FACULDADE DE TECNOLOGIA DE AMERICANA
Curso de Análise e Desenvolvimento de Sistemas

DIEGO VERONEZ NUNES

SQL TUNING: PERFORMANCE POR MEIO DE UMA MELHOR CODIFICAÇÃO

Americana, SP

2017

FACULDADE DE TECNOLOGIA DE AMERICANA
Curso de Análise e Desenvolvimento de Sistemas

DIEGO VERONEZ NUNES

SQL TUNING: PERFOMANCE POR MEIO DE UMA MELHOR CODIFICAÇÃO

DIEGO VERONEZ NUNES

Trabalho de Conclusão de Curso desenvolvido em cumprimento à exigência curricular do Curso Análise e Desenvolvimento de Sistemas, sob orientação da Prof. MSc Eduardo Antonio Vicentini.

Área: Programação, Banco de Dados, SQL

Americana, SP

2017

FICHA CATALOGRÁFICA – Biblioteca Fatec Americana - CEETEPS
Dados Internacionais de Catalogação-na-fonte

N924s NUNES, Diego Veronez

SQL tuning: performance por meio de uma melhor codificação./ Diego Veronez Nunes. – Americana: 2017.

54f.

Monografia (Curso de Tecnologia em Análise e Desenvolvimento de Sistemas) - - Faculdade de Tecnologia de Americana – Centro Estadual de Educação Tecnológica Paula Souza

Orientador: Prof. Ms. Eduardo Antonio Vicentini

1. SQL - banco de dados I. VICENTINI, Eduardo Antonio II. Centro Estadual de Educação Tecnológica Paula Souza – Faculdade de Tecnologia de Americana

CDU: 681.3.07

Diego Veronez Nunes

SQL TUNING:

Performance por meio de uma melhor codificação

Trabalho de graduação apresentado como exigência parcial para obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas pelo CEETEPS/Faculdade de Tecnologia – FATEC/ Americana.
Área de concentração: Banco de Dados.

Americana, 26 de junho de 2017.

Banca Examinadora:

Eduardo Antonio Vicentini (Presidente)
Mestre
Fatec Americana

Jonas Bodê (Membro)
Especialista
Fatec Americana

Antônio Alfredo Lacerda (Membro)
Especialista
Fatec Americana

RESUMO

Desde o surgimento da informática o armazenamento de dados sempre foi foco de preocupação, tendência está presente nos dias de hoje no qual produzimos grande volume de informações as quais necessitam serem armazenadas. O presente trabalho tem por objetivo realizar um estudo que conceitua desde o surgimento dos bancos de dados relacionais conjuntamente a linguagem SQL (Structured Query Language), a atual necessidade de aumento de desempenho. Técnicas e métricas foram utilizadas para demonstrar de forma expositiva e exemplificada a possibilidades de melhoria de desempenho com a utilização de técnicas de tuning na codificação.

Palavras Chave: Banco de Dados, SQL Tuning, Performance

ABSTRACT

Since the advent of computer science, data storage has always been a focus of concern, a trend is present nowadays in which we produce large amounts of information which need to be stored. The present work aims to conduct a study that conceptualizes since the emergence of relational databases together the SQL (Structured Query Language), the current need for performance increase. Techniques and metrics were used to demonstrate in an expository and exemplified way the possibilities of performance improvement with the use of tuning techniques in coding.

Keywords: Databases, SQL Tuning, Performance

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

INTRODUÇÃO

O armazenamento de dados no âmbito mundial se encontra em um crescente que excede a lei de Moore. Esta necessidade junto ao desenvolvimento do mercado são os grandes motores das novas atualizações e novos paradigmas dos sistemas gerenciadores de bancos de dados.

Dado o cenário, é cada vez mais necessário desempenho nas transações relacionadas à movimentação destes dados. A evolução tecnológica dos Sistemas Gerenciadores de Banco Dados (SGBD) junto ao hardware buscam por esta finalidade, porém podem ser anulados com um mal-uso de uma codificação não estruturada corretamente ou com métodos menos eficientes.

Sendo assim, este trabalho tem por objetivo a compreensão do SGBD, assim como o comportamento esperado e sua rotina de execução de comandos SQL, convergindo na apresentação de melhores práticas de utilização por meio de uma melhor codificação e estruturação nas aplicações desenvolvidas.

Este trabalho foi estruturado em três capítulos sendo o primeiro conceitual sobre o que é um banco de dados relacional e seu funcionamento, além da linguagem utilizada pelas soluções relacionais, de seu surgimento a características técnicas.

O segundo capítulo trata formas de melhoria de desempenho, assim como um maior entendimento sobre a forma de execução de uma consulta frente a um SGBD.

Baseando-se nos capítulos anteriores o terceiro capítulo finaliza o conteúdo proposto em forma de uma análise sobre todo o conteúdo apresentado e as possibilidades e desafios enfrentados em sua implementação.

A elaboração deste trabalho tem por objetivo demonstrar de forma a conciliar a teoria e prática possíveis pontos de melhoria de desempenho, através de técnicas de tuning, porém há uma grande variedade de soluções de bancos de dados relacionais disponíveis no mercado e suas particularidades.

Dentre elas soluções voltadas para utilização industrial como o DB2 da IBM, fruto das pesquisas do Dr. Codd teve sua origem focada em *mainframes* sendo distribuído através do tempo para outros ambientes e sistemas operacionais

diversos. E soluções as Oracle, a qual foi à primeira empresa a desenvolver e distribuir uma solução para bancos de dados relacionais, seu lançamento de versões.

A Oracle tem por cultura buscar atender a próxima tendência de mercado identificada, já tendo atuado com foco na Internet, computação em *Grid* e atualmente em *Cloud computing*.

Existem também outras soluções como o Mysql desenvolvido em 1994 pelos europeus David Axmark, Allan Larsson e Michael "Monty" Widenius. Tendo utilização amplamente difundida principalmente junto à linguagem de programação PHP. E tendo consequentemente sua maior utilização no que se diz respeito a armazenamento em servidores web.

Visando uma solução com maior disponibilidade para execução de pesquisas e alta aceitação de mercado, foi escolhida para este trabalho a solução da Oracle versão 11G, a qual disponibiliza uma versão gratuita para estudo e ainda atualmente domina o mercado mundial no quesito banco de dados.

1. BANCO DE DADOS

1.1. BANCO DE DADOS

Conforme apresentado por Fehily (2008), há comumente uma confusão relacionada aos termos banco de dados (BD) e sistemas gerenciadores de bancos de dados (SGBD), sendo muitas vezes confundidos como idênticos, porém trata-se de coisas distintas.

Sendo assim, BD nada mais é que um componente do SGBD, se referindo aos dados em si. Podendo ser comparado a um container de informações, que possui um ou mais arquivos, e armazena a informação de física e forma estruturada.

No entanto de acordo com Taylor (2008) bancos de dados consistem em uma junção de dados e metadados, sendo metadados os dados os quais descrevem a estrutura atribuída a um banco de dados, os quais facilitam o acesso aos dados graças ao conhecimento da forma a qual estão organizados no modelo relacional. Estes metadados são armazenados no banco de dados em uma área especifica denominado dicionário de dados, o qual descreve as tabelas, colunas, indexes, chaves e outros itens os quais o compõe.

1.2. SISTEMAS GERENCIADORES DE BANCO DE DADOS

No que se refere ao Sistema Gerenciador de Banco Dados (SGBD) Fehily (2008) atribui a ele a responsabilidade em controlar os dados de forma organizada, prezando por sua integridade e permitindo o retorno destes dados. Além disso, o SGBD é responsável pelas tarefas de armazenamento físico, segurança, replicação, *backup* e recuperação de possíveis erros.

Segundo a ótica de Taylor (2008) um SGBD é um pacote de programas os quais são utilizados para definir, administrar e processar bancos de dados e as aplicações a ele relacionadas. Podendo ser definido como a ferramenta a ser utilizada para construir a estrutura e efetuar operações com os dados contidos em um BD.

Silberchatz (2006) por sua vez designa como principais vantagens do SGBD à utilização de armazenamento em arquivos, como era praticado anteriormente a sua existência, a redundância e inconsistência dos dados, dificuldade de acessos a dados, isolamento de dados, problemas de integridade, problemas de atomicidade, anomalias de acesso concorrente e problemas de segurança.

Em seguida são apresentadas as soluções propostas pelo surgimento do sistema gerenciador de banco dados (SGBD):

Redundância e inconsistência dos dados: Indiferentemente do número de programadores ou linguagens de programação envolvidas o armazenamento em um banco de dados pode se aproveitar de uma estrutura padronizada, assim como alguns problemas de inconsistência de dados resolvidos pelo fato do armazenamento se encontrar todo em um mesmo local.

Dificuldade de acessos a dados: A recuperação de dados se dá de forma mais eficaz e organizada pela estrutura aplicada no desenvolvimento do banco, não tendo a necessidade de um desenvolvimento de um novo programa especifico para cada consulta. Sendo suprida a necessidade apenas pela utilização em conjunto de uma consulta SQL nos dados desejados.

Isolamento de dados: O fato dos dados todos estarem contidos em um mesmo local permite um acesso mais simplificado eliminando a possibilidade de incompatibilidade na busca por informações em arquivos de tipos diferentes.

Problemas de integridade: Facilidade na aplicação e manutenção de restrições de consistência necessárias, mais uma vez tendo como facilidade a aplicação de código voltada para um local focalizado ao invés de diversos arquivos.

Problemas de atomicidade: No caso de haver falha em uma transação os dados devem voltar em seu estado original.

Anomalias de acesso concorrente: O mesmo dado pode ser acessado por múltiplos usuários ou sistemas diferentes simultaneamente, e a alteração deste pode apresentar resultados errôneos de acordo com essa competição de acesso. O

SGBD tem por finalidade gerenciar este tipo de transação resolvendo o problema em questão.

Problemas de segurança: Geralmente os usuários em sua totalidade não devem ter acesso a todas as informações, ou mesmo aos objetos contidos no banco. Sendo uma tarefa do SGBD a gestão de permissão de acessos aos usuários conforme a necessidade do projeto.

Ao observar o fluxo de acesso de um usuário ao BD evidencia-se a existência do SGBD como uma entidade a qual gerencia o banco também como uma espécie de camada de acesso.

A figura 1 ilustra um diagrama o qual apresenta de forma decrescente o caminho trilhado pelo usuário para acessar um banco de dados em uma instalação local:

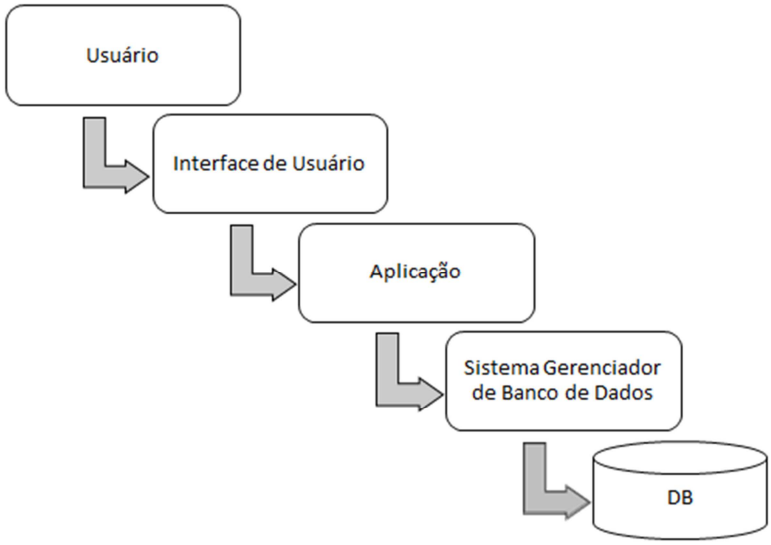


Figura 1: Diagrama acesso banco de dados
FONTE: Autoria própria

1.3. MODELO RELACIONAL

1.3.1. HISTÓRICO

O modelo relacional, tal qual conhecemos, foi criado em 1970 por um pesquisador da IBM chamado Dr. Edgar Frank Codd. Sendo no entanto curiosamente o modelo por ele desenvolvido só aplicado comercialmente uma década após sua pesquisa por uma pequena companhia de startup chamada Oracle.

Nos dias atuais o modelo relacional tem sua utilização altamente difundida, estando presentes em diversas ferramentas líderes de mercado como IBM DB2, Mysql, SQL Server e nas próprias versões da ferramenta Oracle.

1.3.2. TABELAS

Segundo Silberschatz (2006) um banco de dados relacional é composto por uma coleção de tabelas identificadas por nomes distintos, as quais contêm em sua composição linhas e colunas que representam uma relação entre um determinado conjunto de valores armazenados em cada registro. Podemos assim considerar uma tabela como um conjunto de entidades, as quais têm seus registros armazenados no formato de linha.

A base de todo banco de dados relacional se dá nas tabelas nele contidas, as quais armazenam os dados de forma organizada quanto ao seu tipo e relação com as demais tabelas contidas no Banco de Dados (BD).

Conforme demonstrado na Figura 2, as tabelas são divididas em três principais partes linhas, colunas e dados:

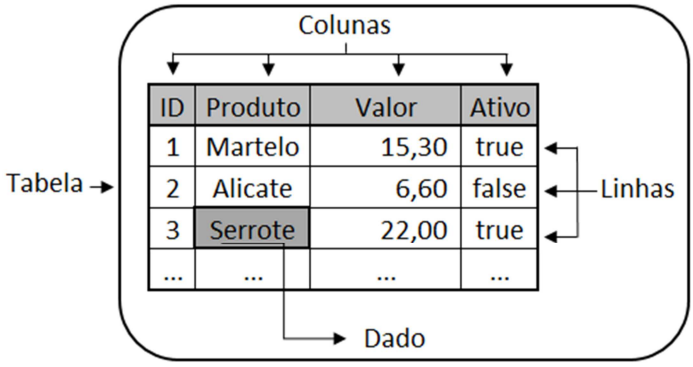


Figura 2: Estrutura básica de uma tabela
FONTE: Autoria própria

1.3.3. CHAVES

O funcionamento de um Banco de Dados (BD), assim como no desenvolvimento de software, necessita atender algumas regras segundo o comportamento esperado dos dados que serão armazenados. A identificação e especificação destas regras são dadas na modelagem de dados, onde o esquema do banco é pensado sistemicamente segundo suas relações e necessidades.

As chaves permitem o cumprimento destas regras especificadas, abortando qualquer operação que infrinja a ação do dado e a chave utilizada. Abaixo se encontram as principais chaves da linguagem SQL e suas restrições:

NOT NULL: Não permite que a coluna indicada pela chave armazene um valor nulo.

UNIQUE: Garante que não haja valores duplicados em uma coluna.

PRIMARY KEY: É utilizada geralmente como identificação dos dados armazenados, se trata de uma combinação entre *NOT* e *UNIQUE* não permitindo assim um valor duplicado ou nulo.

FOREIGN KEY: Referencia um valor o qual se encontra armazenado em outra tabela, usualmente o valor de uma *PRIMARY_KEY*.

CHECK: Garante que o valor de uma coluna atende uma condição especifica.

DEFAULT: Especifica um valor padrão para a coluna.

1.4. AS DOZE REGRAS DE CODD

O criador do modelo relacional Dr. Edgar Frank Codd publicou em 1985, durante o semanário da indústria de computação (*COMPUTER WORLD*), um artigo que apresentava em forma de tópicos, quais regras um Sistema Gerenciador de Banco Dados (SGBD) no modelo relacional deve atender. Uma visão mais refinada sobre o modelo relacional pode ser conseguida após a compreensão das regras propostas por Codd (Codd's 12 Rules, 2016).

Regra 0: Sistema relacional

Para que um Sistema Gerenciador de Banco Dados (SGBD) se qualifique como sendo relacional há a necessidade de que o sistema de relacionamento, proposto por Codd seja utilizado exclusivamente para o gerenciamento do banco de dados.

1ª regra: As regras para informações

A primeira regra diz respeito à estrutura de dados utilizada para armazenar os dados e representar os relacionamentos entre eles, objetivando então que as relações sejam as únicas estruturas de dados utilizadas em um banco de dados relacional. Sendo assim, o conceito explicita que um banco de dados relacional deve possuir como estrutura apenas tabelas.

Como justificativa a esta regra Codd baseia-se na flexibilidade entre os relacionamentos lógicos, onde a relação entre dados pode exceder a projetada na implementação do banco com operações de relacionamento tal qual o *JOIN*. E a também flexibilidade nos esquemas dos bancos de dados relacionais onde há também a possibilidade de modificar o esquema durante a utilização do banco.

2ª regra: A regra de acesso garantido

Deve existir a garantia de que os dados inseridos em um banco de dados relacional possam ser retornados e ordenados de forma unitária ou totalitária. Por

definição esta regra demonstra que com apenas três aspectos de um dado específico necessitam ser conhecidos para localizá-lo. Sendo eles: o nome da tabela, o nome da coluna e a chave primária da linha a que se refere o registro.

Não há, no entanto, nenhuma regra dentre este conjunto que especifique a obrigatoriedade de que se tenha uma chave primária única em cada linha, porém algumas relações apenas são possíveis graças à existência desta.

3ª regra: Tratamento sistemático de valores nulos

Valores nulos são apresentados pelo banco de forma a expressar que um dado é desconhecido ou se encontra ausente, sendo distintos de um conjunto de caracteres vazio ou mesmo em branco e a qualquer número. Sendo assim, Codd apresenta a necessidade de um tratamento especial para os nulos independentemente do tipo de dado.

4ª regra: Catálogo on-line dinâmico baseado no modelo relacional

Valores nulos são apresentados pelo banco de forma a expressar que um dado é desconhecido ou se encontra ausente, sendo distinto a um conjunto de caracteres vazio ou mesmo em branco e a qualquer número.

Sendo assim, Codd primeiramente apresentou a necessidade de que o Sistema Gerenciador de Banco Dados (SGBD) possa armazenar um valor nulo, independentemente do tipo de dado o qual representa, e em seguida que o banco possa retornar tais valores e tratá-los se necessário.

5ª regra: A regra de sublinguagem de dados abrangentes

Apesar de poder suportar diversas linguagens e formas de utilização de terminal, o Sistema Gerenciador de Banco Dados (SGBD) deve possuir pelo menos uma linguagem declarativa bem definida a qual permita a definição, visualização e manipulação dos dados. Assim como a restrição de integridade e a autorização e gerenciamento das transações. Em sua maioria a linguagem utilizada é a SQL (Structured Query Language), a qual será abordada no próximo capitulo deste trabalho.

6ª regra: A regra da visualização de atualização

A sexta regra expressa que visualizações e atualizações de dados necessárias sejam efetuadas através do Sistema Gerenciador de Banco Dados (SGBD), onde uma vez que a atualização atenda aos critérios necessários o SGBD deve ser capaz de efetuar a atualização e propagar estas as tabelas básicas.

7ª regra: Inserção de alto nível, atualização e exclusão

O Sistema Gerenciador de Banco Dados (SGBD) deve dar suporte à inserção, atualização e exclusão de dados de forma a poder manipular um conjunto de dados. Ou seja, Codd assegura com esta regra que múltiplas linhas possam ser tratadas ao mesmo tempo pelo SGBD pelos comandos DML (*Data Manipulation Language*) *INSERT*, *UPDATE* e *DELETE*.

8ª regra: Independência de dados físicos

Os aplicativos e recursos não devem sofrer influência lógica quando ocorrer qualquer alteração a estrutura física tais como o método de acesso ou estruturas de armazenamento físico.

9ª regra: Independência de dados lógicos

Os recursos e aplicativos não devem sofrer qualquer influência lógica quando houver alterações de estruturas de tabela, que contenham os valores originais da mesma (inserção de uma nova coluna ou alteração da ordem). Assim como alterações nos relacionamentos e *views* as quais não devem causar nenhum impacto nas aplicações ou ter uma baixa influência sobre.

10ª regra: Independência de integridade

O Sistema Gerenciador de Banco Dados (SGBD) deve permitir que todas as restrições de integridade relacionais sejam definidas na linguagem relacional SQL, e armazenadas no catálogo do sistema e não no nível da aplicação. As aplicações não devem sofrer quaisquer influências quando ocorrer mudanças nas restrições de integridade.

11ª regra: Independência de distribuição

O Sistema Gerenciador de Banco Dados (SGBD) deve permitir que os dados possam ser armazenados de forma distribuída, ou seja, em mais de um computador, porém deve ser apresentado de maneira centralizada para o usuário final. Sendo assim o banco de dados é a soma de todas as partes distribuídas em máquinas distintas.

12ª regra: Não transposição das regras

Caso o Sistema Gerenciador de Banco Dados (SGBD) de suporte ao acesso de baixo nível aos dados do Banco de Dados (BD), não pode existir um modo que permita negligenciar as demais regras de integridade.

1.5. SQL - STRUCTURED QUERY LANGUAGE

O acrônimo SQL tem por significado Structured Query Language o qual traduzido tem por significado (linguagem estruturada de pesquisa), teve seu desenvolvimento inicial voltado para bancos de dados relacionais.

Teve origem na década de 70, mais especificamente em 1974, nos laboratórios de pesquisa da IBM na Califórnia onde foi apelidado de SEQUEL Structured English Query Language. O primeiro protótipo de aplicação foi implementado em 1975 tendo diversas revisões e ampliações entre 1976 e 1977 e também seu nome alterado por razões jurídicas para SQL.

Com o passar do tempo à linguagem foi difundida em diversos Sistemas Gerenciadores de Banco de Dados (SGBD) tornando-se um padrão no que se referem ambientes de bancos de dados relacionais. Sendo assim, naturalmente em 1982 tornou-se a linguagem padrão de bancos de dados relacionais pelo ANSI – *American National Standard Institute*.

No entanto apesar de ser uma linguagem padronizada existem diversas implementações, baseadas em SQL, desenvolvidas pelas empresas responsáveis pelas soluções de SGBD. Este trabalho, todavia, será inteiramente referenciado ao padrão ANSI.

Originalmente a ideia da linguagem SQL previa sua utilização somente de forma interativa, no entanto ao longo do tempo ele adquiriu diversas outras funcionalidades além do papel atribuído aos SGBD.

Abaixo estão listadas algumas outras funcionalidades adquiridas pela linguagem, segundo Machado (2012):

- **Linguagem interativa de consulta:** Graças à possibilidade de efetuar consultas poderosas, usuários não necessitam criar um programa para busca de dados podendo utilizar *Forms* ou ferramentas de relatório para isto;
- **Linguagem de programação para acesso a banco de dados:** Acesso de dados através de código SQL embutido a programas de aplicação;
- **Linguagem de administração de banco de dados:** Os administradores de banco de dados se utilizam de comandos SQL para realizar diversas tarefas;
- **Linguagem cliente/servidor:** Através de uma rede local as comunicações entre as máquinas clientes e a base de dados no servidor se utilizam de comandos SQL, viabilizando assim a diminuição do tráfego na rede;
- **Linguagem de banco de dados distribuído:** A linguagem SQL auxilia na comunicação entre diversos nós conectados ao sistema de computação assim como na comunicação com outros sistemas;
- **Caminho de acesso a diferentes bancos de dados em diferentes máquinas:** O SQL auxilia na conversão entre diferentes soluções de banco de dados encontradas locais diferentes.

1.5.1. CATEGORIAS DE COMANDOS SQL

Segundo Fehily (2008) os comandos SQL podem ser classificados em três principais categorias, visto suas características e funções. Agrupando assim conjuntos de instruções com funções similares para melhor compreensão.

Abaixo temos representadas estas categorias:

- **DML – Data manipulation language:** São os comandos responsáveis pela inserção, retorno, edição e deleção dos dados do banco de dados.

- **DDL – Data definition language:** São os comandos responsáveis pela criação, modificação e destruição de objetos do banco de dados como tabelas, índices e *views*.

- **DCL – Data control language:** Estes comandos dizem respeito as permissões de usuários para manipular determinados objetos dentro do banco de dados.

Apesar da classificação os comandos podem conter variações em sua sintaxe de acordo com o Sistema Gerenciador de Banco Dados (SGBD) utilizado Fehily (2008), sendo assim é uma boa prática sempre consultar a documentação da solução em questão para se certificar sobre como efetuar a implementação.

Como os comandos DCL têm por finalidade somente a organização quanto a permissões de acesso ao banco de dados para manipulação dos dados, os mesmos não serão abordados neste trabalho por terem finalidade administrativa.

1.5.2. SÍNTAXE BÁSICA SQL

A semântica básica da linguagem SQL é demonstrada na figura 3, tendo em seguida à apresentação da sintaxe em forma de tópicos:

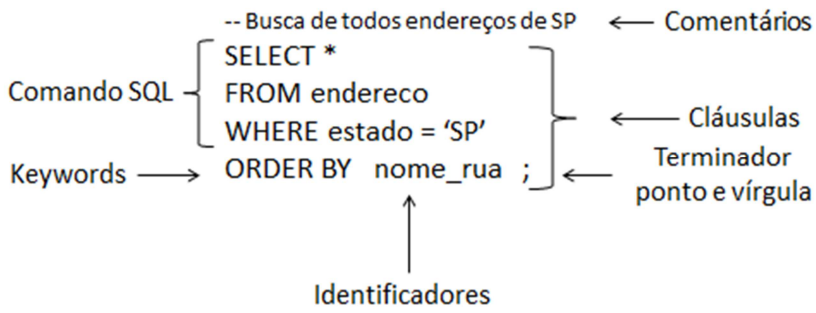


Figura 3: Sintaxe comando SQL
FONTE: Autoria própria

Comentários: Comentários são inseridos por dois hifens consecutivos e se estendem por toda a linha, por padrão se destinam somente ao peopleware e são ignorados pelos interpretadores. Eles permitem que sejam inseridos pelos desenvolvedores textos os quais descrevem o que determinado pedaço de código executa, explicam sua execução ou ainda marcam a alteração de um pedaço de código e o motivo da mesma.

Comando SQL: Comandos SQL são designados pela combinação de tokens introduzidos por uma *keyword*. Tokens por sua vez se referem às partículas básicas e indivisíveis da linguagem SQL, não podendo ser abreviados.

Cláusulas: Um comando SQL possui uma ou mais cláusulas geralmente é um fragmento do comando SQL que é introduzido por uma keyword, é requerido ou opcional, e tem de seguir uma ordem particular. *SELECT*, *FROM* e *WHERE* são designam três cláusulas neste exemplo.

Keywords: São palavras reservadas pelo SQL que possuem um significado especial para a linguagem. Cada Sistema Gerenciador de Banco Dados (SGBD) utiliza de um conjunto específico de *Keywords*, que pode ser encontrado em sua respectiva documentação.

Identificadores: São palavras utilizadas para nomear os objetos do banco de dados.

Terminador ponto e vírgula: Um comando SQL é encerrado por padrão por ponto e vírgula.

1.5.3. TIPOS DE DADOS

A linguagem SQL junto à estrutura de dados das tabelas permite o armazenamento de diversos tipos de dados diferentes, onde de acordo com a necessidade encontrada se é declarada um tipo específico de dado, também conhecidos como tipo de coluna, a declaração do tipo é feita principalmente nas cláusulas *CREATE* e *ALTER TABLE*.

Conforme descrito anteriormente à organização dos dados em um banco de dados é um ponto crucial para seu correto funcionamento, para isso a linguagem SQL dispõe de tipos específicos de dados os quais são utilizados de acordo com as características dos dados que serão inseridos em suas tabelas.

A figura 4 apresenta o uso de diferentes tipos de dados aplicados a uma tabela:

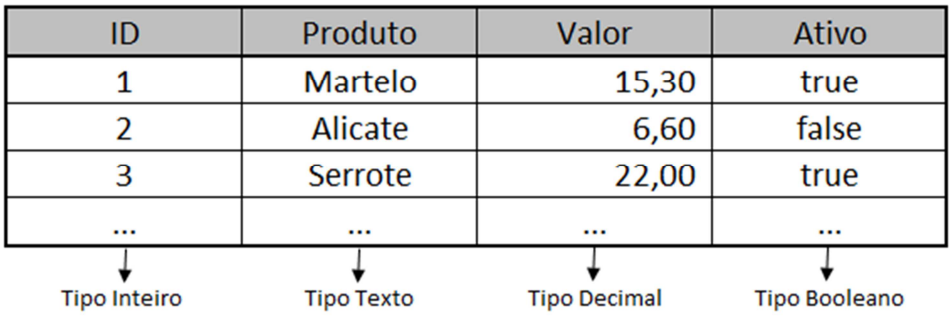


Figura 4: Representação tabela e seus tipos de dados
FONTE: Autoria própria

A tabela 1 representada por categoria, conforme Taylor (2008), os principais tipos de dados definidos por padrão na linguagem SQL, podendo variar esses de acordo com o Sistema Gerenciador de Banco Dados (SGBD) escolhido:

Tipo de dado	Descrição
CHARACTER(n)	String de caractere. Tamanho fixo especificado em (n)
VARCHAR(n) ou	String de caractere. Tamanho variável. Tamanho máximo (n)
CHARACTER VARYING(n)	
BINARY(n)	String binária. Tamanho máximo (n)
BOOLEAN	Armazena valores TRUE ou FALSE (Verdadeiro ou falso)
VARBINARY(n) ou	String binária. Tamanho da variável. Tamanho máximo (n)
BINARY VARYING(n)	

Tabela 1: Tipos de dados categorizados
FONTE: TAYLOR, Allen G. SQL For Dummies.

Ao declarar um tipo texto tem-se como requisito a declaração do tamanho o qual pode ser armazenado. Graças a esta declaração temos como principal característica, a qual diferencia, os tipos texto a alocação de espaço de forma dinâmica ou estática.

Para armazenamento de numerais tem-se outros tipos específicos os quais se encontram relacionados na tabela 2:

Tipo de dado	Descrição
INTEGER(p)	Número inteiro (sem casas decimais). Precisão (p)
SMALLINT	Número inteiro (sem casas decimais). Precisão 5
INTEGER	Número inteiro (sem casas decimais). Precisão 10
BIGINT	Número inteiro (sem casas decimais). Precisão 19
DECIMAL(p,s)	Número exato, precisão (p), escala (s).
NUMERIC(p,s)	Número exato, precisão (p), escala (s). (Idêntico ao DECIMAL)
FLOAT(p)	Número aproximado, com um valor de precisão mínimo.
REAL	Número aproximado com precisão de 7 dígitos
FLOAT	Número aproximado com precisão de 16 dígitos
DOUBLE PRECISION	Número aproximado com precisão de 16 dígitos

Tabela 2: Tipos de dados numerais
FONTE: TAYLOR, Allen G. SQL For Dummies

Os tipos numéricos têm como principais diferenças se referenciam ao conjunto dos números inteiros ou racionais, sua capacidade de armazenamento assim como sua precisão de casas decimais.

A tabela 3 apresenta os principais tipos de dados referentes ao armazenamento de data e alguns tipos específicos:

Tipo de dado	Descrição
DATE	Armazena os valores (ano, mês e dia)
TIME	Armazena os valores (hora, minuto e segundos)
TIMESTAMP	Armazena os valores (Ano, mês, dia, hora, minuto e segundo)
INTERVAL	Composto por um número de campos de números inteiros, representa um período de tempo
ARRAY	É um conjunto de uma coleção ordenada de elementos
MULTISET	Uma variável de comprimento e coleta desordenada de elementos.
XML	Armazena dados XML

Tabela 3: Tipos de dados datas
FONTE: TAYLOR, Allen G. SQL For Dummies

1.5.4. COMANDOS DML

1.5.4.1. SELECT

É a declaração utilizada para a recuperação de dados do banco retornando os conteúdos das colunas de uma ou mais tabelas as quais são especificadas na cláusula *FROM*.

Abaixo um exemplo de uma cláusula *SELECT* a qual retorna dados de uma tabela:

SELECT nome_coluna, nome_coluna
FROM nome_tabela;

1.5.4.2. DISTINCT

A cláusula *DISTINCT* tem por finalidade o retorno de dados distintos quando há vários dados idênticos em uma consulta, ou seja, valores repetidos.

Abaixo um exemplo de sua sintaxe:

SELECT DISTINCT nome_coluna, nome_coluna
FROM nome_tabela;

1.5.4.3. WHERE

A cláusula *WHERE* é utilizada para filtrar os resultados segundo um critério especificado. Permitindo a utilização de diversos predicativos os quais auxiliam na filtragem de dados.

Os predicados são divididos principalmente em predicados de comparação e outros predicados.

Predicados de comparação	
=	Igual
<>	Não igual
<	Menor que
<=	Menor igual que
>	Maior que
>=	Maior igual que

Tabela 4: Predicados comparação
FONTE: FEHILY, Chris. Visual QuickStart Guide SQL.

Outros predicados	
ALL	Todos os valores
DISTINCT	Valores distintos, se, repetição
IN	Valores em formato de listagem
MATCH	Checa dados durante transação
NOT LIKE	Não semelhante
OVERLAPS	Determina intervalos de tempo
SOME, ANY	Algum
BETWEEN	Entre
EXISTS	Existe
LIKE	Semelhante
NOT IN	Não contido
NULL	Nulo
SIMILAR	Similar
UNIQUE	Único

Tabela 5: Outros predicados
FONTE: FEHILY, Chris. Visual QuickStart Guide SQL.

Abaixo um exemplo da sintaxe:

SELECT nome_coluna, nome_coluna
FROM nome_tabela
WHERE nome_coluna valor_operador;

1.5.4.4. OPERADORES LÓGICOS

Como complemento a cláusula *WHERE* podem-se utilizar os operadores lógicos *AND*, *OR* e *NOT* para adicionar condições ao *SELECT*, sendo o operador *AND* verdadeiro somente se a condição anterior também for verdadeira, e o operador *OR* é verdadeiro se a primeira ou segunda condição forem verdadeiros.

O operador *NOT* funciona de forma a negar a condição expressa, e tem também por diferença não ter uma conexão com os demais operadores podendo ser utilizado somente na clausula *WHERE* em alguns casos.

Abaixo um exemplo de sua sintaxe:

SELECT nome_coluna, nome_coluna
FROM nome_tabela
WHERE NOT nome_coluna condição valor_operador
AND nome_coluna condição valor_operador
OR nome_coluna condição valor_operador;

Pode-se utilizar mais de uma cláusula *AND*, *OR* ou *NOT* em uma clausula de consulta de acordo com a especificidade necessária.

1.5.4.5. ORDER BY

A cláusula *ORDER BY* tem por função ordenar os resultados retornados pelo *SELECT* de forma ascendente ou descendente e também pode ser aplicada em mais de uma coluna, seguindo a ordenação conforme declarado.

Abaixo um exemplo de sua sintaxe:

SELECT nome_coluna, nome_coluna
FROM nome_tabela
WHERE NOT nome_coluna condição valor_operador
AND nome_tabela
OR nome_tabela
ORDER BY nome_coluna ASC|DESC, nome_coluna ASC|DESC;

A ordenação ascendente é dada por default na cláusula *ORDER BY*, no entanto pode-se utilizar o parâmetro ASC em frente ao nome da coluna para designar a ordenação, assim como o parâmetro DESC para uma ordenação descendente dos dados.

1.6. FUNÇÕES

Para facilitar seu uso o SQL possui diversas funções nativas a linguagem para a execução de cálculos e tratamentos de dados, as principais funções são descritas a seguir:

1.6.1. UCASE

A função *UCASE*( ) converte o todo o valor de um campo texto para caracteres maiúsculos, abaixo um exemplo de sua sintaxe:

SELECT MAX(nome_coluna)
FROM nome_tabela;

1.6.2. LCASE

A função *LCASE*( ) converte o todo o valor de um campo texto para caracteres minúsculos, abaixo um exemplo de sua sintaxe:

SELECT LCASE(nome_coluna)
FROM nome_tabela;

1.6.3. ROUND

A função *ROUND*() é utilizada para arredondar um campo numérico determinado para o número de casas decimais desejadas, abaixo um exemplo de sua sintaxe:

SELECT ROUND(nome_coluna, numero_casas_decimais)
FROM nome_tabela;

1.6.4. LEN

A função *LEN*( ) retorna o tamanho, número de caracteres, de valor de um campo texto especificado, abaixo um exemplo de sua sintaxe:

SELECT LEN(nome_coluna)
FROM nome_tabela;

1.6.5. MIN

A função *MIN*( ) retorna o menor valor de uma coluna selecionada, abaixo um exemplo de sua sintaxe:

SELECT MIN(nome_coluna)
FROM nome_tabela;

1.6.6. MAX

A função *MAX*( ) retorna o maior valor de uma coluna selecionada, abaixo um exemplo de sua sintaxe:

SELECT MAX(nome_coluna)
FROM nome_tabela;

1.6.7. GROUP BY

A função *GROUP BY* agrupa valores retornados de uma pesquisa por uma ou mais colunas especificadas, abaixo um exemplo de sua sintaxe:

SELECT MIN(nome_coluna), função-agregação(nome_coluna)
FROM nome_tabela
WHERE nome_coluna condição valor_operador
GROUP BY nome_coluna;

1.6.8. SUM

A função *SUM*( ) retorna a soma total dos valores contidos em uma coluna, abaixo um exemplo de sua sintaxe:

SELECT MAX(nome_coluna**)**
FROM nome_tabela;

1.6.9. AVG

A função *AVG*( ) retorna a média dos valores contidos em uma coluna, abaixo um exemplo de sua sintaxe:

SELECT AVG(nome_coluna**)**
FROM nome_tabela;

1.6.10. COUNT

A função *COUNT*( ) retorna o número de linhas que correspondem a um critério especifico, abaixo um exemplo de sua sintaxe:

SELECT COUNT(nome_coluna)
FROM nome_tabela;

1.6.11. HAVING

A cláusula *HAVING* é adicionada ao SQL por que a cláusula WHERE não pode ser utilizada com funções de agregação, abaixo um exemplo de sua sintaxe:

SELECT nome_coluna, nome_coluna
FROM nome_tabela
WHERE nome_coluna condição valor_operador
GROUP BY nome_coluna
HAVING função_agregação(nome_coluna) operador valor ESC;

1.7 COMANDOS DML

Os comandos DML conforme descritos anteriormente tem por características a manipulação dos dados de banco de dados relacional, suas principais cláusulas são descritas a seguir.

1.7.1 INSERT

O *INSERT* tem por funcionalidade a inserção de dados em uma tabela e colunas determinadas, abaixo um exemplo de sua sintaxe:

INSERT INTO tabela
(coluna1, coluna2, ..., coluna N)
VALUES(valor1, valor2, ..., valor N);

1.7.2 DELETE

O *DELETE* tem por funcionalidade a deleção de dados em uma tabela e colunas determinadas ou de todo o conteúdo da mesma, abaixo um exemplo de sua sintaxe:

DELETE FROM tabela
[**WHERE** condição busca];

1.7.3 UPDATE

O *UPDATE* tem por funcionalidade a atualização de dados em uma tabela e colunas determinadas, abaixo um exemplo de sua sintaxe:

UPDATE tabela
SET coluna = expressão
[**WHERE** condição busca];

1.8 COMANDOS DDL

Os comandos DDL conforme descritos anteriormente tem por características a definição da estrutura dos dados de um banco de dados relacional, suas principais clausulas são descritas a seguir.

1.8.1 CREATE

O *CREATE* tem por funcionalidade a definição e criação de um objeto no banco de dados, abaixo um exemplo de sua sintaxe na criação de uma tabela:

CREATE TABLE nome_tabela*(*
nome_coluna1 tipo_de_dado(tamanho),
nome_coluna2 tipo_de_dado(tamanho));

1.8.2 DROP

O *DROP* tem por funcionalidade a deleção de um objeto no banco de dados, abaixo um exemplo de sua sintaxe na deleção de uma tabela:

DROP TABLE table_name;

1.8.3 ALTER

O *ALTER* tem por funcionalidade a alteração de um objeto no banco de dados, abaixo um exemplo de sua sintaxe na adição de uma coluna em uma tabela:

ALTER TABLE table_name
ADD column_name datatype

2. OTIMIZAÇÃO DE CÓDIGO SQL

Conforme descrito por Nossov, Ernst e Chupis (2016) a prática do *tuning* atua em três principais frentes, sendo a primeira focada no desenvolvedor de sistemas, objeto de estudo deste trabalho, a segunda no administrador de banco de dados e por fim foco no próprio Sistema Gerenciador de Banco Dados (SGBD).

A melhoria de performance pode ser implantada em diversos momentos de um projeto de software, podendo ser iniciada com boas práticas do desenvolvedor, e reaparecer ao longo do tempo graças a necessidade de manutenção ou desenvolvimento de um projeto.

O foco proposto ao administrador de banco, referente ao tuning, é de monitorar e garantir um uso eficiente dos recursos computacionais disponíveis. Sendo portanto o foco de tuning do SGBD a aplicação de *upgrades* de software e melhoria no hardware.

Outro fator importante é o acompanhamento de novas funcionalidades conforme o lançamento de novos releases do SGBD as quais comumente lançam

soluções proprietárias para a melhoria de desempenho como novas funções ou ferramentas integradas.

2.1. Execução de um comando SQL

A execução de um comando SQL no Oracle, segundo Chan (2016), é dada de forma sistemática por etapas as quais visam validar desde a sintaxe do comando SQL até a própria execução do comando em questão. A figura 5 apresenta o fluxo de execução conforme a documentação da Oracle:

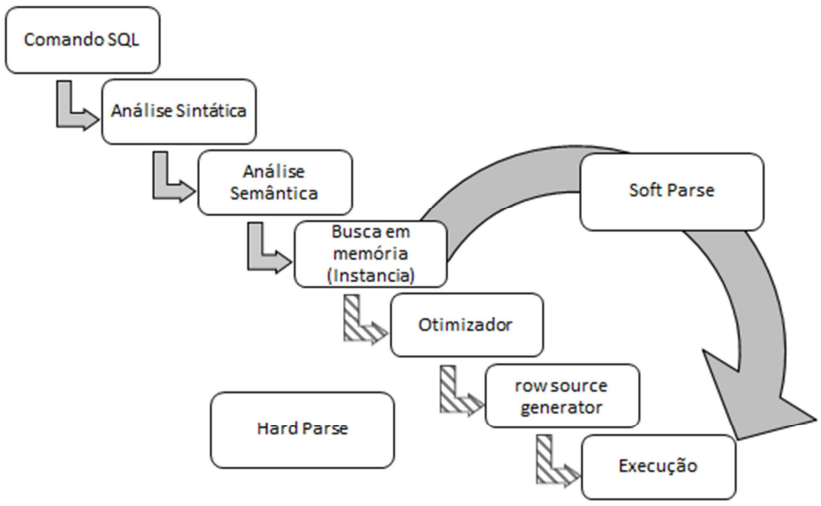


Figura 5: Representação execução comando SQL no Oracle
FONTE: Autoria própria

O processo se inicia com a escrita do comando SQL o qual se deseja executar, tendo como sequência o pedido ao SGBD a execução do comando, após a solicitação ser efetuada a primeira etapa processada é análise sintática a qual vai buscar erros na sintaxe SQL do comando. A figura 6 apresenta um erro na sintaxe na palavra chave *FROM* de um comando:

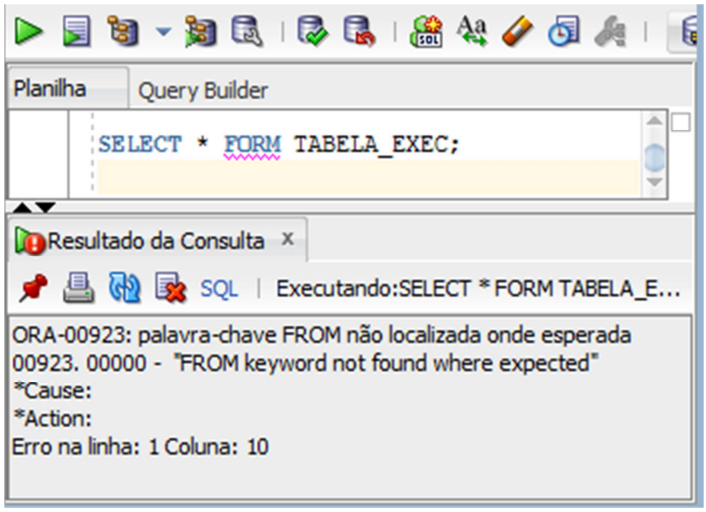


Figura 6: Erro sintaxe
FONTE: Autoria própria

Após a sintaxe ser aprovada pelo SGBD a próxima etapa diz respeito à análise semântica a qual busca um significado ao comando sintaticamente aprovado, nesta fase é verificado, por exemplo, a existência dos objetos referenciados no dicionário de dados. A imagem 7 demonstra um erro de análise semântica:

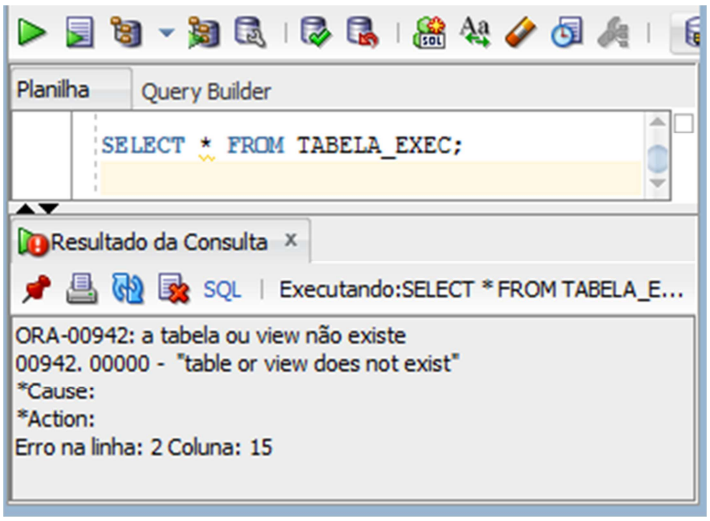


Figura 7: Erro semântico
FONTE: Autoria própria

Conforme pode se observar confrontando as imagens 6 e 7 um comando o qual possui analise sintática aprovada pode não ser aprovada semanticamente, no caso apresentado por não existir a tabela referenciada no BD.

Após obter sucesso nas duas etapas anteriores o SGBD inicia o processo de PARSE, que consiste em uma análise mais profunda ao que será executado buscando utilizar a menor quantidade de recursos possível, ou mesmo obtendo a melhor performance destes.

Neste momento a execução passa a ter a possibilidade de duas formas diferentes. Sendo elas, o *SOFT PARSE* e o *HARD PARSE*, onde o Oracle busca o comando solicitado em sua memória de execução e caso encontre uma execução prévia realiza um reutiliza *SOFT PARSE* reaproveitando o trabalho de análise anterior o qual já executou nas etapas otimização e *ROW SOURCE GENERATION*.

Caso não encontre correspondência em memória, um HARD PARSE é iniciado e as etapas de otimização e *ROW SOURCE GENERATION* são iniciadas. A etapa de otimização consiste em uma análise para cada comando DML existente no pedido de execução gerando assim um equivalente executável ao SGBD. A figura 8 demonstra a execução de um comando SQL e seu respectivo plano de execução:

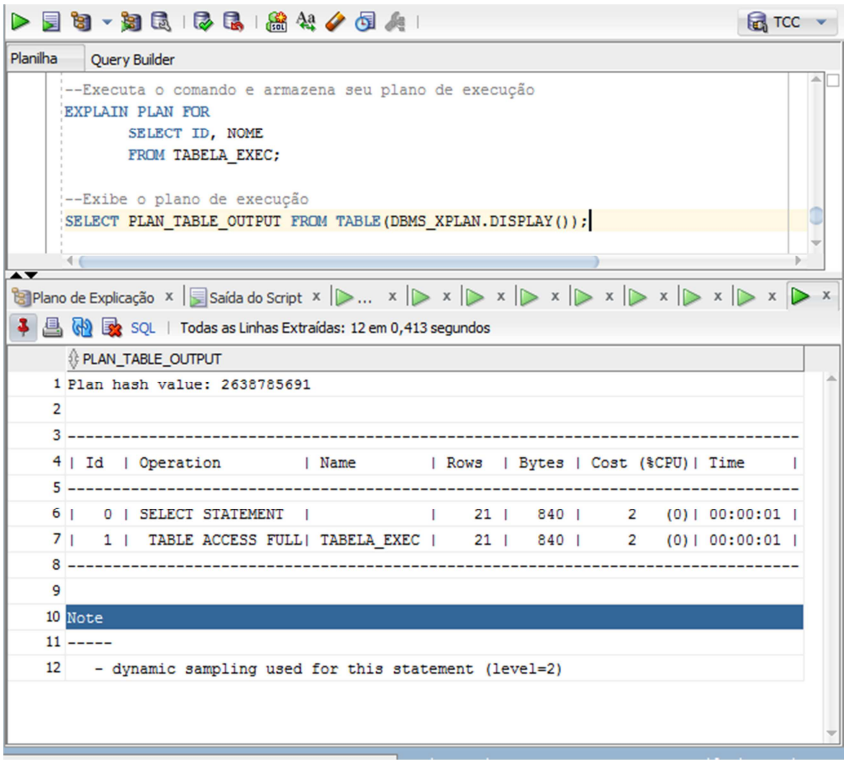


Figura 8: Execução de um plano de execução
FONTE: Autoria própria

Conforme exibido na figura 8 pelo plano de execução, o SGBD identifica a operação efetuada por tabela e descreve de forma detalhada o número de linhas afetadas assim como os custos de processamento e tempo necessários.

2.2. Indices

Segundo a documentação da Oracle (Indexes and Index-Organized Tables, 2017) um índice pode ser definido como uma estrutura opcional associada a uma tabela, a qual às vezes pode acelerar o acesso aos dados. Os índices são um dos meios capazes de reduzir a leitura e escrita em disco. Permitindo ao criar um índice, em uma ou mais colunas de uma tabela, a capacidade de recuperar um pequeno conjunto de linhas distribuídas aleatoriamente em uma tabela.

Quando realizada uma consulta em uma tabela a qual não possui índices, o banco necessita efetuar uma busca completa para localizar um valor. Analogamente quando populamos uma tabela os dados são adicionados de forma aleatória e uma

busca numa planilha de 1000 linhas o qual retorna um resultado contido na linha 500 necessita de uma busca completa na tabela, sendo o desempenho afetado toda vez que mais linhas forem adicionadas.

De forma geral a criação de índices tende a ser mais bem aplicada quando:

- A coluna ou conjunto de linhas são consultados frequentemente;
- Existe uma restrição de integridade referencial na coluna;
- Existe uma restrição de integridade de chave UNIQUE na coluna.

Os índices são objetos representados lógica e fisicamente independentes dos dados com os quais eles estão associados, podendo então um índice ser criado ou removido sem afetar física ou logicamente uma tabela. Tendo como único ponto de impacto a remoção de um índice a possibilidade de uma consulta que o utilizava anteriormente ter uma queda de desempenho.

A existência ou não de um índice não requer alterações nas instruções SQL de consulta, sendo o índice somente uma forma de rápido acesso a uma única linha de dados. Influenciando apenas na velocidade de execução, onde em um valor de dados indexado, o índice aponta diretamente para o local das linhas os quais possuem os valores.

O banco de dados utiliza e mantém automaticamente os índices criados. Dando também manutenção no índice de forma automática quando acontecem alterações nos dados como exclusão, adição e atualização de linhas de uma tabela.

O desempenho de busca em uma tabela a qual possui índices permanece constante mesmo havendo a adição de mais dados, porém uma tabela a qual possua muitos índices pode gerar uma queda de desempenho nos comandos DML pela necessidade de atualizar os índices.

As figuras seguintes exemplificam a utilização de um índice e a alteração no plano de execução causada pelo mesmo. A figura 9 demonstra a criação de duas

tabelas as quais é utilizada uma coluna da primeira tabela como chave estrangeira da segunda tabela:

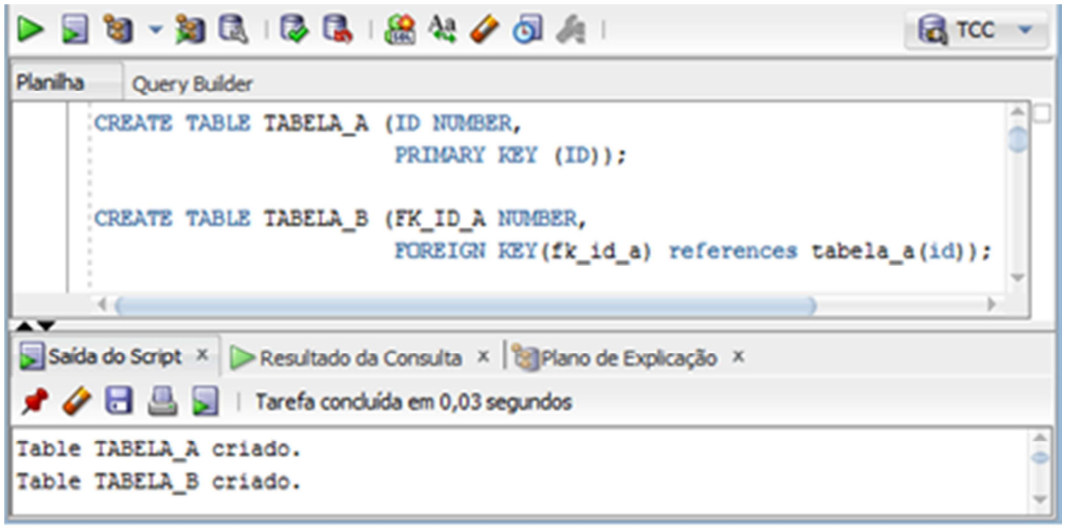


Figura 9: Criação de tabelas dependentes
FONTE: Autoria própria

A figura 10 demonstra a inserção de 8 valores na primeira tabela:

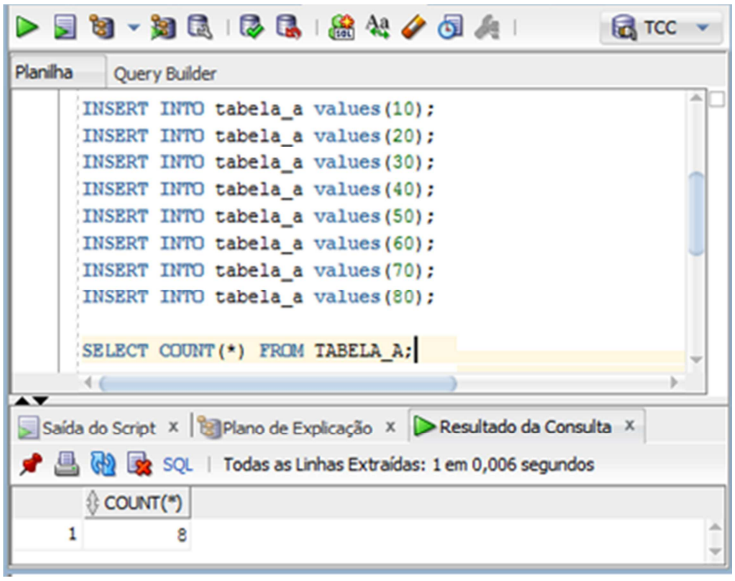


Figura 10: inserção de valores tabela A
FONTE: Autoria própria

A figura 11 demonstra a inserção de 1524 valores aleatórios baseados nos valores da primeira tabela:

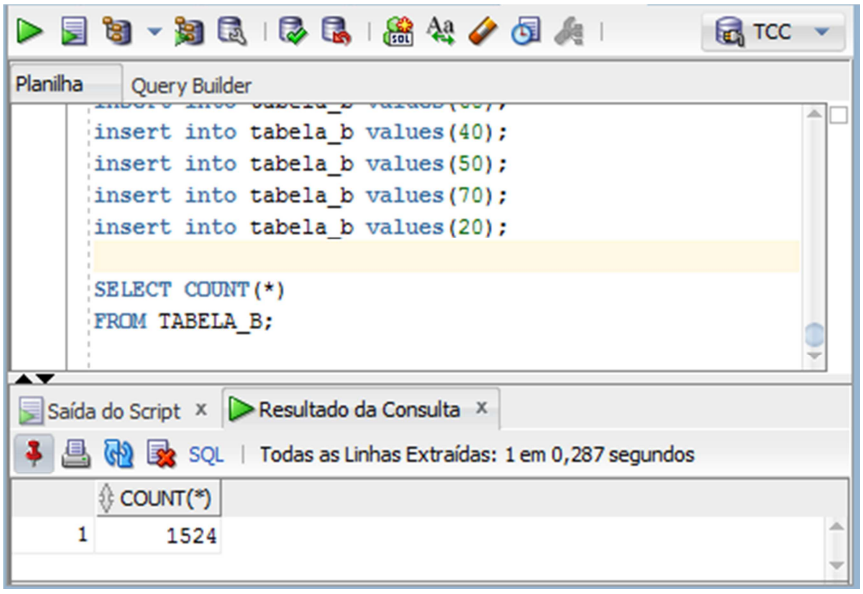


Figura 11: Inserção de valores tabela B
FONTE: Autoria própria

A figura 12 demonstra o plano de execução de um consulta a qual busca todos os registros com valor igual a 10, sem a utilização de um índice:

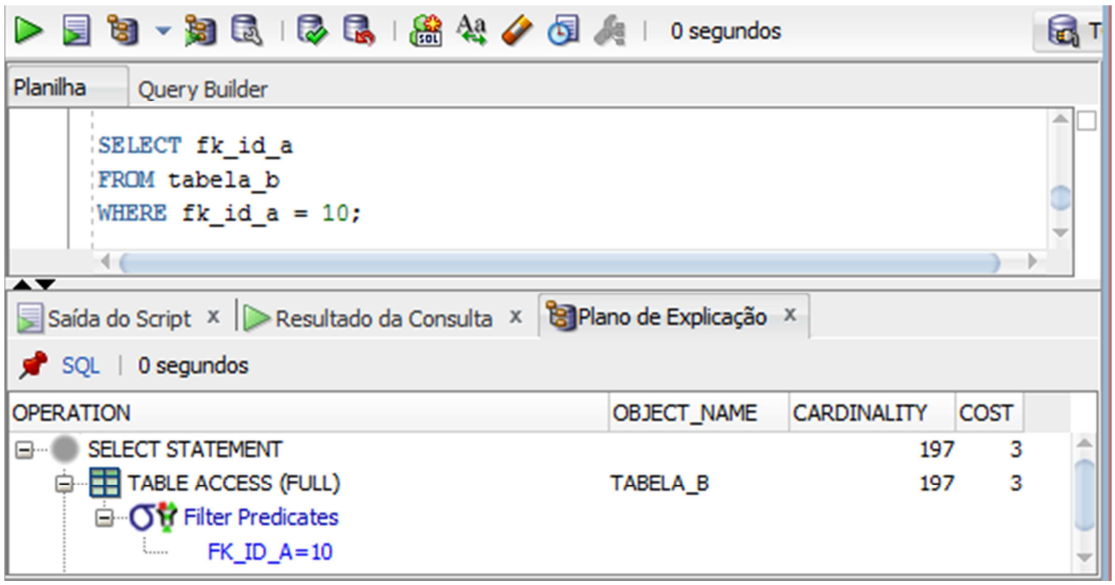


Figura 12: Consulta plano de execução sem utilização de índice
FONTE: Autoria própria

Por não possuir um índice a consulta tem a necessidade de efetuar uma leitura completa na tabela para encontrar os resultados. A figura 13 demonstra a criação de um índice na coluna a qual estão sendo buscados os valores na tabela B:

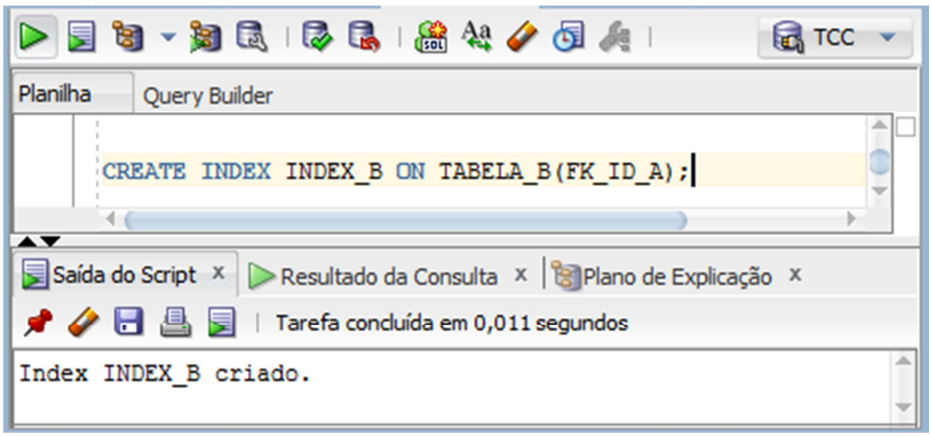


Figura 13: Criação de índice na tabela B
FONTE: Autoria própria

Na figura 14, após a criação do índice a mesma consulta é executada novamente no formato de plano de execução:

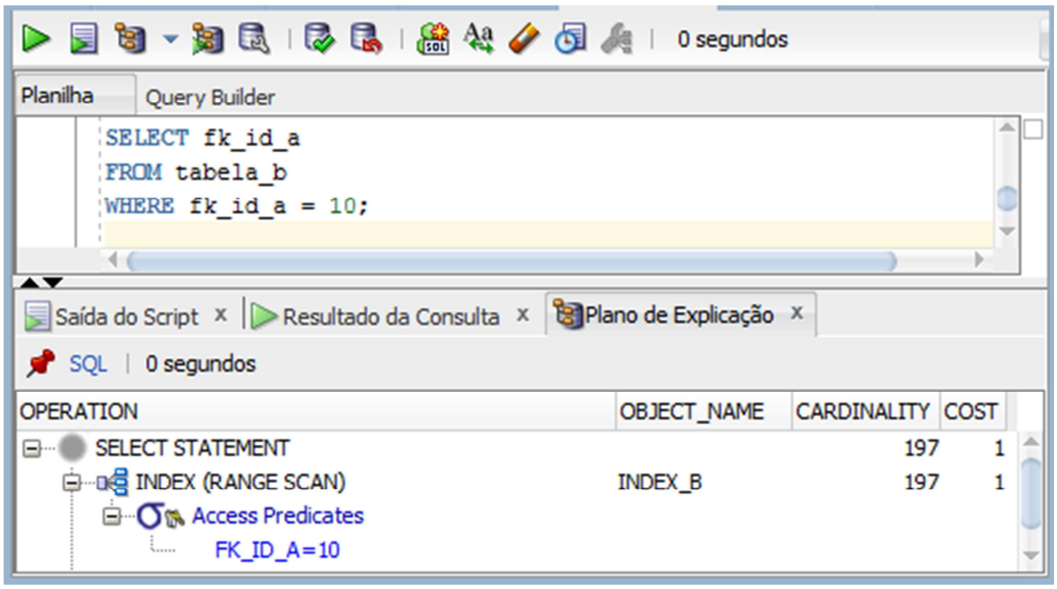


Figura 14: Execução do plano de execução com índice
FONTE: Autoria própria

A criação do índice na tabela B gerou uma alteração no plano de execução da consulta, assim como na redução do custo o qual era antes necessário para

execução, de custo 3 para custo 1, auxiliando assim a performance da consulta e do SGBD .

2.3. Massa de Dados

Para geração da massa de dados para popular, as tabelas utilizadas, foi criada uma planilha de Excel para geração aleatória de valores do tipo texto e numérico. Permitindo assim a geração de massas de dados elaboradas com grande tamanho e complexidade. A figura 15 mostra de forma reduzida o método utilizado para criação de 10 valores:

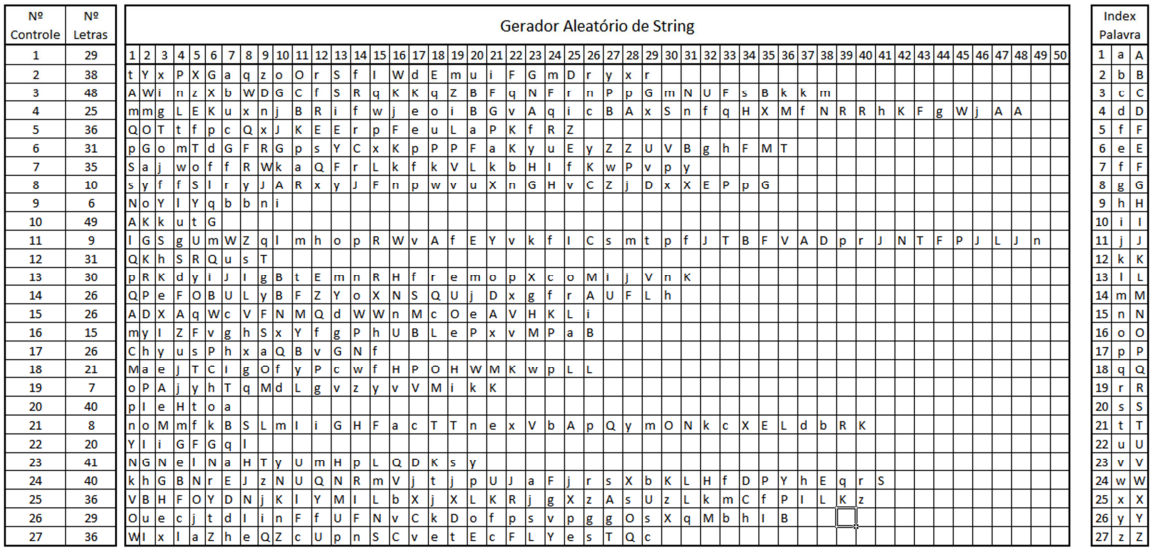


Figura 15: Planilha de geração aleatória
FONTE: Autoria própria

Como critério base foram consideradas palavras de tamanho mínimo de 3 letras e máximo de 50, a primeira coluna de nome ‘Nº de controle’ tem como objetivo facilitar a contabilização do número de palavras geradas, a coluna ‘Nº Letras’ gera aleatoriamente um tamanho para a palavra entre 3 e 50, conforme a figura 16:

Nº Controle	Nº Letras
1	49
2	=ALEATÓRIOENTRE(4;50)
3	32

Figura 16: Fórmula para definição de tamanho de palavra
FONTE: Autoria própria

A coluna 'Index Palavra' se refere à letra que será retornada sendo representadas por uma coluna minúscula reapresentada de forma maiúscula a direita. A seleção entre maiúscula e minúscula também e dada de forma aleatória nas fórmulas utilizadas nas colunas numeradas, sendo o 2 para minúsculo e 3 para maiúsculo onde se referem ao número da coluna da tabela em questão, conforme figura 17.

Index Palavra		
1	a	A
2	b	B
3	c	=MAIÚSCULA(BE5)
4	d	MAIÚSCULA(texto)
5	f	F

Figura 17: Índice de caracteres
FONTE: Autoria própria

As colunas numeradas de 1 a 50 se referem a posição a própria palavra, sendo as colunas de 1 a 3 geradas obrigatoriamente e as de 4 a 50 de forma lógica validando o tamanho atribuído a palavra.

Conforme apresentado nas figuras 18 e 19:

Nº Controle	Nº Letras	Gerador Alea																							
1	33	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
2	16	J	j	=PROCV(ALEATÓRIOENTRE(1;27);$BD$3:$BF$29;ALEATÓRIOENTRE(2;3);0)																						

Figura 18: Fórmula definida palavras de três caracteres
FONTE: Autoria própria

Nº Controle	Nº Letras	Gerador Aleatório de String																														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	3
1	48	W	I	j	S	B	O	V	F	A	s	c	w	j	I	R	E	M	y	s	G	I	f	B	P	Z	s	F	I	Q	K	X	O	G
2	8	x	Y	u	w	Z	J	g	B	=SE($C4>L$3;PROCV(ALEATÓRIOENTRE(1;27);$BD$3:$BF$29;ALEATÓRIOENTRE(2;3);0);"")																											
3	12	S	k	b	b	L	x	c	C	z	C	G	f																					
4	27																																	

Figura 19: Fórmula definida palavras de mais de três caracteres
FONTE: Autoria própria

Finalizando os valores foram concatenados com a função disponibilizada pelo Excel, seguindo a sintaxe do comando insert sendo em seguida executado para popular o banco.

2.4. SQL Developer e versão Oracle

O SQL Developer é um ambiente de desenvolvimento integrado disponibilizado gratuitamente pela Oracle como uma solução gráfica e gratuita que permite conexão de usuários e administradores de banco de dados ao banco de dados e tem por objetivo a facilitar a vida do usuário final.

A versão utilizada neste trabalho 4.1.3.20 suporta as versões de 10g, 11g e 12c do Sistema Gerenciador de Banco Dados (SGBD) da Oracle, sendo utilizado neste trabalho a 11g release 11.2.0.2.0 Express Edition. Esta versão de SGBD por ter finalidade didática possui algumas limitações para evitar seu uso comercialmente, como a restrição de uso de somente um core para processamento, um giga de memória *ram* e onze gigas para armazenamento em disco.

2.5. Colunas virtualizadas

Conforme Niemic (2015) colunas virtualizadas demonstram resultados baseados em dados gravados em disco e tem por objetivo transmitir sentido a regras de negócio. Tem por objetivo reduzir impactos no desempenho e complexidade na implementação, por não existirem fisicamente junto à estrutura da tabela são armazenadas no dicionário de dados do banco.

As colunas virtualizadas podem ser utilizadas para cálculos ou mesmo operações lógicas podendo se basear em dados existentes a tabela a qual pertence, constantes e até mesmo funções criadas do banco de dados.

Abaixo um exemplo de sua sintaxe básica:

Nome_Coluna [*datatype*] *[GENERATED ALWAYS] AS* (*expression*) [*VIRTUAL*]

Algumas restrições são dadas a este tipo de coluna como a não possibilidade de referência à outra coluna que também seja virtualizada, não poder ter um valor explicitamente declarado através de inserção, não podem se basear diretamente a colunas de outras colunas e por fim não podem retornar alguns tipos de dados como *LOB* ou *LONG RAW* além de tipos definidos pelo usuário.

A figura 20 apresenta um exemplo da criação de uma tabela a qual possui uma coluna virtual responsável por uma operação aritmética:

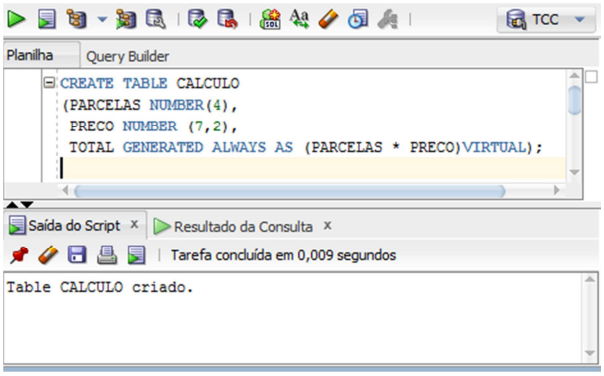


Figura 20: Criação de coluna virtual
FONTE: Autoria própria

Para melhor visualização da estrutura da tabela 'CALCULO' a figura 21 apresenta o resultado de sua estrutura segundo pesquisa realizada em uma *view* a qual auxilia os administradores de banco de dados neste propósito:

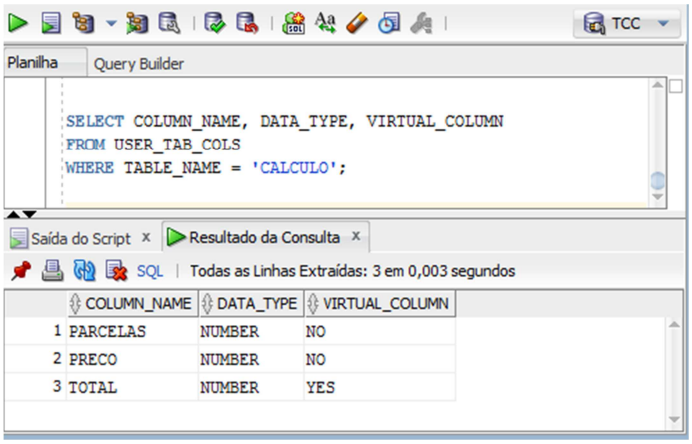


Figura 21: Exibição de estrutura de coluna virtual
FONTE: Autoria própria

Conforme o retorno da consulta executada, confirmamos que a coluna 'TOTAL' é do tipo virtualizada e tem por padrão atribuído um tipo de dado numérico atribuído, mesmo não tendo sido um tipo atribuído a ela anteriormente.

Após sua criação a tabela recebeu a inserção de 10.000 registros com valores aleatórios conforme demonstrado anteriormente e tem por fim seu conteúdo consultado, conforme figuras 22 e 23:

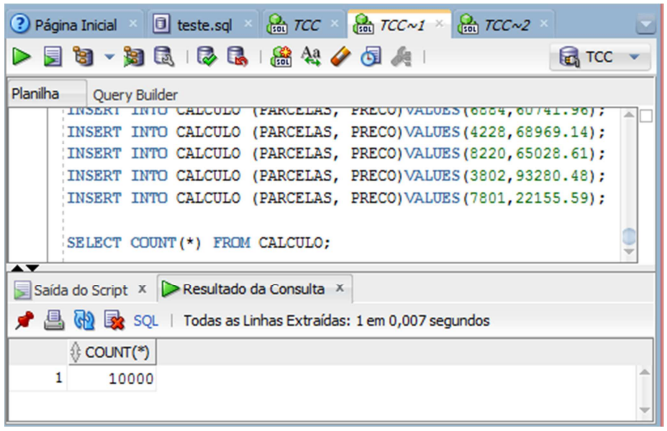


Figura 22: Inserção de registros
FONTE: Autoria própria

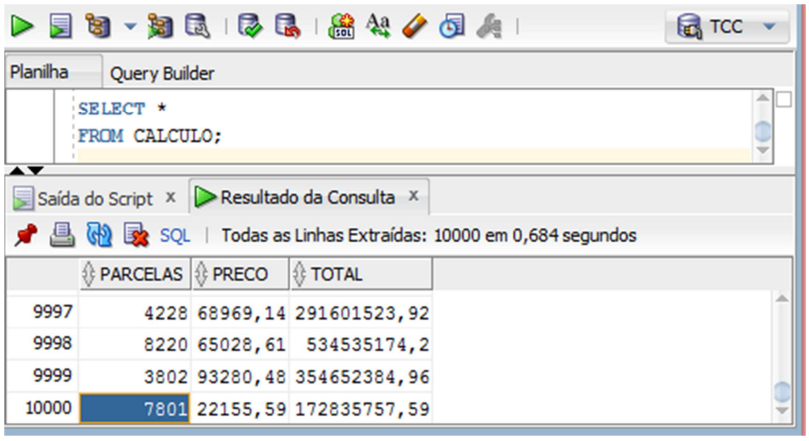


Figura 23: Execução de busca com valores calculados virtualmente
FONTE: Autoria própria

Se efetuarmos uma consulta na mesma tabela efetuando a mesma operação aritmética percebe-se que o tempo de execução é menor do que utilizando as virtualização de colunas, isto ocorre pela necessidade de maior processamento na execução deste tipo de coluna, sendo então seu uso recomendado para solução de redução de volume de dados armazenados e volumes de dados não muito extensos quando se trata de operações aritméticas.

Conforme figura 24:

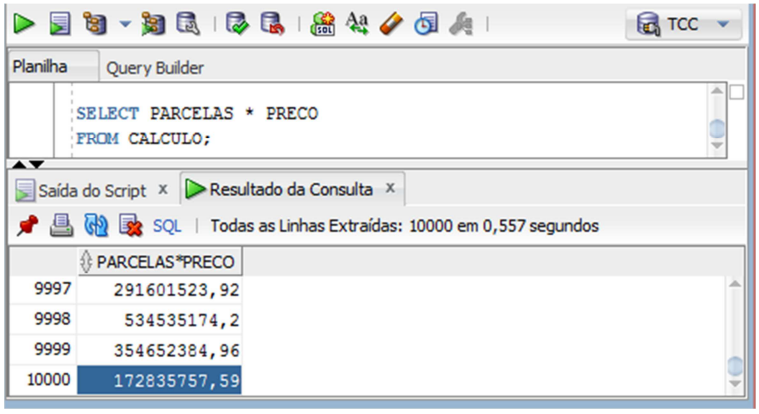


Figura 24: Execução de busca com valores calculados pelo comando
FONTE: Autoria própria

Por outro lado é possível uma melhoria de performance quando se utilizam colunas virtualizadas como substitutas a outras necessidades como a inserção de valores por *triggers*, figura 25, o exemplo a seguir propõe a resolução de uma classificação referente aos dados da coluna 'ENDIVID_TOTAL'.

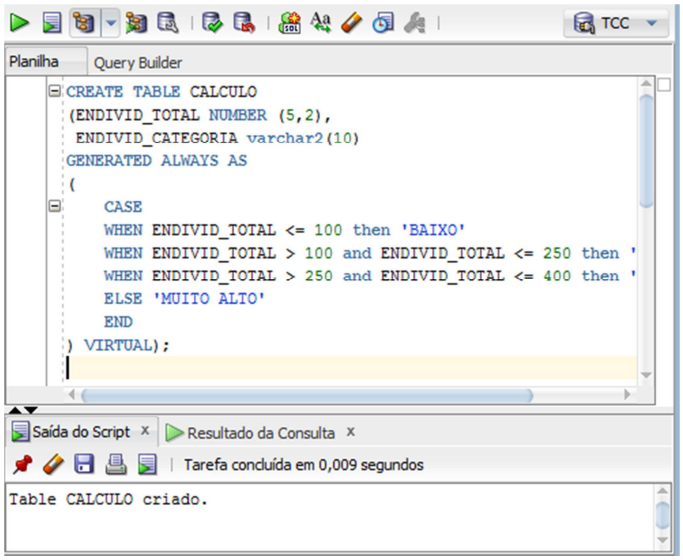


Figura 25: Criação de coluna virtual com função lógica
FONTE: Autoria própria

Após a inserção e consulta de valores podemos observar que a coluna 'ENDIVID_CATEGORIA' possui uma execução lógica através de um case o qual efetua uma categorização por faixas de valores, a utilização de colunas virtuais pode vir a substituir neste caso não somente a inserção de dados pela execução de uma trigger como também facilita na manutenção da regra aplicada a categorização, onde ao invés de atualizar todos os dados do banco tem-se sua manutenção focalizada na coluna em questão. Figura 26:

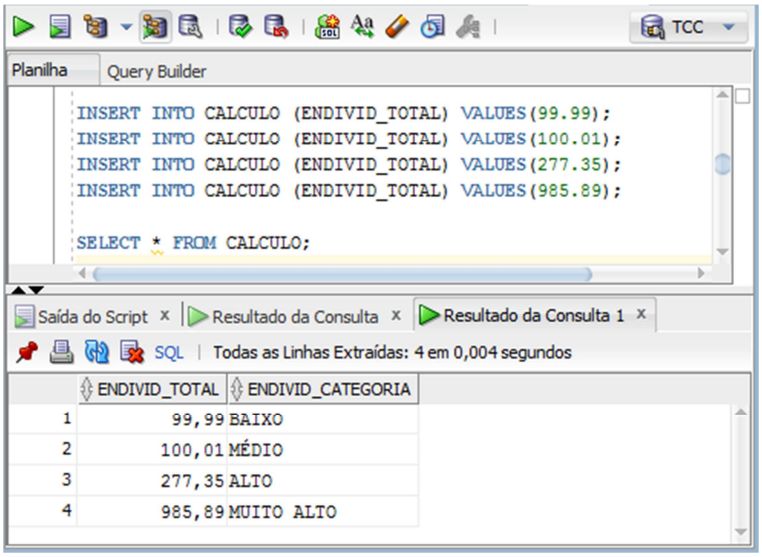


Figura 26: Inserção de valores
FONTE: Autoria própria

Por fim é demonstrada na figura 27 a possibilidade de utilização de uma coluna virtualizada como índice a uma tabela permitindo assim a possibilidade de estratégias de melhoria de desempenho conforme demonstrado anteriormente.

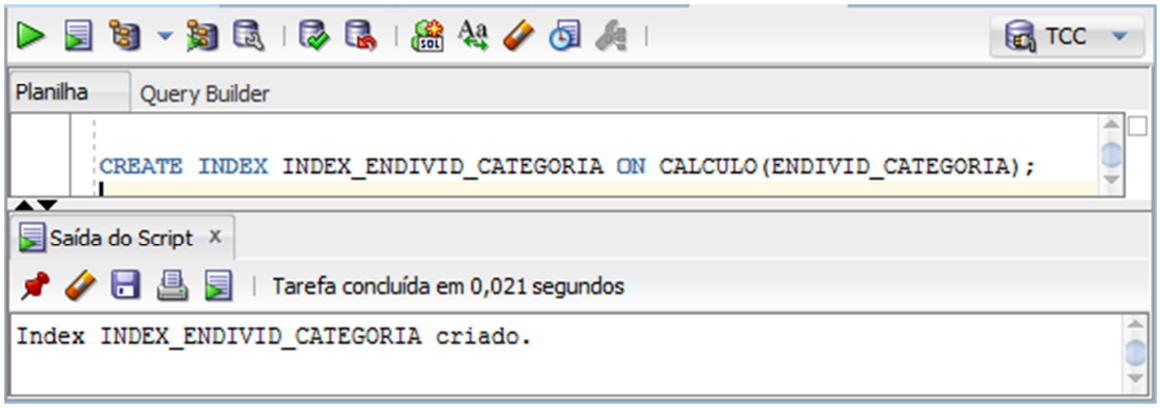


Figura 27: Criação de índice em coluna virtualizada
FONTE: Autoria própria

2.6. INDICADORES DE PERFORMANCE

Os Sistema Gerenciador de Banco Dados (SGBD) geralmente possuem nativamente alguns facilitadores para o acompanhamento da performance do banco, no caso da solução Oracle os administradores do banco de dados possuem um conjunto de *views* para acompanhar desde o desempenho de memória ao consumo de recursos de queries em execução.

Conforme apresentado por Niemiec (2014) é possível através deste tipo de recurso podemos ranquear as consultas menos performáticas e assim nortear possíveis pontos críticos de performance para um estudo de viabilidade de melhoria de performance. Abaixo temos um exemplo de uma consulta a qual utiliza das *views* V$SQLAREA para listar as consultas com pior desempenho e precisam ser otimizadas:

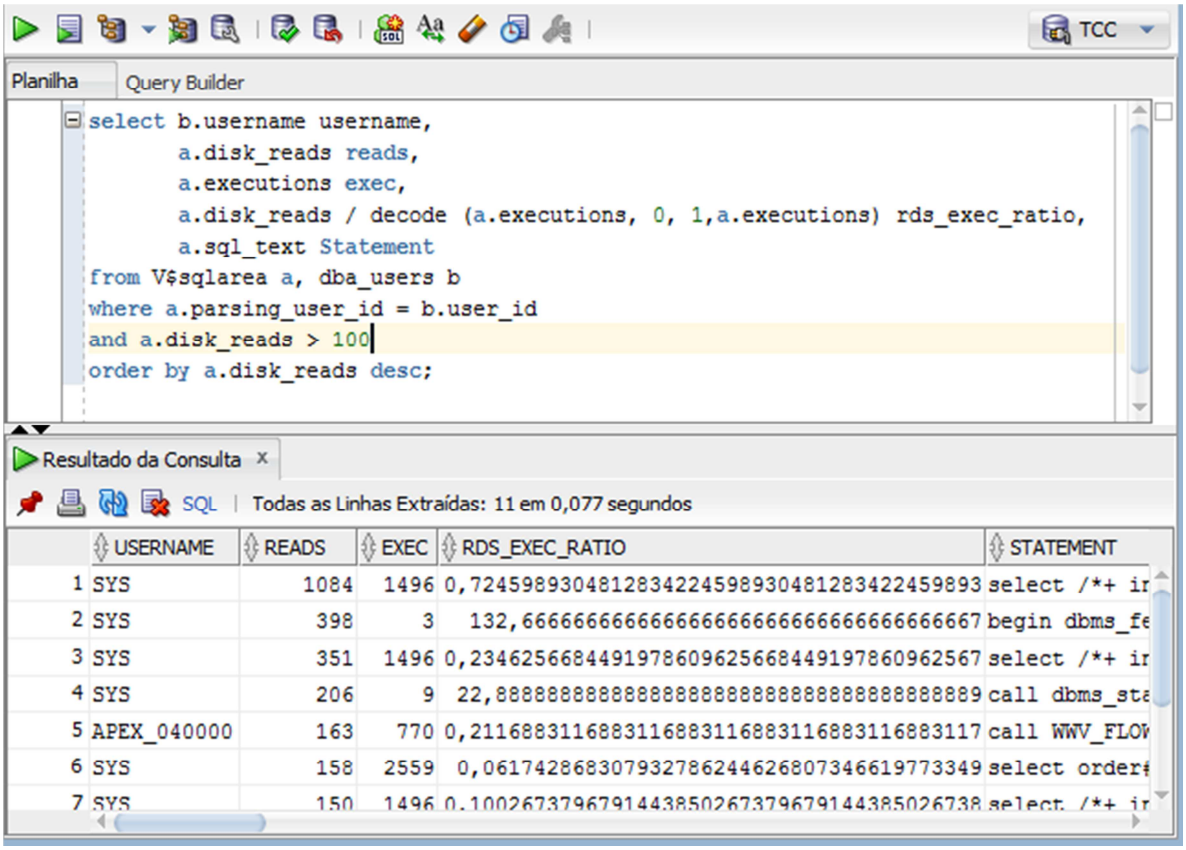
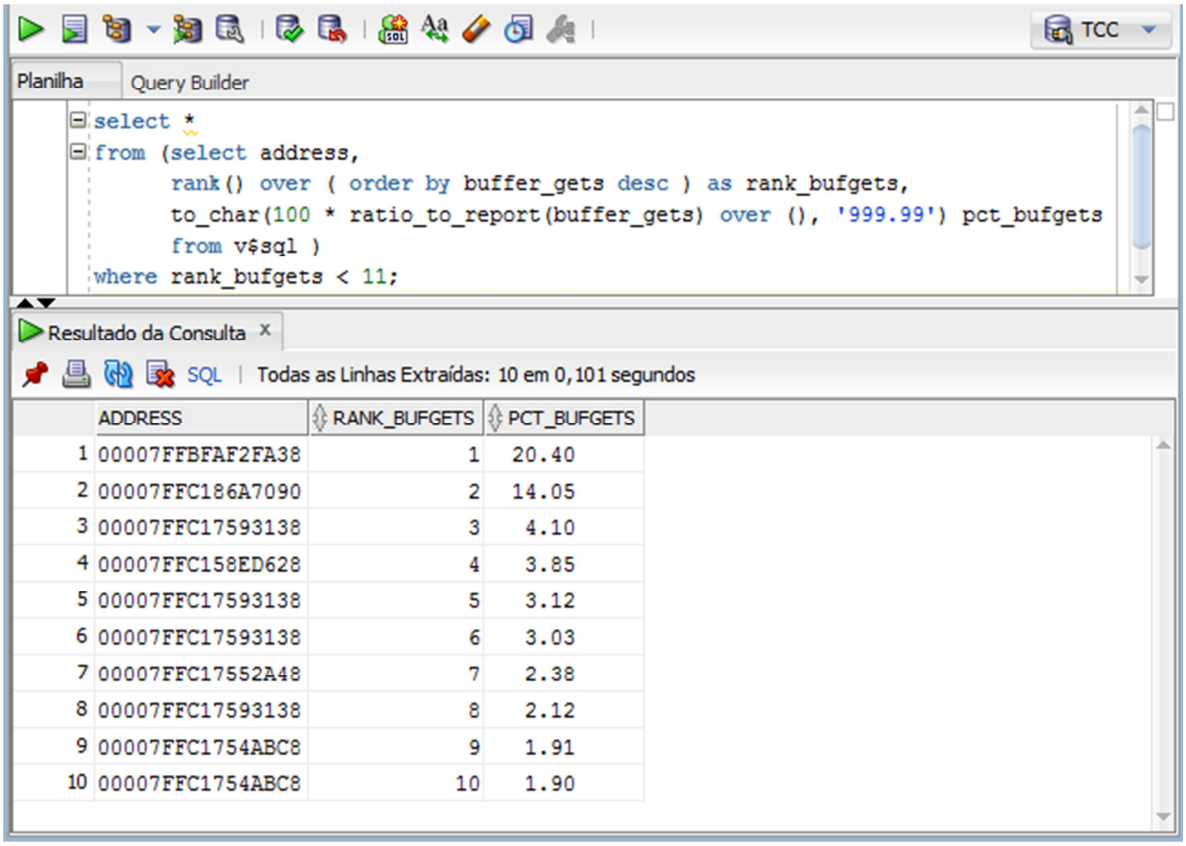


Figura 28: Consultas com maior quantidade de leituras em disco
FONTE: Autoria própria

No exemplo a coluna READS apresenta o volume de leituras de disco que estão sendo executados pelo sistema, sendo o exemplo delimitado pela cláusula AND em volumes maiores que 100 leituras somente para exemplificação, sendo a sugestão dada por Niemiec (2014) volumes acima de 100000.

O exemplo, da figura 29, apresenta de forma ranqueada as instruções que mais necessitam de memória para sua execução, sendo o parâmetro para organização a coluna BUFFER_GETS:



	ADDRESS	RANK_BUFGETS	PCT_BUFGETS
1	00007FFBFAF2FA38	1	20.40
2	00007FFC186A7090	2	14.05
3	00007FFC17593138	3	4.10
4	00007FFC158ED628	4	3.85
5	00007FFC17593138	5	3.12
6	00007FFC17593138	6	3.03
7	00007FFC17552A48	7	2.38
8	00007FFC17593138	8	2.12
9	00007FFC1754ABC8	9	1.91
10	00007FFC1754ABC8	10	1.90

Figura 29: Consultas com maior necessidade de recurso de memória
FONTE: Autoria própria

Com o auxilio destas duas colunas consegue-se a visão de maiores custos referentes a consulta em disco e memória utilizada pelo SGBD, estes junto ao tráfico de rede compõem em grande maioria os principais gargalos sendo assim focos de otimização.

3. CONSIDERAÇÕES FINAIS

O presente trabalho trouxe como proposta a melhoria de desempenho na utilização de bancos de dados relacionais através de uma melhor codificação. Buscando este entendimento foi realizada pesquisa teórica com a finalidade de compreender o objeto de estudo desde seu contexto histórico, origem do modelo relacional e o surgimento da linguagem SQL e sua sintaxe, assim como a compreensão de como é dado o processamento dos comandos SQL por um SGBD Oracle, e por fim ferramentas as quais podem trazer melhoria de desempenho.

Seguindo então a estrutura deste trabalho um primeiro passo para melhoria de codificação é a obtenção de conhecimento da linguagem SQL e suas funcionalidades, sendo a linguagem um influenciador direto no desempenho ainda revisada e atualizada pelo ANSI - *American National Standard Institute* visando avanços em diversos aspectos.

Um segundo passo para a melhoria de desempenho é a compreensão da forma a qual é dada a execução do comando SQL pelo SGBD, sendo uma ferramenta facilitadora, para isso o plano de execução o qual permite a visualização da forma a qual a execução está sendo realizada. Tendo por exemplo grande efetividade a utilização das informções trazidas pelo plano de execução junto a utilização de índices em muitos casos.

Por fim foram apresentados algumas soluções da Oracle para melhoria de desempenho, como colunas virtuais e indicadores de performance, o que ressalta a necessidade de conhecimento, além do SQL baseado no padrão ANSI de soluções as quais compõe a solução utilizada, tendo cada SGBD propriedades que buscam facilitar seu uso e aumentar o desempenho.

De forma a prover o desenvolvimento prático deste trabalho utilizou-se a versão Oracle 11G Express Edition, do SGBD, e como interface gráfica o Oracle SQL Developer em sua versão 4.1.3.20.

A melhoria de desempenho é necessária graças ao aumento constante no volume de dados produzido, os quais necessitam ser armazenados e consultados também em grande velocidade. O surgimento de novos serviços online e automatizados somados a popularização da internet e dispositivos móveis estão diretamente ligados a este aumento.

Bases de dados tornaram-se um ativo precioso das organizações, as quais cada vez mais buscam utilizar estes dados de forma apoiar situações diversas. Sendo muitas vezes utilizadas não somente com a finalidade de armazenamento de dados, mas também como fonte provedora de informações as quais auxiliam outras atividades como tomada de decisões no meio empresarial, utilização atividades de marketing ativo.

Sendo a necessidade o simples armazenamento ou consulta de dados para fins diversos, o desempenho nestas atividades é uma constante. E sendo a codificação um momento propicio para esta preocupação por se tratar de muitas vezes de um momento inicial.

Como trabalhos futuros, poderia-se efetuar um estudo mais aprofundado sobre planos de execução e a influência de índice.

REFERÊNCIAS

CHAN, Immanuel; ASHDOWN, Lance. **Oracle® Database:** Performance Tuning Guide 11g Release 2 (11.2). 2014. Disponível em: <http://docs.oracle.com/cd/E11882_01/server.112/e41573/toc.htm>. Acesso em: 20 nov. 2016.

Codd's 12 Rules. Disponível em: <https://www.tutorialspoint.com/dbms/dbms_codds_rules.htm>. Acesso em: 03 nov. 2016.

Database SQL Tuning Guide. Disponível em: <https://docs.oracle.com/database/121/TGSQL/toc.htm> Acesso em: 02 abr. 2017.

FEHILY, Chris. **Visual QuickStart Guide SQL.** 3. ed. Berkeley: Peachpit Press, 2008. 483 p.

Indexes and Index-Organized Tables. Disponível em: < https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm#CNCPT721> Acesso em: 18/05/2017.

MACHADO, Felipe Nery Rodrigues; ABREU, Maurício Pereira de. **Projeto de banco de dados: uma visão prática.** 17. ed. São Paulo: Editora Érica, 2012.

NIEMIEC, Rich. **Quick Start Guide to Oracle Query Tuning::** Tips for DBAs and Developers. 129. ed. New York: Mcgraw-hill Education, 2014.

NOSSOV, Leonid; ERNST, Hanno; CHUPIS, Victor. **Formal SQL Tuning for Oracle Databases:** Practical Efficiency - Efficient Practice. Dusseldorf: Springer, 2016. 109 p.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. **Sistema de banco de dados.** Rio de Janeiro: Elsevier, 2006.

SQL Tutorial. 2011. Disponível em: < http://www.w3schools.com/sql/default.asp>. Acesso em: 29 nov. 2016.

TAYLOR, Allen G.. **SQL For Dummies.** 8. ed. Hoboken: John Wiley & Sons, Inc., 2008.

Volume de dados no mundo mais que dobra a cada dois anos. 2011. Disponível em: <http://computerworld.com.br/tecnologia/2011/06/28/volume-de-dados-no-mundo-mais-que-dobra-a-cada-dois-anos>. Acesso em: 29 nov. 2016.

What is SQL Developer?. Disponível em: <http://www.oracle.com/technetwork/developer-tools/sql-developer/what-is-sqldev-093866.html> Acesso em: 08 mai. 2017.