmsw28u_adv_vc_custom.dll
22/05/2012 04:45 316.416 wxmsw28u_gui_vc_custom.dll
22/05/2012 04:44 2.817.536 wxmsw28u_core_vc_custom.dll
22/05/2012 04:46 475.648 wxmsw28u_html_vc_custom.dll
22/05/2012 04:50 526.336 wxmsw28u_stc_vc_custom.dll
22/05/2012 04:47 494.880 wxmsw28u_xrc_vc_custom.dll
02/04/2013 00:29 60.928 zic.exe
10/05/2012 11:52 66.048 zlib.dll
54 arquivo(s) 28.070.072 bytes
2 pasta(s) 162.948.268.832 bytes disponíveis

C:\PostgreSQL\9.2\bin>pg_dump -U postgres -C -f C:\backup\RH_completo.dump RH
Senha:

```

b. Restore

[illegible]

```
SQL Shell (psql)

Server [localhost]:
Database [postgres]:
Port [5432]:
Username [postgres]:
Senha para usuário postgres:
psql (9.2.4)
AUI50: página de código do Console (850) difere da página de código do Windows (
1252)
caracteres de 8 bits podem não funcionar corretamente. Veja página de
referência do psql "Notes for Windows users" para obter detalhes.
Digite "help" para ajuda.

postgres=# \c RH
AUI50: página de código do Console (850) difere da página de código do Windows (
1252)
caracteres de 8 bits podem não funcionar corretamente. Veja página de
referência do psql "Notes for Windows users" para obter detalhes.
Você está conectado agora ao banco de dados "RH" como usuário "postgres".
RH=# select into tfinancaira_temp from tfinancaira;
ERROR: column tfinancaira does not exist
RH=#
```

```
SQL Shell (psql)
```

```
postgres=# \c RH
RH=# \i /usr/share/postgresql/9.5/doc/man/pgtutorial.sgml
NOTICE: página de código do Console (850) difere da página de código do Windows (
1252)
        caracteres de 8 bits podem não funcionar corretamente. Veja página de
referência do psql "Notes for Windows users" para obter detalhes.
Você está conectado agora ao banco de dados "RH" como usuário "postgres".
RH=# select * into tffinanceira_temp from tffinanceira;
SELECT 4533294
RH=# truncate tffinanceira;
ERROR: erro de sintaxe em ou próximo a "truncate"
LINE 1: 1: truncate tffinanceira;
          ^
RH=# truncate tffinanceira;
TRUNCATE TABLE
RH=# select * from tffinanceira;
    periodo | cdenpresa | cdnomeacao | cdlotacao | cdcargo | cdfuncao | cdniuel | c
dcusto | cdsituacao | cdbanco | cdagencia | cdconta | cdevento | parcelaatual |
quantidade | valor | cdfolha | sequevto | datacalcul | anomereserencia
```

periodo	cdenpresa	cdnomeacao	cdlotacao	cdcargo	cdfuncao	cdniuel	c
dcusto	cdsituacao	cdbanco	cdagencia	cdconta	cdevento	parcela	atual
quantidade	valor	cdfolha	sequevto	datacalcul	anomesrencia		

```
(0 registro)
```

```
RH=#
```



III. DESENVOLVIMENTO

e. Particionando a tabela tffinanceira

Na tabela tffinanceira a coluna datacalculado foi definida como coluna chave de particionamento. Esta coluna armazena a data (dia, mês e ano) do processamento da folha de pagamento. Portanto, a tabela foi particionada por mês, ficando dividida em 13 partes de janeiro a dezembro mais uma partição para décimo terceiro. Elas armazenam os dados de cada mês em suas respectivas partições usando o critério de seleção na coluna chave de particionamento no campo mês combinado com o tipo de folha para determinar se é folha de pagamento mensal ou décimo terceiro. Na tabela tfolha estão os tipos de folhas que são: Folha 1 para mensal, folha 2 para complementar e 3 para décimo. A folha complementar é uma folha complementar da folha mensal realizada quando ocorre um fato inesperado. Abaixo segue os detalhes do particionamento da tabela tffinanceira.

- O código abaixo mostra a estrutura da tabela tffinanceira que existe na base de dados RH, o processo de limpeza só apagou os dados, não necessitando recriá-la.

```
CREATE TABLE tffinanceira
(
    periodo integer NOT NULL,
    cdempresa integer NOT NULL,
    cdnomeacao integer NOT NULL,
    cdlotacao integer NOT NULL,
    cdcargo integer NOT NULL,
    cdfuncao integer NOT NULL,
    cdnivel integer NOT NULL,
    cdcusto integer NOT NULL,
    cdsituacao integer NOT NULL,
    cdbanco integer NOT NULL,
    cdagencia character varying(8) NOT NULL,
    cdconta character varying(10) NOT NULL,
    cdevento integer NOT NULL,
    parcelaatual integer NOT NULL,
    quantidade integer NOT NULL,
    valor double precision NOT NULL,
    cdfolha integer NOT NULL,
    sequevento numeric,
    datacalculado date,
    anomesreferencia integer,
    CONSTRAINT tffinanceira_cdbanco_fkey FOREIGN KEY (cdbanco)
        REFERENCES tbanco (cdbanco) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION,
    CONSTRAINT tffinanceira_cdcargo_fkey FOREIGN KEY (cdcargo)
        REFERENCES tcargo (cdcargo) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION,
    CONSTRAINT tffinanceira_cdcusto_fkey FOREIGN KEY (cdcusto)
        REFERENCES tcusto (cdcusto) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION,
    CONSTRAINT tffinanceira_cdevento_fkey FOREIGN KEY (cdevento)
        REFERENCES tevento (cdevento) MATCH SIMPLE
        ON UPDATE NO ACTION ON DELETE NO ACTION,
)
```

```
ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT tffinanceira_cdfolha_fkey FOREIGN KEY (cdfolha)
    REFERENCES tfolha (cdfolha) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT tffinanceira_cdfuncao_fkey FOREIGN KEY (cdfuncao)
    REFERENCES tfuncao (cdfuncao) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT tffinanceira_cdlotacao_fkey FOREIGN KEY (cdlotacao)
    REFERENCES tlotacao (cdlotacao) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT tffinanceira_cdnivel_fkey FOREIGN KEY (cdnivel)
    REFERENCES tniveiscargo (cdnivel) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE NO ACTION,
CONSTRAINT tffinanceira_cdnomeacao_fkey FOREIGN KEY (cdnomeacao)
    REFERENCES tnomeacao (cdnomeacao) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE RESTRICT,
CONSTRAINT tffinanceira_cdsituacao_fkey FOREIGN KEY (cdsituacao)
    REFERENCES tsituacao (cdsituacao) MATCH SIMPLE
    ON UPDATE NO ACTION ON DELETE NO ACTION
)
WITH (
    OIDS=FALSE
```

Figura 26: Estrutura da tabela tffinanceira.

- O código da figura 27 cria 13 partições na tabela tffinanceira onde são armazenados os dados nas respectivas partições.

```
CREATE TABLE janeiro() INHERITS(tffinanceira);
CREATE TABLE fevereiro() INHERITS(tffinanceira);
CREATE TABLE marco() INHERITS(tffinanceira);
CREATE TABLE abril() INHERITS(tffinanceira);
CREATE TABLE maio() INHERITS(tffinanceira);
CREATE TABLE junho() INHERITS(tffinanceira);
CREATE TABLE julho() INHERITS(tffinanceira);
CREATE TABLE agosto() INHERITS(tffinanceira);
CREATE TABLE setembro() INHERITS(tffinanceira);
CREATE TABLE outubro() INHERITS(tffinanceira);
CREATE TABLE novembro() INHERITS(tffinanceira);
CREATE TABLE dezembro() INHERITS(tffinanceira);
CREATE TABLE decimo() INHERITS(tffinanceira);
```

Figura 27: Criando partições.

- Para criar as regras de distribuições dos dados para as partições foi definida a coluna datacalculado como chave de particionamento. Nesta chave foi usada a função *date_part* para selecionar o mês (month) da data de processamento da folha de pagamento na tabela tffinanceira. A função distribui os dados nas respectivas

III. DESENVOLVIMENTO

partições baseado no número do mês, em destaque para dezembro e décimo que usam o mês 12 e o tipo de folha para selecionar os dados. Neste particionamento o décimo é pago em uma única parcela no mês de dezembro, mas poderia ser duas parcelas uma no mês de junho e a outra em dezembro, estes detalhes dependem muito da regra de negócio da empresa.

A implementação do código deve ser para todas as partições. Para simplificar a figura 28 mostra o código para os meses de janeiro, dezembro e décimo.

```
CREATE OR REPLACE RULE insert_janeiro AS
ON INSERT TO tffinanceira
WHERE date_part ('month', new.datacalcula)=01
DO INSTEAD INSERT INTO janeiro (periodo,cdempresa,cdnomeacao,cdlotacao,
cdcargo,cdfuncao,cdnivel,cdcusto,cdsituacao,cdbanco,cdagencia,cdconta,
cdevento,parcelaactual,quantidade,valor,cdfolha,segevento,datacalcula)
VALUES (new.periodo, new.cdempresa, new.cdnomeacao, new.cdlotacao,
new.cdcargo, new.cdfuncao, new.cdnivel,new.cdcusto, new.cdsituacao,
new.cdbanco, new.cdagencia, new.cdconta, new.cdevento, new.parcelaactual,
new.quantidade, new.valor, new.cdfolha,new.segevento,new.datacalcula);

.....

CREATE OR REPLACE RULE insert_dezembro AS
ON INSERT TO tffinanceira
WHERE date_part ('month', new.datacalcula)=12 and new.cdfolha <> 3
DO INSTEAD INSERT INTO dezembro (periodo,cdempresa,cdnomeacao,cdlotacao,
cdcargo,cdfuncao,cdnivel,cdcusto,cdsituacao,cdbanco,cdagencia,cdconta,
cdevento,parcelaactual,quantidade,valor,cdfolha,segevento,datacalcula)
VALUES (new.periodo, new.cdempresa, new.cdnomeacao, new.cdlotacao,
new.cdcargo, new.cdfuncao, new.cdnivel,new.cdcusto, new.cdsituacao,
new.cdbanco, new.cdagencia, new.cdconta, new.cdevento, new.parcelaactual,
new.quantidade, new.valor, new.cdfolha,new.segevento,new.datacalcula);

CREATE OR REPLACE RULE insert_decimo AS
ON INSERT TO tffinanceira
WHERE date_part ('month', new.datacalcula)=12 and new.cdfolha=3
DO INSTEAD INSERT INTO decimo (periodo,cdempresa,cdnomeacao,cdlotacao,
cdcargo,cdfuncao,cdnivel,cdcusto,cdsituacao,cdbanco,cdagencia,cdconta,
cdevento,parcelaactual,quantidade,valor,cdfolha,segevento,datacalcula)
VALUES (new.periodo, new.cdempresa, new.cdnomeacao, new.cdlotacao,
new.cdcargo, new.cdfuncao, new.cdnivel,new.cdcusto, new.cdsituacao,
new.cdbanco, new.cdagencia, new.cdconta, new.cdevento, new.parcelaactual,
new.quantidade, new.valor, new.cdfolha,new.segevento,new.datacalcula);
```

Figura 28: Código de regra de distribuição dos dados.

Após a implementação do código de regras de distribuição dos dados é possível visualizar as regras usando o pgAdmin como mostra a figura 29.

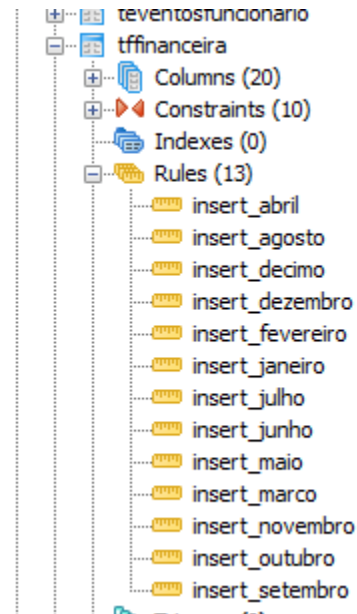


Figura 29: Regras de distribuição.

- A última etapa deste processo é a importação dos dados da tabela temporária para a tabela particionada usando os comandos *select* e *insert into*, como demonstrado abaixo na figura 30.

```
INSERT INTO tffinanceira (periodo,cdempresa,cdnomeacao,cdlotacao,cdcargo,
cdfuncao,cdnivel,cdcusto,cdsituacao,cdbanco,cdagencia,cdconta,cdevento,
parcelaactual,quantidade,valor,cdfolha,segevento,datacalcula)
select periodo,cdempresa,cdnomeacao,cdlotacao,cdcargo,cdfuncao,cdnivel,
cdcusto,cdsituacao,cdbanco,cdagencia,cdconta,cdevento,parcelaactual,
quantidade,valor,cdfolha,segevento,datacalcula
from tffinanceira1
```

Figura 30: Inserção dos dados.

A inserção dos dados na tabela *tffinanceira* é a parte que requer mais tempo em todo o processo. Isto é claro depende muito da configuração do equipamento e tempo dedicado para realizar a tarefa. A formatação dos dados quando forem importados requer atenção, principalmente os que possuem casas decimais. Após a inserção serão executados testes de consultas nas tabelas para medir os desempenhos. Estes testes serão demonstrados na seção resultados e análise.

IV. RESULTADOS E ANÁLISE

1. Teste de consulta nas tabelas

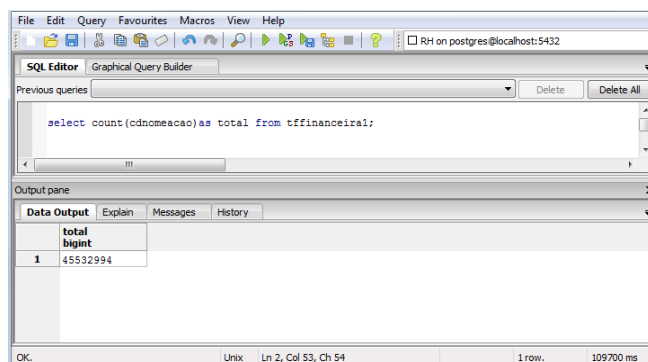
Nesta seção são demonstrados os resultados e análises dos testes de desempenho nas tabelas *tffinanceira* com partição e *tffinanceira1* sem partição, ambas contêm os mesmos registros como mostra as figuras 31 e 32 abaixo. Estes testes são realizados direto na máquina servidor sem usuários conectados ao banco, com exceção do usuário que irá efetuar os testes, para afastar possíveis interferências externa ao Sistema Gerenciador de Banco de dados, como exemplo, problema de cabeamento, grande volume de dados na rede no momento da consulta, entre outras. Assim, dando maior precisão nos resultados dos testes. Portanto, o maior objetivo deste trabalho é analisar a performance das consultas submetidas nessas duas tabelas que possuem dados históricos. Quando a base de dados for um ambiente de teste não descarta testar o desempenho destas tabelas nas operações de inserção, atualização e remoção. Destacando que estes comandos tem um impacto muito semelhante porque alteram os dados na tabela, oposto a consulta que só pesquisa os dados.

a. Comando *select*

Para os testes que estão demonstrados abaixo foi utilizado o comando *select* para realizar pesquisas de dados de diversas maneiras. Para obter os resultados dos testes aplicados nas tabelas *tffinanceira* com partição e na *tffinanceira1* sem partição foi aplicado o mesmo código de consulta, primeiro na tabela sem partição depois na tabela com partição e por último na partição individualizada nos casos que coube a consulta. Sabendo que cada partição individualizada representa 1/13 da tabela principal - seção desenvolvimento figura 27 – é dedutível sem sombra de dúvida que o resultado da pesquisa é bem superior que o da tabela principal (mãe).

• Teste 1

É um teste simples de contagem de registro aplicado nas tabelas *tffinanceira1*, sem partição e *tffinanceira*, com partição, demonstrados nas figuras 31 e 32. A pesquisa executada na tabela com partição teve melhor performance com uma diferença de 6,397 ms, equivalente a 6% mais rápida que o resultado da pesquisa na tabela *tffinanceira1*.



The screenshot shows a SQL Editor window with the following content:

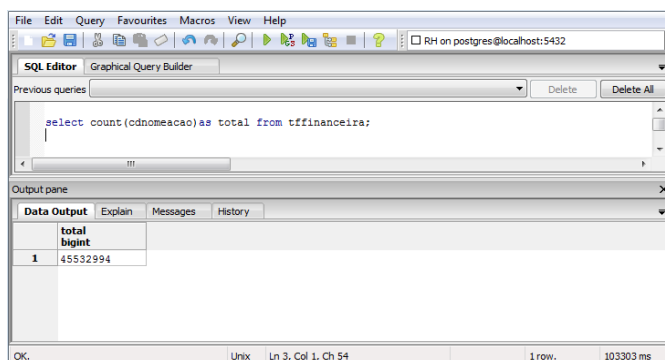
```
select count(odnomeacao) as total from tffinanceira1;
```

The Output pane displays the result:

total
bigint
1 45532994

At the bottom, it indicates "1 row. 109700 ms".

Figura 31: Quantidade de registro na tabela não particionada



The screenshot shows a SQL Editor window with the following content:

```
select count(odnomeacao) as total from tffinanceira;
```

The Output pane displays the result:

total
bigint
1 45532994

At the bottom, it indicates "1 row. 103303 ms".

Figura 32: Quantidade de registro na tabela particionada

• Teste 2

Este teste pesquisa os dados do mês de janeiro do ano de 2008 de um funcionário específico. Analisando as pesquisas, a que teve melhor desempenho foi a consulta da tabela particionada, tempo 84.708 ms, com desempenho de aproximadamente 36% superior a tabela sem partição com 115.737 ms, como mostra as figuras 33 e 34. Esta mesma pesquisa foi realizada diretamente na partição janeiro que contém os dados pesquisados e teve o tempo de 6,411 ms, figura 35. O resultado desta pesquisa comparado com os resultados realizados nas tabelas sem partição e com partição, consecutivamente, foi superior em 18 e 13 vezes. Demonstrando a grande vantagem do particionamento de tabela para pesquisa de grande volume de dados.

IV. RESULTADOS E ANÁLISE

Query:

```
select a.periodo,a.cdnomeacao,c.nmfuncionario,a.cdevento,d.nmevento,d.tpevento,a.valor
from tffinancieira a
left join tnomeacao b on b.cdnomeacao=a.cdnomeacao
left join tfuncionario c on c.cdfuncionario=b.cdfuncionario
left join tevento d on d.cdevento=a.cdevento
where a.periodo='200801' and a.cdnomeacao='23465837'
order by d.tpevento desc
```

Output pane:

	periodo	cdnomeacao	nmfuncionario	cdevento	nmevento	tpevento	valor
	integer	integer	character varying(50)	integer	character varying(40)	character(1)	double precision
1	200801	23475482	MARIA ANTONIA DA COSTA	1	VENCIMENTO EFETIVO	P	3050.22
2	200801	23475482	MARIA ANTONIA DA COSTA	100	FERIAS	P	1292.33
3	200801	23475482	MARIA ANTONIA DA COSTA	101	HORA NOTURNA	P	387.70
4	200801	23475482	MARIA ANTONIA DA COSTA	106	EMPRESTIMO BB	D	486.32
5	200801	23475482	MARIA ANTONIA DA COSTA	108	IPER	D	457.53
6	200801	23475482	MARIA ANTONIA DA COSTA	109	IRRF	D	457.53
7	200801	23475482	MARIA ANTONIA DA COSTA	110	EMPRESTIMO CEF	D	486.32

8 rows. 115737 ms

Figura 33: Pesquisa na tabela sem partição

Query:

```
select a.periodo,a.cdnomeacao,c.nmfuncionario,a.cdevento,d.nmevento,d.tpevento,a.valor
from tffinancieira a
left join tnomeacao b on b.cdnomeacao=a.cdnomeacao
left join tfuncionario c on c.cdfuncionario=b.cdfuncionario
left join tevento d on d.cdevento=a.cdevento
where a.periodo='200801' and a.cdnomeacao='23465837'
order by d.tpevento desc
```

Output pane:

	periodo	cdnomeacao	nmfuncionario	cdevento	nmevento	tpevento	valor
	integer	integer	character varying(50)	integer	character varying(40)	character(1)	double precision
1	200801	23475482	MARIA ANTONIA DA COSTA	1	VENCIMENTO EFETIVO	P	3050.22
2	200801	23475482	MARIA ANTONIA DA COSTA	100	FERIAS	P	1292.33
3	200801	23475482	MARIA ANTONIA DA COSTA	101	HORA NOTURNA	P	387.70
4	200801	23475482	MARIA ANTONIA DA COSTA	106	EMPRESTIMO BB	D	486.32
5	200801	23475482	MARIA ANTONIA DA COSTA	108	IPER	D	457.53
6	200801	23475482	MARIA ANTONIA DA COSTA	109	IRRF	D	457.53
7	200801	23475482	MARIA ANTONIA DA COSTA	110	EMPRESTIMO CEF	D	486.32

8 rows. 84708 ms

Figura 34: Pesquisa na tabela com partição

Query:

```
select a.periodo,a.cdnomeacao,c.nmfuncionario,a.cdevento,d.nmevento,d.tpevento,a.valor
from janeiro a
left join tnomeacao b on b.cdnomeacao=a.cdnomeacao
left join tfuncionario c on c.cdfuncionario=b.cdfuncionario
left join tevento d on d.cdevento=a.cdevento
where a.periodo='200801' and a.cdnomeacao='23465837'
order by d.tpevento desc
```

Output pane:

	periodo	cdnomeacao	nmfuncionario	cdevento	nmevento	tpevento	valor
	integer	integer	character varying(50)	integer	character varying(40)	character(1)	double precision
1	200801	23475482	MARIA ANTONIA DA COSTA	1	VENCIMENTO EFETIVO	P	3050.22
2	200801	23475482	MARIA ANTONIA DA COSTA	100	FERIAS	P	1292.33
3	200801	23475482	MARIA ANTONIA DA COSTA	101	HORA NOTURNA	P	387.70
4	200801	23475482	MARIA ANTONIA DA COSTA	106	EMPRESTIMO BB	D	486.32
5	200801	23475482	MARIA ANTONIA DA COSTA	108	IPER	D	457.53
6	200801	23475482	MARIA ANTONIA DA COSTA	109	IRRF	D	457.53
7	200801	23475482	MARIA ANTONIA DA COSTA	110	EMPRESTIMO CEF	D	486.32

8 rows. 6411 ms

Figura 35: Pesquisa diretamente na partição

- Teste 3

Neste teste foram realizadas as somatórias dos proventos e dos descontos do mês de novembro de 2010 e subtraindo dos proventos os descontos, como mostram as figuras 36, 37 e 38. Comparando estes resultados é inquestionável que a consulta aplicada diretamente na partição onde estão os dados foi a que teve o melhor desempenho atingindo o tempo 20 vezes melhor que as outras duas consultas realizadas nas tabelas sem partição e com partição. A pesquisa executada na tabela sem partição, tempo de 112,226 ms, teve uma leve vantagem sobre a pesquisa da tabela com partição, tempo de 118,467. A diferença entre os tempos destas duas tabelas é de 6,241ms, maior que o tempo da pesquisa executada diretamente na partição, 5,102 ms.

Query:

```
select a.periodo,a.cdnomeacao,c.nmfuncionario,
sum(case when d.tpevento='P' then a.valor else 0 end) as PROVENTO,
sum(case when d.tpevento='D' then a.valor else 0 end) as DESCONTO,
sum(case when d.tpevento='P' then a.valor else 0 end)-
sum(case when d.tpevento='D' then a.valor else 0 end) as SLiquido
from tffinancieira a
left join tnomeacao b on b.cdnomeacao=a.cdnomeacao
left join tfuncionario c on c.cdfuncionario=b.cdfuncionario
left join tevento d on d.cdevento=a.cdevento
where a.periodo='201011'
group by a.periodo,a.cdnomeacao,c.nmfuncionario
```

Output pane:

	periodo	cdnomeacao	nmfuncionario	provento	desconto	sliquido
	integer	integer	character varying(50)	double precision	double precision	double precision
1	201011	23457651	JACKSON GRIGORIO SILVA	5793.47	2596.61	3196.85
2	201011	23457652	MARINA FONSECA RAMOS	4157.47	1887.70	2269.77
3	201011	23457653	TEREZINHA SILVA LIMA	3922.54	1887.70	2034.83
4	201011	23457654	VALDETE EDUARDO ALVES	4456.47	2252.04	2204.43
5	201011	23457655	JOSE ALBERTO CAVALCANTI MEDEIROS	5226.33	2596.61	2629.71
6	201011	23457656	NEUDIRAN DE MORAIS NASCIMENTO	5793.47	2596.61	3196.85
7	201011	23457657	CLEMILDA MAGALHAES PINHEIRO	2228.69	1081.14	1147.54

53286 rows. 112226 ms

Figura 36: Pesquisa na tabela sem partição

Query:

```
select a.periodo,a.cdnomeacao,c.nmfuncionario,
sum(case when d.tpevento='P' then a.valor else 0 end) as PROVENTO,
sum(case when d.tpevento='D' then a.valor else 0 end) as DESCONTO,
sum(case when d.tpevento='P' then a.valor else 0 end)-
sum(case when d.tpevento='D' then a.valor else 0 end) as SLiquido
from tffinancieira a
left join tnomeacao b on b.cdnomeacao=a.cdnomeacao
left join tfuncionario c on c.cdfuncionario=b.cdfuncionario
left join tevento d on d.cdevento=a.cdevento
where a.periodo='201011'
group by a.periodo,a.cdnomeacao,c.nmfuncionario
```

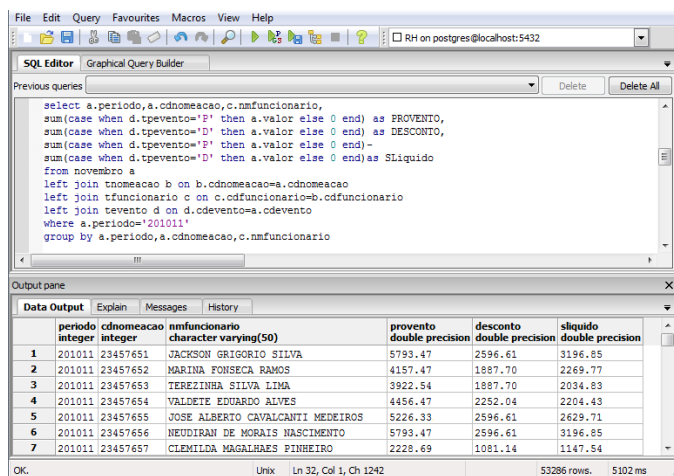
Output pane:

	periodo	cdnomeacao	nmfuncionario	provento	desconto	sliquido
	integer	integer	character varying(50)	double precision	double precision	double precision
1	201011	23457651	JACKSON GRIGORIO SILVA	5793.47	2596.61	3196.85
2	201011	23457652	MARINA FONSECA RAMOS	4157.47	1887.70	2269.77
3	201011	23457653	TEREZINHA SILVA LIMA	3922.54	1887.70	2034.83
4	201011	23457654	VALDETE EDUARDO ALVES	4456.47	2252.04	2204.43
5	201011	23457655	JOSE ALBERTO CAVALCANTI MEDEIROS	5226.33	2596.61	2629.71
6	201011	23457656	NEUDIRAN DE MORAIS NASCIMENTO	5793.47	2596.61	3196.85
7	201011	23457657	CLEMILDA MAGALHAES PINHEIRO	2228.69	1081.14	1147.54

53286 rows. 118467 ms

Figura 37: Pesquisa na tabela com partição

IV. RESULTADOS E ANÁLISE



SQL Editor: Previous queries

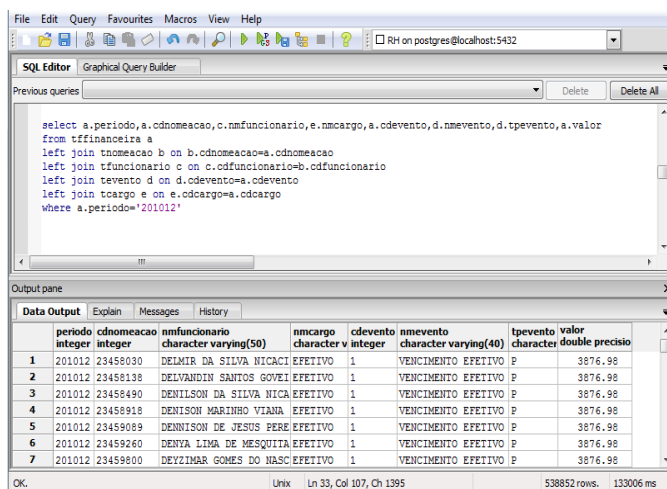
```
select a.periodo,a.cdnomeacao,c.nmfuncionario,
sum(case when d.tpevento='P' then a.valor else 0 end) as PROVENTO,
sum(case when d.tpevento='D' then a.valor else 0 end) as DESCONTO,
sum(case when d.tpevento='D' then a.valor else 0 end) as SLIQUIDO
from novembro a
left join tnomeacao b on b.cdnomeacao=a.cdnomeacao
left join tfuncionario c on c.cdfuncionario=b.cdfuncionario
left join tevento d on d.cdevento=a.cdevento
where a.periodo='201011'
group by a.periodo,a.cdnomeacao,c.nmfuncionario
```

Output pane: Data Output

	periodo	cdnomeacao	nmfuncionario	provento	desconto	sliquido
	integer	integer	character varying(50)	double precision	double precision	double precision
1	201011	23457651	JACKSON GRIGORIO SILVA	5793.47	2596.61	3196.85
2	201011	23457652	MARINA FONSECA RAMOS	4157.47	1987.70	2269.77
3	201011	23457653	TEREZINHA SILVA LIMA	3922.54	1987.70	2034.83
4	201011	23457654	VALDETE EDUARDO ALVES	4456.47	2252.04	2204.43
5	201011	23457655	JOSE ALBERTO CAVALCANTI MEDEIROS	5226.33	2596.61	2629.71
6	201011	23457656	NEUDIRAN DE MORAIS NASCIMENTO	5793.47	2596.61	3196.85
7	201011	23457657	CLEMILDA MAGALHAES PINHEIRO	2228.69	1081.14	1147.54

OK. Unix Ln 32, Col 1, Ch 1242 53286 rows. 5102 ms

Figura 38: Pesquisa diretamente na partição



SQL Editor: Previous queries

```
select a.periodo,a.cdnomeacao,c.nmfuncionario,e.nmcargo,a.cdevento,d.nmevento,a.valor
from tffinanceira a
left join tnomeacao b on b.cdnomeacao=a.cdnomeacao
left join tfuncionario c on c.cdfuncionario=b.cdfuncionario
left join tevento d on d.cdevento=a.cdevento
left join toargo e on e.cdcargo=a.cdcargo
where a.periodo='201012'
```

Output pane: Data Output

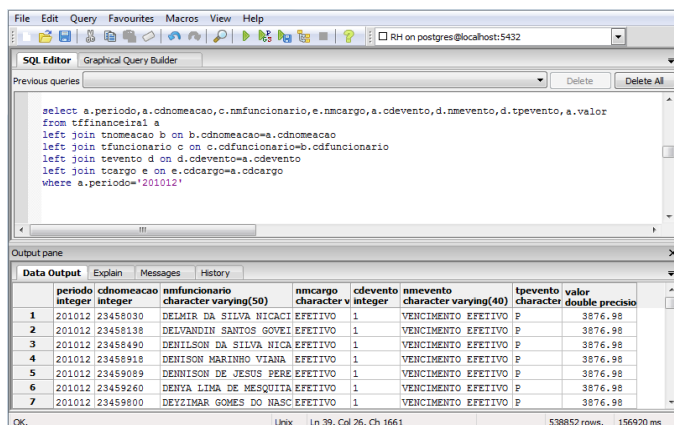
	periodo	cdnomeacao	nmfuncionario	nmcargo	cdevento	nmevento	tpevento	valor
	integer	integer	character varying(50)	character v	integer	character varying(40)	character	double precisio
1	201012	23458030	DELMIR DA SILVA NICACI	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
2	201012	23458138	DELVANDIN SANTOS GOVEI	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
3	201012	23458490	DENILSON DA SILVA NICA	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
4	201012	23458918	DENISON MARINHO VIANA	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
5	201012	23459089	DENNISON DE JESUS PERE	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
6	201012	23459260	DENYA LIMA DE MESQUITA	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
7	201012	23459800	DEYZIMAR GOMES DO NASC	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98

OK. Unix Ln 33, Col 107, Ch 1395 538852 rows. 133006 ms

Figura 40: Pesquisa na tabela com partição

• Teste 4

Foi aplicado para este teste uma consulta nos dados do mês de dezembro do ano de 2010 com um total de 538.852 registros. A pesquisa na tabela tffinanceira, com partição, teve o tempo de 133,006 ms comprando com o tempo de 156,920 ms da pesquisa na tabela tffinanceira1, sem partição. A consulta na tabela com partição foi mais rápida com 23,914 ms de diferença, tendo uma vantagem na performance de 17%. Esta mesma pesquisa aplicada diretamente na partição dezembro, onde estão armazenados os dados, teve um tempo de 12,308 ms, sendo 10 vezes melhor que os tempos das tabelas tffinanceira e tffinanceira1, como mostram as figuras 39, 40 e 41 abaixo.



SQL Editor: Previous queries

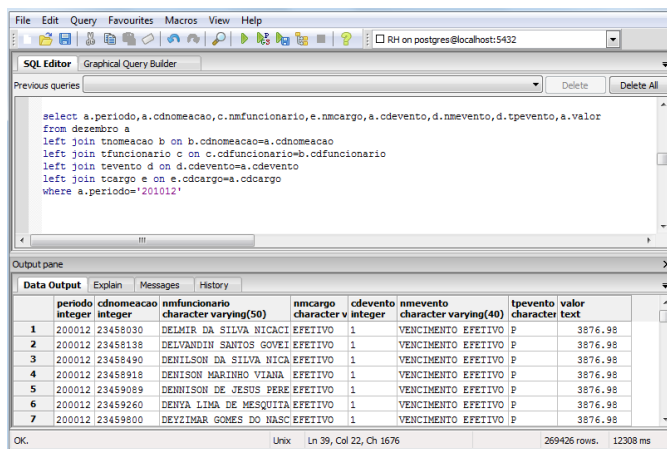
```
select a.periodo,a.cdnomeacao,c.nmfuncionario,e.nmcargo,a.cdevento,d.nmevento,a.valor
from tffinanceira1 a
left join tnomeacao b on b.cdnomeacao=a.cdnomeacao
left join tfuncionario c on c.cdfuncionario=b.cdfuncionario
left join tevento d on d.cdevento=a.cdevento
left join toargo e on e.cdcargo=a.cdcargo
where a.periodo='201012'
```

Output pane: Data Output

	periodo	cdnomeacao	nmfuncionario	nmcargo	cdevento	nmevento	tpevento	valor
	integer	integer	character varying(50)	character v	integer	character varying(40)	character	double precisio
1	201012	23458030	DELMIR DA SILVA NICACI	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
2	201012	23458138	DELVANDIN SANTOS GOVEI	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
3	201012	23458490	DENILSON DA SILVA NICA	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
4	201012	23458918	DENISON MARINHO VIANA	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
5	201012	23459089	DENNISON DE JESUS PERE	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
6	201012	23459260	DENYA LIMA DE MESQUITA	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
7	201012	23459800	DEYZIMAR GOMES DO NASC	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98

OK. Unix Ln 39, Col 26, Ch 1661 538852 rows. 156920 ms

Figura 39: Pesquisa na tabela sem partição



SQL Editor: Previous queries

```
select a.periodo,a.cdnomeacao,c.nmfuncionario,e.nmcargo,a.cdevento,d.nmevento,a.valor
from dezembro a
left join tnomeacao b on b.cdnomeacao=a.cdnomeacao
left join tfuncionario c on c.cdfuncionario=b.cdfuncionario
left join tevento d on d.cdevento=a.cdevento
left join toargo e on e.cdcargo=a.cdcargo
where a.periodo='201012'
```

Output pane: Data Output

	periodo	cdnomeacao	nmfuncionario	nmcargo	cdevento	nmevento	tpevento	valor
	integer	integer	character varying(50)	character v	integer	character varying(40)	character	double precisio
1	201012	23458030	DELMIR DA SILVA NICACI	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
2	201012	23458138	DELVANDIN SANTOS GOVEI	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
3	201012	23458490	DENILSON DA SILVA NICA	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
4	201012	23458918	DENISON MARINHO VIANA	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
5	201012	23459089	DENNISON DE JESUS PERE	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
6	201012	23459260	DENYA LIMA DE MESQUITA	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98
7	201012	23459800	DEYZIMAR GOMES DO NASC	EFETIVO	1	VENCIMENTO EFETIVO	P	3876.98

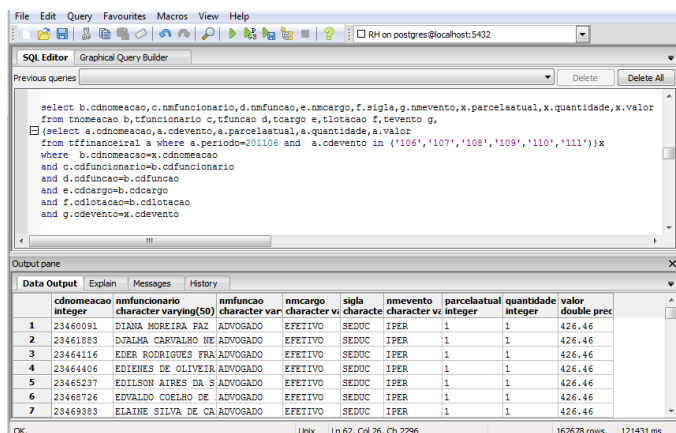
OK. Unix Ln 39, Col 22, Ch 1676 269426 rows. 12308 ms

Figura 41: Pesquisa diretamente na partição

• Teste 5

Esta pesquisa usa uma subconsulta para buscar os dados desejados. Os resultados das pesquisas foram 121,431 ms para a tabela sem partição e 125,487 ms para a tabela com partição. Portanto, a pesquisa dos dados na tabela sem partição foi um pouco mais rápida com 4,056 ms abaixo do que a da tabela com partição, como mostra as figuras 42 e 43. A figura 44 mostra o resultado da consulta aplicada diretamente na partição junho, onde estão os dados, e teve tempo de 9,328 ms 13 vezes abaixo do que os tempos das tabelas com e sem partição.

IV. RESULTADOS E ANÁLISE



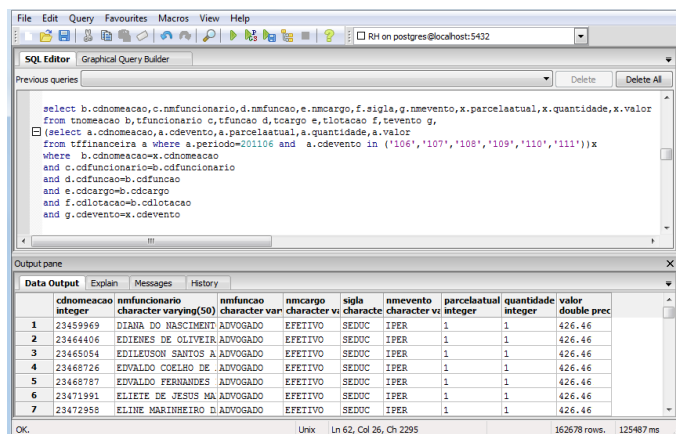
Query:

```
select b.cdnomeacao,c.nmfuncionario,d.nmfuncao,e.nmcargo,f.sigla,g.nmevento,x.parcelaatual,x.quantidade,x.valor
from tnomeacao b,tfuncionario c,tfuncao d,tcargo e,tlotacao f,tevento g,
(select a.cdnomeacao,a.cdevento,a.parcelaatual,a.quantidade,a.valor
from tffinancial1 a where a.periodo=201106 and a.cdevento in ('106','107','108','109','110','111'))x
where b.cdnomeacao=x.cdnomeacao
and c.cdfuncionario=b.cdfuncionario
and d.cdfuncao=b.cdfuncao
and e.cdcargo=b.cdcargo
and f.cdlotacao=b.cdlotacao
and g.cdevento=x.cdevento
```

Output pane:

	cdnomeacao integer	nmfuncionario character varying(50)	nmfuncao character var	nmcargo character va	sigla character	nmevento character v	parcela integer	quantidade integer	valor double prec
1	23466091	DIANA MOREIRA PAZ	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
2	23461883	DJALMA CARVALHO NE	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
3	23464116	EDER ROBERTO FRA	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
4	23464406	EDIEDES DE OLIVEIRA	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
5	23465237	EDILSON AIRES DA S	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
6	23468726	EDVALDO COELHO DE	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
7	23469383	ELAINE SILVA DE CA	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46

Figura 42: Pesquisa na tabela sem partição



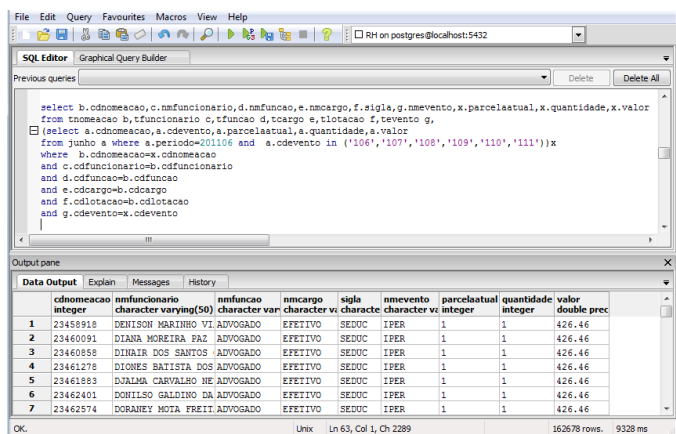
Query:

```
select b.cdnomeacao,c.nmfuncionario,d.nmfuncao,e.nmcargo,f.sigla,g.nmevento,x.parcelaatual,x.quantidade,x.valor
from tnomeacao b,tfuncionario c,tfuncao d,tcargo e,tlotacao f,tevento g,
(select a.cdnomeacao,a.cdevento,a.parcelaatual,a.quantidade,a.valor
from tffinancial1 a where a.periodo=201106 and a.cdevento in ('106','107','108','109','110','111'))x
where b.cdnomeacao=x.cdnomeacao
and c.cdfuncionario=b.cdfuncionario
and d.cdfuncao=b.cdfuncao
and e.cdcargo=b.cdcargo
and f.cdlotacao=b.cdlotacao
and g.cdevento=x.cdevento
```

Output pane:

	cdnomeacao integer	nmfuncionario character varying(50)	nmfuncao character var	nmcargo character va	sigla character	nmevento character v	parcela integer	quantidade integer	valor double prec
1	23469969	DIANA DO NASCIMENT	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
2	23464406	EDIEDES DE OLIVEIRA	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
3	23465054	EDILTON SANTOS	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
4	23468726	EDVALDO COELHO DE	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
5	23468787	EDVALDO FERNANDES	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
6	23471991	ELIETE DE JESUS MA	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
7	23472958	ELAINE MARINHEIRO	D	ADVOGADO	EFETIVO	SEDOC	IPER	1	426.46

Figura 43: Pesquisa na tabela com partição



Query:

```
select b.cdnomeacao,c.nmfuncionario,d.nmfuncao,e.nmcargo,f.sigla,g.nmevento,x.parcelaatual,x.quantidade,x.valor
from tnomeacao b,tfuncionario c,tfuncao d,tcargo e,tlotacao f,tevento g,
(select a.cdnomeacao,a.cdevento,a.parcelaatual,a.quantidade,a.valor
from junho a where a.periodo=201106 and a.cdevento in ('106','107','108','109','110','111'))x
where b.cdnomeacao=x.cdnomeacao
and c.cdfuncionario=b.cdfuncionario
and d.cdfuncao=b.cdfuncao
and e.cdcargo=b.cdcargo
and f.cdlotacao=b.cdlotacao
and g.cdevento=x.cdevento
```

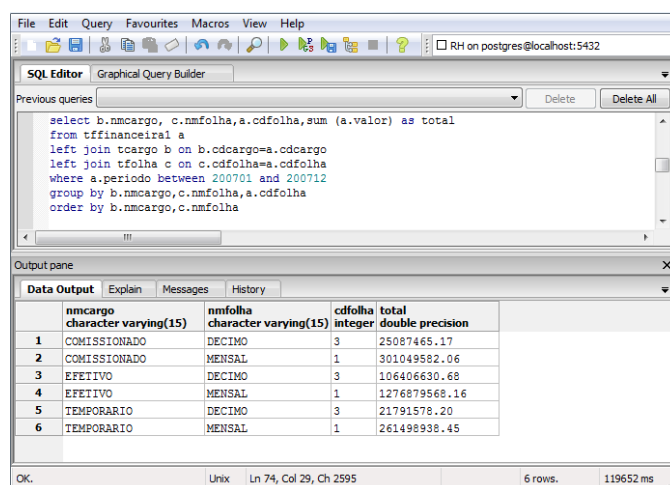
Output pane:

	cdnomeacao integer	nmfuncionario character varying(50)	nmfuncao character var	nmcargo character va	sigla character	nmevento character v	parcela integer	quantidade integer	valor double prec
1	23459818	DENISON MARINHO VI	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
2	23460091	DIANA MOREIRA PAZ	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
3	23460858	DINAIR DOS SANTOS	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
4	23461278	DIONES BATISTA DOS	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
5	23461883	DJALMA CARVALHO NE	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
6	23462401	DONILSO GALDINO DA	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46
7	23462574	DORAYNE MOTA FREIT	ADVOGADO	EFETIVO	SEDOC	IPER	1	1	426.46

Figura 44: Pesquisa diretamente na partição

• Teste 6

As consultas executadas nas tabelas tffinancial e tffinancial1, respectivamente, sem partição e com partição, pesquisaram os dados do ano de 2007 realizando a somatória por código da folha e cargo. Analisando os resultados das consultas, a pesquisa aplicada na tabela com partição figura 46, teve melhor performance cerca de 8% sobre a pesquisa executada na tffinancial1, figura 45.



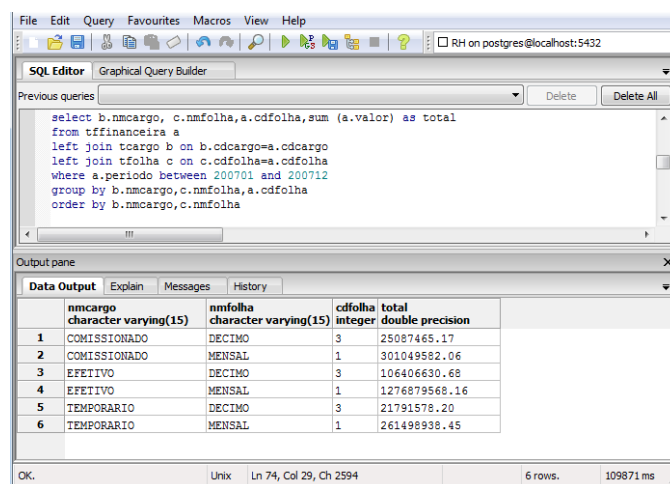
Query:

```
select b.nmcargo, c.nmfolha,a.cdfolha,sum (a.valor) as total
from tffinancial1 a
left join tcargo b on b.cdcargo=a.cdcargo
left join tfolha c on c.cdfolha=a.cdfolha
where a.periodo between 200701 and 200712
group by b.nmcargo,c.nmfolha,a.cdfolha
order by b.nmcargo,c.nmfolha
```

Output pane:

	nmcargo character varying(15)	nmfolha character varying(15)	cdfolha integer	total double precision
1	COMISSONADO	DECIMO	3	25087465.17
2	COMISSONADO	MENSAL	1	301049582.06
3	EFETIVO	DECIMO	3	106406630.68
4	EFETIVO	MENSAL	1	1276879568.16
5	TEMPORARIO	DECIMO	3	21791578.20
6	TEMPORARIO	MENSAL	1	261498938.45

Figura 45: Pesquisa na tabela sem partição



Query:

```
select b.nmcargo, c.nmfolha,a.cdfolha,sum (a.valor) as total
from junho a
left join tcargo b on b.cdcargo=a.cdcargo
left join tfolha c on c.cdfolha=a.cdfolha
where a.periodo between 200701 and 200712
group by b.nmcargo,c.nmfolha,a.cdfolha
order by b.nmcargo,c.nmfolha
```

Output pane:

	nmcargo character varying(15)	nmfolha character varying(15)	cdfolha integer	total double precision
1	COMISSONADO	DECIMO	3	25087465.17
2	COMISSONADO	MENSAL	1	301049582.06
3	EFETIVO	DECIMO	3	106406630.68
4	EFETIVO	MENSAL	1	1276879568.16
5	TEMPORARIO	DECIMO	3	21791578.20
6	TEMPORARIO	MENSAL	1	261498938.45

Figura 46: Pesquisa na tabela com partição

IV. RESULTADOS E ANÁLISE

• Teste 7

Esta pesquisa busca todos os dados do ano de 2005 nas tabelas *tffinancial* e *tffinancial* relacionadas com outras tabelas para montar a informação necessária. Os resultados foram 544,909 ms para a tabela sem partição e com partição 447,923 ms. Ficando com melhor desempenho a consulta executada na tabela com partição com aproximadamente 21% de vantagem na performance. Esta mesma consulta foi aplicada diretamente nas partições – janeiro a dezembro mais decimo – usando o comando *union all* para juntar as informações, obteve tempo 264,732 ms praticamente a metade do tempo que levou as outras duas pesquisas, ficando inquestionável a performance de consulta aplicada diretamente na partição.

periodo	cdnomeacao	nmfuncionario	nmcargo	nmfolha	cdfolha	nmevento	valor
1	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	EMPRESTIMO BB	486.32
2	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	EMPRESTIMO CEF	486.32
3	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	HORA EXTRA	484.62
4	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	HORA NOTURNA	387.69
5	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	IPER	541.32
6	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	IRRF	1082.65
7	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	VENICIMENTO EFETIVO	4921.15

Figura 49: Pesquisa diretamente na partição

• Teste 8

Esta consulta foi submetida em um volume maior de dados que a consulta aplicada no teste anterior e os resultados estão apresentados nas figuras 50, 51 e 52. Comparando estes resultados observa se que a pesquisa executada na tabela com partição teve desempenho superior em 5% que a da tabela sem partição. Esta mesma pesquisa foi executada diretamente em cada partição usando o comando *union all* para juntar as informações. Mesmos repetindo o código 13 vezes para buscar as informações em cada partição e depois juntá-las foi mais eficiente levando um tempo de 574,206 ms, aproximadamente 50% mais rápida que as consultas aplicadas nas tabelas *tffinancial* e *tffinancial*1.

periodo	cdnomeacao	nmfuncionario	nmcargo	nmfolha	cdfolha	nmevento	valor
1	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	EMPRESTIMO BB	486.32
2	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	EMPRESTIMO CEF	486.32
3	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	HORA EXTRA	484.62
4	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	HORA NOTURNA	387.69
5	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	IPER	541.32
6	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	IRRF	1082.65
7	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	VENICIMENTO EFETIVO	4921.15

Figura 47: Pesquisa na tabela sem partição

periodo	cdnomeacao	nmfuncionario	nmcargo	nmfolha	cdfolha	nmevento	valor
1	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	EMPRESTIMO BB	486.32
2	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	EMPRESTIMO CEF	486.32
3	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	HORA EXTRA	484.62
4	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	HORA NOTURNA	387.69
5	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	IPER	541.32
6	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	IRRF	1082.65
7	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	VENICIMENTO EFETIVO	4921.15

Figura 48: Pesquisa na tabela com partição

periodo	cdnomeacao	nmfuncionario	nmcargo	nmfolha	cdfolha	nmevento	valor
1	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	EMPRESTIMO BB	486.32
2	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	EMPRESTIMO CEF	486.32
3	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	HORA EXTRA	484.62
4	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	HORA NOTURNA	387.69
5	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	IPER	541.32
6	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	IRRF	1082.65
7	200501	23457651	JACKSON GRIGORIO S EFETIVO	MENSAL	1	VENICIMENTO EFETIVO	4921.15

Figura 50: Pesquisa na tabela sem partição

V. CONCLUSÃO

O trabalho realizado permite avaliar que a técnica de particionamento em tabela possui grande vantagem quando ela é vultosa ou que venha a ser grande no futuro. Antes de adotar esta técnica deve ser observado como estes dados serão distribuídos nas futuras partições, se esta distribuição não afetará as consultas existentes na busca dos dados, porque eles estarão em partições distintas. É prudente antes de implementar o particionamento analisar com cuidado os impactos das mudanças provocadas pela aplicação da técnica sobre o negócio da empresa, para não ter supressas desagradáveis, como exemplo, tempo de consulta não tão bom quanto esperava, muita mudança nos códigos das consultas, entre outras.

Neste trabalho a técnica de particionamento mostrou uma opção adequada para melhorar o desempenho das consultas, principalmente por que a tabela particionada guarda dados históricos e recebe muitas consultas em determinadas faixas destes dados, como foi exemplificado nos testes mostrados na seção resultados e análise. Estas características também devem ser observadas quando for adotar essa técnica para dar mais subsídio na tomada de decisão.

1. Dificuldade Encontradas

As dificuldades encontradas neste trabalho foram poucas e de pequeno grau. Elas servem mais como recomendações para realização de trabalho similar, evitando, assim, alterações nos dados e desperdício de tempo. Segue abaixo estas recomendações:

- Formatação dos dados na importação, principalmente os dados dos tipos real e numérico.
- Manter copia atualizada dos arquivos importantes em outra máquina.
- Administração do tempo para a realização do trabalho quando envolve a manipulação de grande volume de dados.

- Nas inserções e consultas com grande quantidade de informações ocorreu erro referente à memória. Portanto, inserções e consultas foram feitas com menor quantidade de dados.
- Dificuldades de encontrar livros sobre o assunto tratado neste trabalho.

2. Aplicabilidade do Trabalho

A aplicabilidade da tecnologia citada neste trabalho não restringe um determinado seguimento, empresa, setor ou área, podendo ser implementada em qualquer tabela com volume grande de dados para solucionar problema de lentidão de acesso aos dados, melhorando assim a sua performance. Ela também facilita a administração de cópia de segurança de dados podendo ser executada diretamente na partição desejada. Estas principais funcionalidades da técnica de particionamento melhoras o desempenho do SGDB nas operações de consultas, inserção, atualização e remoção dos dados na tabela particionada e facilitar muito a administração de backup.

BIBLIOGRAFIA

Referências:

- [1] <http://www.oracle.com/technetwork/pt/database/enterprise-edition/documentation/particionamento-banco-de-dados-11g-432098-ptb.pdf>. Acesso em: 06 jul. 2013, 10:40:20.
- [2] http://docs.oracle.com/cd/E11882_01/server.112/e16541/partition.htm#VLDBG00224. Acesso em: 06 jul. 2013, 11:50:25.
- [3] http://docs.oracle.com/cd/B19306_01/server.102/b14220/partconc.htm#i460895. Acesso em: 06 jul. 2013, 15:10:40.
- [4] http://docs.oracle.com/cd/B19306_01/appdev.102/b14258/d_redefi.htm. Acesso em: 06 jul. 2013, 19:35:15.
- [5] http://www.sqlskills.com/resources/Whitepapers/Partitioning%20in%20SQL%20Server%202005%20Beta%20II.htm#_Toc79339947. Acesso em: 07 jul. 2013, 09:54:50.
- [6] <http://msdn.microsoft.com/pt-br/library/ms179854.aspx>. Acesso em: 07 jul. 2013, 12:20:10.
- [7] <http://msdn.microsoft.com/pt-br/library/ms187802.aspx>. Acesso em: 07 jul. 2013, 14:45:15.
- [8] <http://dev.mysql.com/doc/refman/5.5/en/>. Acesso em: 07 jul. 2013, 16:35:45.
- [9] <http://dev.mysql.com/doc/refman/5.5/en/partitioning-types.html>. Acesso em: 07 jul. 2013, 18:10:35.
- [10] <http://www.postgresql.org/docs/8.3/static/ddl-partitioning.html>. Acesso em: 07 jul. 2013, 20:55:25.