UNIVERSIDADE DE CAXIAS DO SUL
Centro de Ciências Exatas e Tecnologia
Curso de Bacharelado em Ciência da Computação

Enor Paim

DESENVOLVIMENTO DE UM ANALISADOR DE CUSTO DE PL/SQL

Caxias do Sul

2008

Enor Paim

DESENVOLVIMENTO DE UM ANALISADOR DE CUSTO DE PL/SQL

Trabalho de Conclusão de Curso para obtenção do Grau de Bacharel em Ciência da Computação da Universidade de Caxias do Sul.

Helena Grazziotin Ribeiro
Orientador

Caxias do Sul

2008

Este trabalho é dedicado
à todas as pessoas que, durante todos estes anos,
acreditaram e torceram por mim.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, que me deram todo o apoio e estrutura necessários para que eu pudesse chegar até aqui.

Agradeço a todos os professores que ao longo desta jornada contribuíram de alguma forma para o meu desenvolvimento pessoal e profissional, em especial à professora Helena Grazziotin Ribeiro, que me orientou durante o desenvolvimento deste trabalho.

Agradeço a todos meus amigos, que estiveram ao meu lado durante esta longa caminhada.

E finalmente, mas não menos especial, agradeço a minha amada namorada Cristiane, por me apoiar incondicionalmente nos momentos mais difíceis, com muito amor e carinho.

"A mente que se abre a uma nova idéia
jamais voltará ao seu tamanho original."
Albert Einstein.

RESUMO

Freqüentemente administradores de banco de dados necessitam analisar a performance de execução de uma instrução SQL. Para auxiliar nesta tarefa, o Oracle possui um recurso chamado *Explain Plan*. Com ele é possível analisar detalhadamente o plano de execução de uma instrução SQL, verificando o seu custo de execução e muitas outras informações. Porém, o Oracle não possui nenhum recurso que faça este mesmo processo para um bloco de comandos PL/SQL. Para se analisar o custo de execução de um PL/SQL é necessário percorrer seu código fonte, buscando por todas as instruções SQL existentes e analisando seus planos de execução através do *Explain Plan*. O objetivo deste trabalho é desenvolver um protótipo de *software* que auxilie nesta tarefa, fazendo a análise do custo de execução de um bloco de comandos PL/SQL, com base no custo de execução de cada instrução SQL contida dentro dele. Para isso, o protótipo implementa um analisador sintático, que faz o reconhecimento de uma gramática definida especificamente para este trabalho, e identifica as instruções SQL contidas no código fonte do PL/SQL. Após identificar todos as instruções SQL, o protótipo utiliza o *Explain Plan* para gerar as informações dos planos de execução e mostra para o usuário, de forma organizada, as informações mais relevantes para a análise de custo de execução do plano.

Palavras-chaves: Analisador, Custo, *Explain Plan*, PL/SQL, SQL, Oracle

ABSTRACT

Often database administrators need to analyze the perfomance of executing a SQL statement. To assist in this task, Oracle has a feature called Explain Plan. With Explain Plan is possible to examine the detailed execution plan for an SQL statement, checking the cost of execution and other information. However, Oracle does not have any resources to do this same process for a block of PL/SQL commands. To analyze the execution cost of a PL/SQL is necessary to check your source code, looking for all existing SQL and analyzing their execution plans by means of Explain Plan. The purpose of this study is to develop a software prototype to help in this task, doing the analysis of the execution cost of a block of commands PL/SQL, based on the cost of executing each SQL statement contained within it. For this, the prototype implements a syntactic analyzer which makes the recognition of a grammar defined specifically for this study, and identifies the SQL statements contained in the source code of PL/SQL. After identifying all the SQL, the prototype uses Explain Plan to generate the information by the execution plans and shows to the user, in an organized way, the most relevant information for the analysis of plan execution cost.

Keywords: Analyzer, Cost, Explain Plan, PL/SQL, SQL, Oracle

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

Sigla	Significado em Português	Significado em Inglês
BNF	Forma Backus-Naur	*Backus-Naur Form*
CBO	Otimização Baseada em Custo	*Cost Based Optimization*
CPU	Unidade Central de Processamento	*Central Process Unit*
DDL	Linguagem de Definição de Dados	*Data Definition Language*
DML	Linguagem de Manipulação de Dados	*Data Manipulation Language*
I/O	Entrada / Saída	*In/Out*
JDBC	Conectividade Java para Bases de Dados	*Java Database Connectivity*
JDK	Kit de Desenvolvimento Java	*Java Development Kit*
JRE	Ambiente de Execução Java	*Java Runtime Environment*
LHS	Lado Esquerdo	*Left Hand Side*
PL/SQL	Linguagem procedural / Linguagem Estruturada de Consulta	*Procedural Language / Structured Query Language*
RHS	Lado Direito	*Right Hand Side*
SGBD	Sistema Gerenciador de Banco de Dados	
SQL	Linguagem Estruturada de Consulta	*Structured Query Language*
TS	Tabela de Símbolos	

Sumário

1 INTRODUÇÃO

O SGBD Oracle possui uma linguagem de programação própria chamada PL/SQL. O PL/SQL é uma linguagem procedural que, além de conter recursos comuns a este tipo de linguagem, também reconhece instruções SQL, o que possibilita ao programador utilizar chamadas SQL em meio ao código procedural [1].

Com o PL/SQL, é possível codificar regras de negócios diretamente dentro do banco de dados, através da criação de objetos, tais como procedimentos, funções, pacotes e gatilhos [1].

Frequentemente administradores de banco de dados necessitam analisar a performance de execução de uma instrução SQL. Para auxiliar nesta tarefa, o Oracle possui um recurso chamado *Explain Plan*. Com ele é possível analisar detalhadamente o plano de execução de uma instrução SQL, verificando o seu custo de execução, ou seja, verificando o seu nível de performance.

Geralmente em equipes de desenvolvimento de *softwares*, os desenvolvedores não possuem um conhecimento aprofundado sobre banco de dados ou até mesmo sobre a modelagem da base de dados para a qual estão escrevendo seus programas. Por este fato, temos uma grande probabilidade de que alguns códigos não sejam escritos da melhor forma possível, tornando-se muito custosos para serem processados pelo SGBD.

Para evitar este problema, normalmente o administrador do banco de dados, ou a pessoa responsável pela performance da aplicação, precisa revisar todos os códigos PL/SQL, analisando sua performance. Porém, diferentemente das instruções SQL, o Oracle não possui recursos capazes de realizar a análise de performance de execução de um bloco de comandos PL/SQL. Para isso, é necessário percorrer manualmente o código PL/SQL, buscando todas as instruções SQL contidas nele, e verificando o nível de performance destes SQL através do *Explain Plan*. Assim, serão identificados todos SQLs com problemas de performance, que poderão ser posteriormente otimizados.

Este processo manual, de verificação de performance de um código PL/SQL, em um primeiro momento parece ser simples, mas pode se tornar uma tarefa muito custosa, visto que um PL/SQL pode conter inúmeras instruções SQL em seu código.

O objetivo deste trabalho é desenvolver um protótipo de um *software* que auxilie nesta tarefa, fazendo a análise do custo de execução de um bloco de comandos PL/SQL, com base

no custo de execução de cada instrução SQL contida dentro dele.

Este trabalho é dividido em 4 capítulos principais, além de Introdução e Conclusão. No capítulo 2 é explicado como funciona o mecanismo de processamento de consultas no Oracle, bem como seus planos de execução e o funcionamento do *Explain Plan*. O capítulo 3 fala sobre a modelagem do protótipo, como funciona o analisador de PL/SQL, a gramática reconhecida pelo protótipo, bem como as funcionalidades e um exemplo de utilização do protótipo. O capítulo 4 fala sobre os detalhes da implementação, tais como a arquitetura de camadas utilizada e a classes que compõem o projeto. O capítulo 5 mostra os testes que foram realizados para verificar o funcionamento do protótipo. Finalizando, o capítulo 6 apresenta as considerações finais do trabalho realizado e algumas propostas para trabalhos futuros.

2 CONSULTAS NO ORACLE

O Oracle é um o sistema gerenciador de banco de dados (SGBD) objeto relacional, criado em meados da década de 70 por Larry Ellison, e é atualmente o SGBD mais utilizado em todo o mundo [2].

A Oracle foi o primeira companhia a comercializar um produto que utilizasse SQL (*Structured Query Language*). A SQL , ou Linguagem de Consultas Estruturada, é atualmente a linguagem padrão de consultas para banco de dados relacional devido a sua simplicidade e facilidade de uso [1].

Embora nos refiramos à linguagem SQL com uma linguagem de consulta, ela possui muitos outros recursos além da consulta ao banco de dados, como meios para a definição da estrutura dos dados a serem armazenados (ou esquemas de dados), para modificação de dados e para a especificação de restrições de segurança e integridade [3].

A linguagem SQL possui duas partes principais, a DDL e a DML. A DDL, ou Linguagem de Definição de Dados, é composta por todas as operações referentes à manipulação dos esquemas de dados e especificações de segurança. Já a DML, ou Linguagem de Manipulação de Dados, engloba as operações referentes z manipulação dos dados [3].

A tabela 1 mostra exemplos de operações DDL que permitem criar, alterar e excluir a tabela “pessoa”, bem como conceder ou revogar privilégios de consulta para um usuário “epaim”.

Tabela 1: Exemplos de operações DDL

Operação DDL
CREATE TABLE pessoa (codigo number(3), nome varchar2(80);
ALTER TABLE pessoa ADD (telefone number(10));
DROP TABLE pessoa;
GRANT select ON pessoa TO epaim;
REVOKE select ON pessoa FROM epaim;

A tabela 2 mostra exemplos de operações DML, para realizar inserção, seleção, atualização e remoção de dados sobre a tabela “pessoa”.

Tabela 2: Exemplos de operações DML

Operações DML
INSERT INTO pessoa (codigo, nome) VALUES (1,'Maria da Silva');
SELECT nome FROM pessoa WHERE codigo =1;
UPDATE pessoa SET nome='Enor Paim' WHERE codigo = 1;
DELETE FROM pessoa WHERE codigo > 10;

Além do SQL, o SGBD Oracle também possui uma linguagem de programação procedural, chamada PL/SQL. Esta linguagem contém recursos de programação similares à outras linguagens procedurais, tais como lógicas condicionais, laços de repetições, entre outros. Esta linguagem também reconhece instruções SQL, o que possibilita ao programador utilizar chamadas SQL em meio ao código procedural. Por isso o nome PL/SQL (*Procedural Language / Structured Query Language*) [1].

Com o PL/SQL, é possível codificar regras de negócios diretamente dentro do banco de dados, através da criação de objetos, tais como procedimentos, funções, pacotes e gatilhos [1].

Para processar uma consulta SQL o SGBD precisar realizar diversas tarefas, desde a tradução daquela instrução SQL em uma representação interna reconhecida pelo SGBD, até a busca dos dados, estejam eles em memória ou em disco [3].

Toda consulta SQL pode ser representada através de uma expressão algébrica relacional, que é utilizada como base para o processamento da consulta. Em muitos casos, uma única consulta pode ser representada por diversas expressões algébricas relacionais diferentes que produzem o mesmo resultado, ou seja, podem existir diversas maneiras de se processar uma mesma consulta. Cada uma destas maneiras é chamada de plano de execução [3].

Para escolher o melhor plano de execução entre os diversos planos existentes para uma mesma consulta, os SGBDs possuem um recurso chamado “otimizador de consultas”. O otimizador de consultas avalia cada plano de execução existente e, com base em estatísticas sobre os dados a serem processados, escolhe o plano de menor custo [3].

Os passos envolvidos no processamento de uma consulta são demonstrados na ilustração 1.

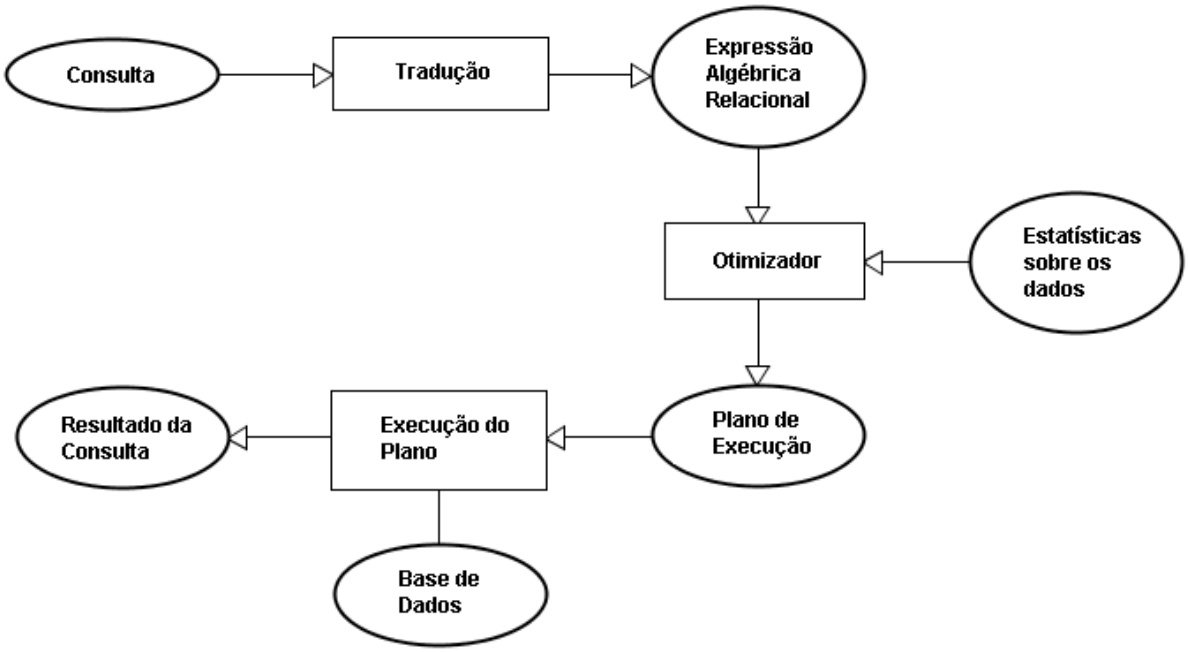

Ilustração 1: Passos do processamento de consultas

2.1 Planos de execução de consultas

Na versão 10g do SGBD Oracle, que foi a versão utilizada neste trabalho, o otimizador de consultas tem por base um mecanismo chamado CBO (C*ost Based Optimization*) [4].

O CBO é responsável por avaliar cada um dos possíveis planos de execução de uma consulta e selecionar o plano de menor custo. Este custo nada mais é do que a quantidade de recursos utilizados pelo SGBD para realizar a consulta. Estes recursos podem ser divididos em três grupos: custo de I/O, custo de CPU e custo de rede [4].

O custo de I/O é o custo da transferência dos dados do disco rígido para a memória. No Oracle, os dados são armazenados em blocos, e estes blocos são armazenados em arquivos no disco rígido. Tipicamente o custo de I/O é o custo mais significativo no processamento de uma consulta. O custo de CPU é o custo para processar os dados que já estão em memória, como por exemplo em uma operação de ordenação ou junção entre tabelas, que é realizada em memória. Para consultas que acessem dados através da rede, como por exemplo em bancos de dados distribuídos, existe um custo de rede (*network* I/O), que é o tempo gasto para trafegar as informações pela rede [4].

Para calcular estes custos, o CBO utiliza algumas estatísticas coletadas sobre os dados existentes na base. Estas estatísticas podem ser coletadas através de uma série de ferramentas

do SGBD reunidas em um recurso chamado "*Gather Schema Statistics*", onde é possível disparar processos que fazem a coleta de estatísticas referentes a grupos de tabelas, a uma única tabela, ou até mesmo a uma determinada coluna de uma tabela. Sem a geração destas estatísticas, ou até mesmo pela falta de atualização delas, o CBO poderá não calcular corretamente os custos das operações, o que acarretará uma visível perda de performance na execução das consultas [6].

Na tabela 4 pode-se ver o exemplo do plano de execução da consulta SQL demonstrada na tabela 3.

Tabela 3: Exemplo de consulta SQL

SQL
SELECT tpf_numero_tel FROM telefone_pesfis, pessoa WHERE tpf_pesfis_pessoa_cod_pessoa = pess_cod_pessoa AND pess_nome LIKE 'Enor Paim%';

Tabela 4: Exemplo de plano de execução de consulta

	Operação	Nome do Objeto	Opções	Custo
1	SELECT STATEMENT			6
2	TABLE ACCESS	TELEFONE_PESFIS	BY INDEX ROWID	3
3	NESTED LOOPS			6
4	INDEX	PESSOA_I3	RANGE SCAN	3
5	INDEX	TPF_PK	RANGE SCAN	1

Na tabela 4 pode-se ver que o plano de execução para a consulta é acessar a tabela TELEFONE_PESFIS através de 2 índices (PESSOA_I3 e TPF_PK), tendo um custo total igual à 6, como pode ser verificado na linha 1 da tabela.

2.2 Explain Plan

O Oracle possui um recurso pelo qual pode-se visualizar as informações referentes aos planos de execução das consultas. Este recurso é chamado de *Explain Plan*. Ao executar o *Explain Plan* de uma consulta, todas as informações referentes ao plano de execução

escolhido pelo CBO são armazenadas em registros de uma tabela na base de dados. Esta tabela chamada-se PLAN_TABLE [5].

Consultando os informações contidas nos registros da tabela PLAN_TABLE pode-se visualizar quais operações o SGBD precisa realizar para executar a consulta, bem como várias outras informações referentes a estas operações, tais como o tipo de acesso feito a cada tabela (completo ou indexado), se e quais índices serão utilizados, o custo de cada operação, entre outros. A tabela 5 descreve os principais atributos existentes na tabela PLAN_TABLE [9]. Foram considerados como principais atributos aqueles que trazem informações relevantes a operação e ao seu custo de execução.

Tabela 5: Descrição da tabela PLAN_TABLE

Atributo	Tipo	Descrição
OPERATION	VARCHAR2(30 BYTE)	Nome da operação realizada neste passo.
OPTIONS	VARCHAR2(255 BYTE)	Opções usadas para a operação.
OBJECT_OWNER	VARCHAR2(30 BYTE)	Dono do objeto.
OBJECT_NAME	VARCHAR2(30 BYTE)	Nome do objeto.
OBJECT_TYPE	VARCHAR2(30 BYTE)	Tipo do objeto.
OPTIMIZER	VARCHAR2(255 BYTE)	Modelo corrente de otimização.
SEARCH_COLUMNS	NUMBER	Número das colunas com predicados correspondentes.
ID	NUMBER	Número identificador da operação no plano de execução.
PARENT_ID	NUMBER	ID da operação pai.
DEPTH	NUMBER	Profundidade da operação na árvore que representa o plano.
POSITION	NUMBER	Ordem de processamento para os passos com mesmo PARENT_ID.
COST	NUMBER	Custo da operação corrente estimado pelo CBO.
CARDINALITY	NUMBER	Número de colunas retornadas pela operação corrente.
BYTES	NUMBER	Número de *bytes* retornados pela operação corrente.
CPU_COST	NUMBER	Custo de CPU da operação.
IO_COST	NUMBER	Custo de I/O da operação.
ACCESS_PREDICATES	VARCHAR2(4000 BYTE)	Predicados utilizados para localizar

		as linhas na estrutura de acesso.
FILTER_PREDICATES	VARCHAR2(4000 BYTE)	Predicados utilizados para filtrar as linhas antes delas serem produzidas.
TIME	NUMBER	Tempo de execução (em segundos) estimado pelo CBO.

Cada registro da tabela faz referência a uma operação do plano de execução da consulta. Uma operação pode ser composta por várias outras, de forma a gerar uma árvore de operações. Para gerenciar esta estrutura de árvore, a PLAN_TABLE utiliza um auto-relacionamento através dos campo ID e PARENT_ID, que indicam, respectivamente, o número identificador da operação e o número identificador da operação pai.

Para realizar o *Explain Plan* de uma consulta SQL, basta executar, na console de comandos SQL do SGBD Oracle, o comando "*explain plan*" seguido da instrução SQL, conforme podemos observar na tabela 6.

Tabela 6: Sintaxe do comando Explain Plan

Comando
SQL> explain plan for select codigo from tabela where nome like 'Enor%';

O *Explain Plan* é um recurso muito utilizado por administradores de banco de dados para verificar o nível de performance da execução de instruções SQL no banco de dados. Através dele, é possível identificar pontos críticos que influenciam na performance da consulta. Após identificados estes pontos críticos, algumas ações podem ser tomadas para tentar resolver o problema da performance, tais como a alteração do próprio código SQL da consulta, ou até mesmo a criação de índices para as tabelas em questão. Tais ações muitas vezes podem surtir grande efeito e melhorar significativamente a performance de execução das consultas.

Freqüentemente administradores de banco de dados e programadores PL/SQL necessitam analisar a performance de execução de um bloco de comandos PL/SQL. Mas diferentemente de uma consulta SQL, o SGBD Oracle não possui nenhum recurso capaz de realizar a análise de custo de execução de um bloco de comandos PL/SQL. Para se ter idéia do custo geral de execução de um PL/SQL, é necessário fazer separadamente a análise de cada SQL contido em seu código, e somar os custos destes SQLs. Este processo pode se tornar uma tarefa muito custosa, visto que um PL/SQL pode conter inúmeras instruções SQL em seu código.

3 MODELAGEM DO PROTÓTIPO

O objetivo do protótipo é facilitar a tarefa de análise de custo de execução de um bloco de comandos PL/SQL. Para isto, o protótipo fará a análise de custo de execução de cada instrução SQL contida no PL/SQL. Para identificar as instruções SQL contidas no bloco PL/SQL, o protótipo utilizará um analisador sintático.

Basicamente, o protótipo é composto pelos componentes: a interface e o analisador sintático. Através da interface o usuário poderá se conectar à base de dados e selecionar um determinado PL/SQL que será enviado ao analisador. Por sua vez, o analisador identificará os SQLs contidos no PL/SQL e buscará as informações do plano de execução destes SQLs, através do *Explain Plan*, retornando para a *interface* todos estes dados de forma clara e organizada. A conexão entre o protótipo e o Oracle sempre é feita através de um JDBC. A ilustração 2 demonstra melhor o funcionamento desta estrutura.

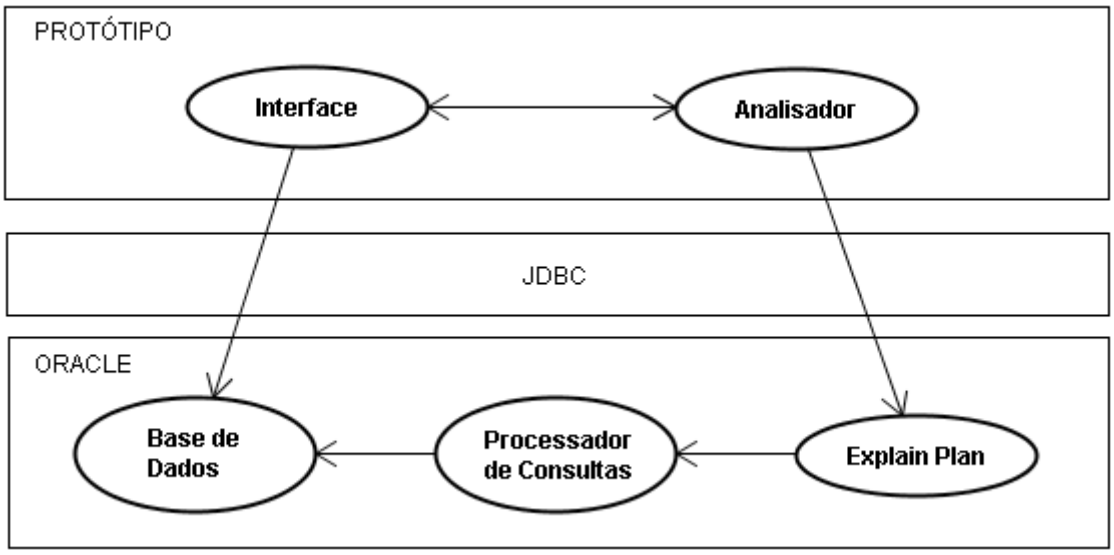

Ilustração 2: Estrutura do protótipo

3.1 O Analisador

Analisador sintático é um algoritmo, programa ou componente de *software* responsável por determinar se uma entrada de dados pode ser derivada de uma gramática

formal [7]. Gramática formal é um objeto matemático que permite especificar uma linguagem, ou seja, é um conjunto de regras capazes de gerar todas as possibilidades combinatórias desta linguagem [7].

Entretanto, o protótipo não necessitará fazer a análise sintática completa do bloco de comandos PL/SQL. Ele apenas utilizará uma gramática pré-definida para reconhecer e diferenciar, dentro do código, quais comandos são instruções SQL e quais comandos são instruções PL/SQL.

Após separadas todas as instruções SQL do código PL/SQL, o protótipo utilizará o *Explain Plan* do Oracle para descobrir o custo de execução de cada SQL.

As instruções SQL podem conter variáveis declaradas no bloco de comandos PL/SQL. Porém, ao submeter um SQL ao *Explain Plan* que contenha variáveis do bloco PL/SQL, o *Explain Plan* não saberá que estas variáveis fazem parte do PL/SQL, e tentará identificá-las, sem sucesso, como algum objeto existente na base de dados. Para que o *Explain Plan* reconheça estas variáveis corretamente e faça o seu devido tratamento, é necessário identificar cada variável com o caractere ':' exatamente a sua frente. Na tabela 7 pode-se verificar a identificação de duas variáveis (data1 e data2) na instrução SQL submetida ao *Explain Plan*.

Tabela 7: Explain Plan com variáveis

Comando
SQL> explain plan for select codigo from tabela where data between :data1 and :data2

3.2 Tabela de Símbolos

Para poder identificar nas instruções SQL as variáveis do bloco PL/SQL, o analisador deverá conhecer os nomes destas variáveis. Para isso, o analisador utilizará uma tabela de símbolos (TS). A TS é uma estrutura utilizada em compiladores para guardar informações sobre os nomes declarados em um programa, bem como informações referentes à estes nomes [8].

Em geral a TS começa a ser construída durante a análise léxica, quando os identificadores são reconhecidos. Na primeira vez em que um identificador é encontrado, o analisador armazena-o na tabela. Toda vez que um identificador é reconhecido no programa

fonte, a TS é consultada, a fim de verificar se o nome já está registrado. Caso não esteja, é feita a sua inserção na tabela [8].

Existem vários modos de organizar e acessar uma TS. Os mais comuns são através de listas lineares e tabelas *hash*. Lista linear é o mecanismo mais simples, mas seu desempenho é pobre quando o número de consultas é elevado. Tabelas *hash* têm melhor desempenho, mas exigem mais memória e esforço de programação [8].

Como neste protótipo não estaremos implementando todas as funções de um compilador, o que implica em uma redução significativa no número de acessos à TS, utilizaremos uma lista linear para armazenar a TS.

O analisador utilizará a TS toda a vez em que for analisar um SQL. Antes de submeter o SQL ao *Explain Plan*, o analisador deverá percorrer todo o código SQL para identificar se nele existe algum nome pertencente à TS. Cada nome encontrado na TS deve ser identificado com o caractere ':' a sua frente, para que assim o *Explain Plan* possa identificá-lo como uma variável.

3.3 A Gramática

O PL/SQL possui uma extensa gramática, constituída por diversas produções, que abrangem o reconhecimento de todos os comandos da linguagem.

No Oracle existem diversos tipos de objetos PL/SQL, tais como funções, procedimentos, gatilhos, pacotes, entre outros. Como opção para este trabalho foi definido que o protótipo fará apenas a análise de funções e procedimentos, pois estes são os tipos de objetos mais utilizados por desenvolvedores PL/SQL e através deles pode-se demonstrar todas as funcionalidades do protótipo.

Considerando que o protótipo reconhecerá apenas funções e procedimentos, e sabendo que é necessário apenas identificar as variáveis nos blocos de declaração e as instruções SQL, não será necessário utilizar toda a gramática da linguagem PL/SQL. Por este motivo, foi definida uma gramática que contém apenas as regras de produção necessárias para estas finalidades.

Para definir esta gramática foi utilizada a notação BNF (*Backus-Naur Form*). A notação BNF é um mecanismo formal utilizado para descrever linguagens de programação. Na notação BNF uma gramática é definida por conjuntos de regras, onde cada regra tem um lado esquerdo, ou LHS (*Left Hand Side*), e um lado direito, ou RHS (*Right Hand Side*),

conforme pode-se observar na ilustração 3 [7].

Ilustração 3: Exemplo de regra em notação BNF

Os símbolos que constituem as regras BNF podem ser terminais ou não terminais. Os símbolos não terminais são delimitados por < e >, e os símbolos terminais por aspas simples. Na BNF também existem dois operadores, conforme a tabela 8 [7].

Tabela 8: Operadores na BNF

Operador	Função
::=	Indica o fim do LHS e início do RHS.
|	Permite especificar símbolos alternativos no RHS. Similar ao “ou” lógico.

Na Figura 3 pode-se observar a aplicação dos símbolos não terminais <bloco> e <comando>, do símbolo terminal ';' e dos operadores ::= e |.

Como o objetivo deste trabalho é criar um protótipo para demonstrar como pode ser feita a análise de custo de um bloco de comandos PL/SQL contendo procedimentos e funções, não serão contempladas pelo analisador todas as produções existentes na gramática oficial da linguagem PL/SQL. O analisador reconhecerá apenas as produções existentes na gramática definida para este trabalho, conforme a tabela 9. Caso o PL/SQL analisado contenha qualquer outra regra de produção que não esteja especificada na gramática, o protótipo não conseguirá realizar a análise do código com sucesso.

Tabela 9: Gramática reconhecida pelo protótipo

Notação BNF
<plsql> ::= <function> | <procedure>
<function> ::= 'function' <id> <parametros> 'return' <tipos> <is_as> <blocodeclare> <bloco>
<procedure> ::= 'procedure' <id> <parametros> <is_as> <blocodeclare> <bloco>

```
<id> ::= token
<parametros> ::= '(' <listaparametros> ')' | []
<listaparametros> ::= <parametro> ',' <listaparametros> | <parametro>
<parametro> ::= <id> <qualquercoisa>
<tipo> ::= token '(' token ')' | token
<is_as> ::= 'is' | 'as'
<blocodecl> ::= <type> <blocodecl> |
                <cursor> <blocodecl> |
                <function> <blocodecl> |
                <procedure> <blocodecl> |
                <id> <qualquercoisa> ';' <blocodecl> | []
<type> ::= 'type' <qualquercoisa> ';'
<cursor> ::= 'cursor' <id> <parametros> 'is' <sql> ';'
<bloco> ::= 'declare' <blocodeclare> 'begin' <comandos> 'end' ';' |
            'declare' <blocodeclare> 'begin' <comandos> 'end' <id> ';' |
            'begin' <comandos> 'end' ';' |
            'begin' <comandos> 'end' <id> ';'
<comandos> ::= <bloco> <comandos> | <comando> <comandos> | <comando>
<comando> ::= <forcursor> | <qualquercoisa_comando> ';'
<forcursor> ::= 'for' <id> 'in' <id> '(' <qualquercoisa> ')'  'loop' |
                'for' <id> 'in' <id> 'loop'
<qualquercoisa> ::= token <qualquercoisa> | token
<qualquercoisa_comando> ::= <sql> | token <qualquercoisa_comando> | token
<sql> ::= 'insert' <qualquercoisa> ';' |
          'delete' <qualquercoisa> ';' |
          'update' <qualquercoisa> ';' |
          'select' <qualquercoisa> ';'
<comentario> ::= '/*' <qualquercoisa> '*/' |
                 '--' <qualquercoisa> '\n'
```

3.4 Funcionalidades do protótipo

O protótipo possui quatro funcionalidades principais. Através delas o usuário pode conectar-se à base de dados, selecionar um PL/SQL, executar a análise do PL/SQL e visualizar os planos de execução dos SQLs, conforme demonstra a ilustração 4.

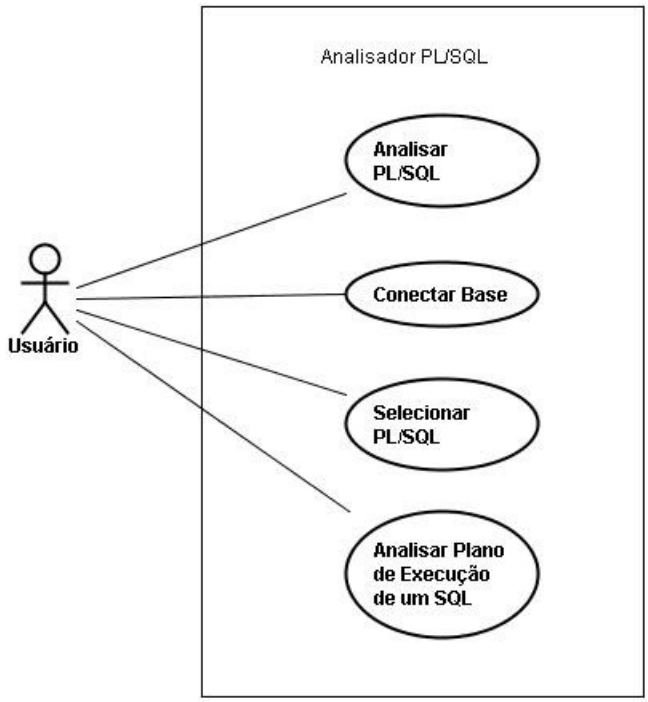

Ilustração 4: Funcionalidades do protótipo

Estas funcionalidades possuem uma certa ordem de execução. Primeiramente o usuário deverá se conectar ao banco de dados e selecionar uma função ou procedimento, para que o protótipo execute a análise deste objeto. Após a execução da análise do objeto, o usuário poderá visualizar separadamente o plano de execução de cada SQL contido no objeto. Esta seqüência de execução pode ser observada na ilustração 5.

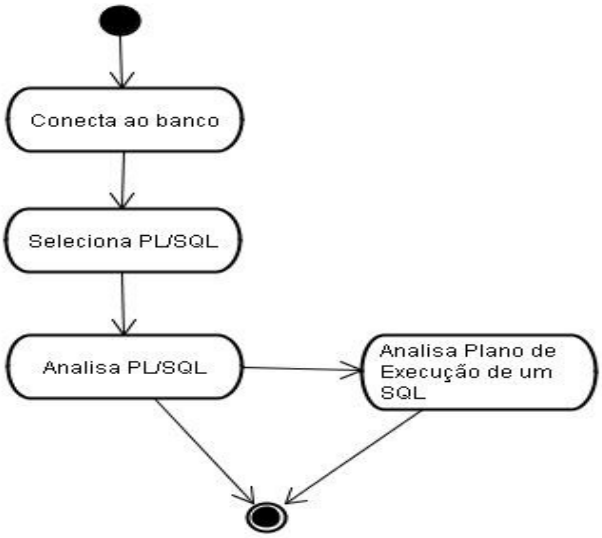

Ilustração 5: Seqüência de execução das funcionalidades do protótipo

3.5 Exemplo de utilização do protótipo

Nesta sessão será demonstrado, através de um exemplo, um passo-a-passo de como utilizar cada uma das funcionalidades do protótipo.

Para estabelecer uma conexão com o banco de dados, o usuário deverá abrir a tela "*connect*" através do menu "*File → Connect*" ou a da tecla de atalho "Ctrl+C". Nesta tela, o usuário deverá fornecer todas as informações referentes a conexão com o banco de dados, conforme demonstra a ilustração 6.

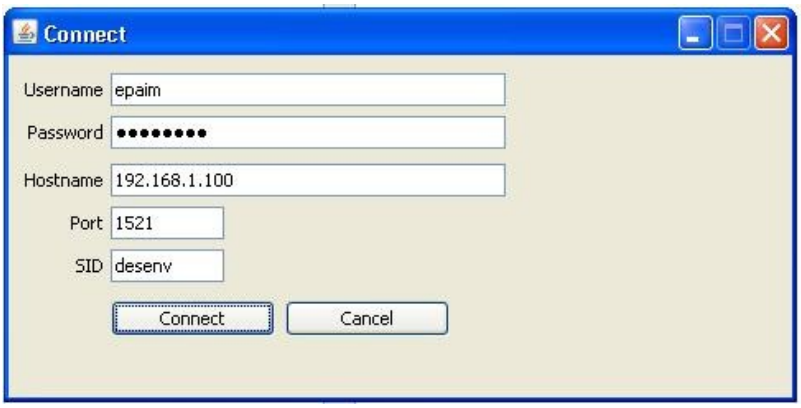

Ilustração 6: Tela de Conexão

Após estabelecida a conexão com o banco de dados, o protótipo trará, na tela principal, uma lista de todos os objetos (funções e procedimentos) PL/SQL existentes na base de dados, conforme a ilustração 7.

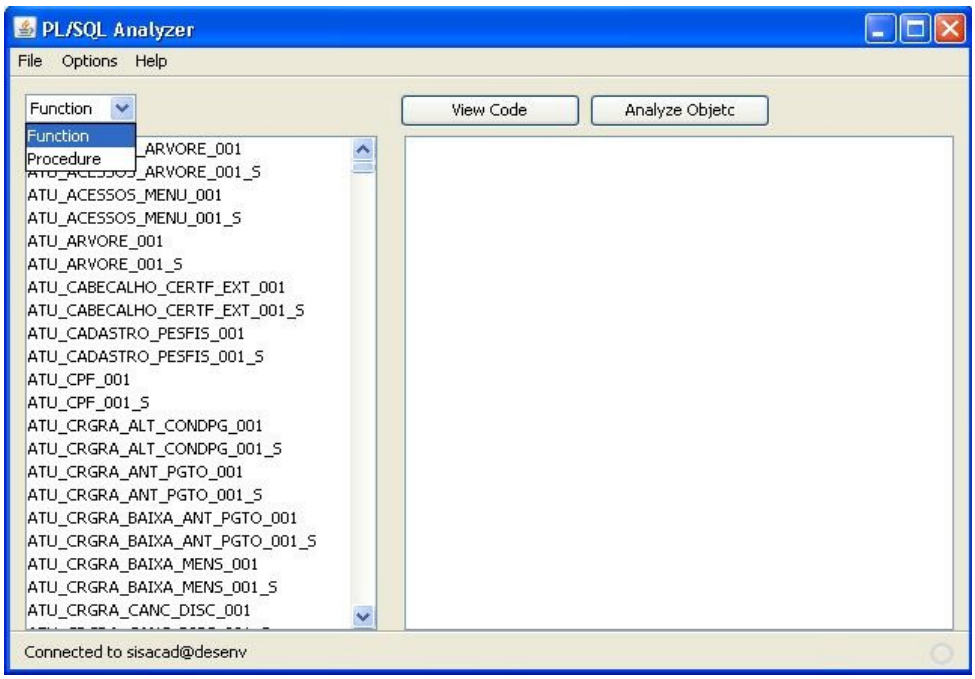

Ilustração 7: Tela principal - Objetos PL/SQL

Para selecionar um objeto e visualizar seu código fonte, basta dar dois cliques no nome do objeto na lista à esquerda, ou selecioná-lo na lista e clicar no botão '*View Code*', conforme demonstra a ilustração 8.

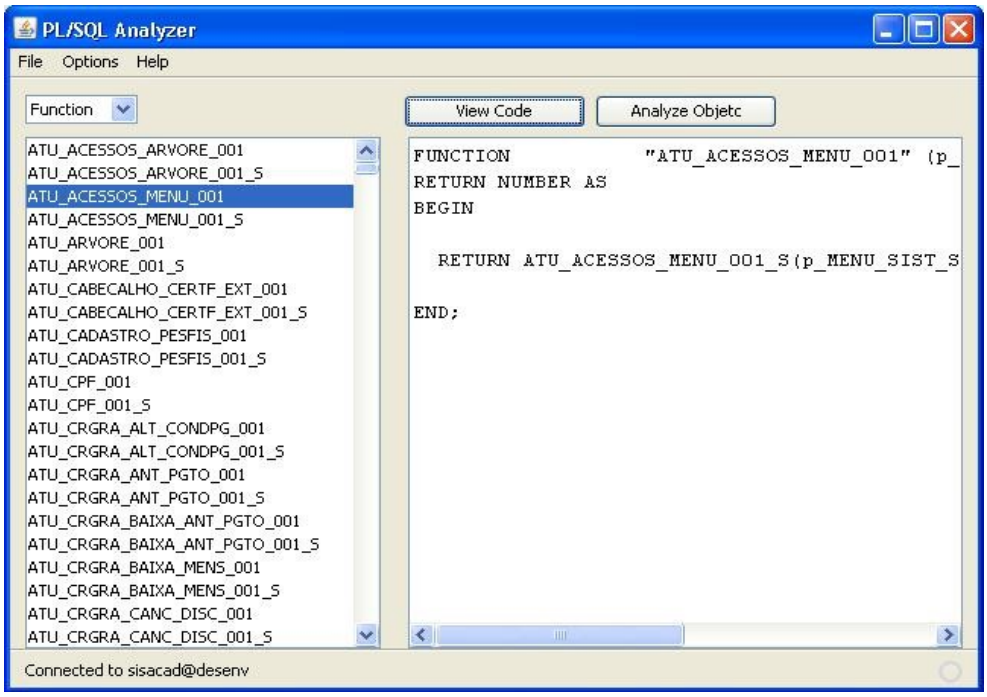

Ilustração 8: Tela principal - Selecionar objeto / Visualizar código fonte

Após selecionar um objeto, o usuário poderá executar a análise do seu código clicando no botão ”*Analyze Object*”. Ao clicar neste botão, uma nova tela será aberta contendo todos os SQLs presentes no código fonte do objeto, bem como as informações de custo de execução e a linha em que o SQL se encontra dentro do código. Dois campos totalizadores também são mostrados, um contendo o número total de SQLs existentes no código e outro contendo o custo total de execução do código, que nada mais é do que a soma dos custos de todos os SQLs. A tela “*Analyze Object*” pode ser observada na ilustração 9.

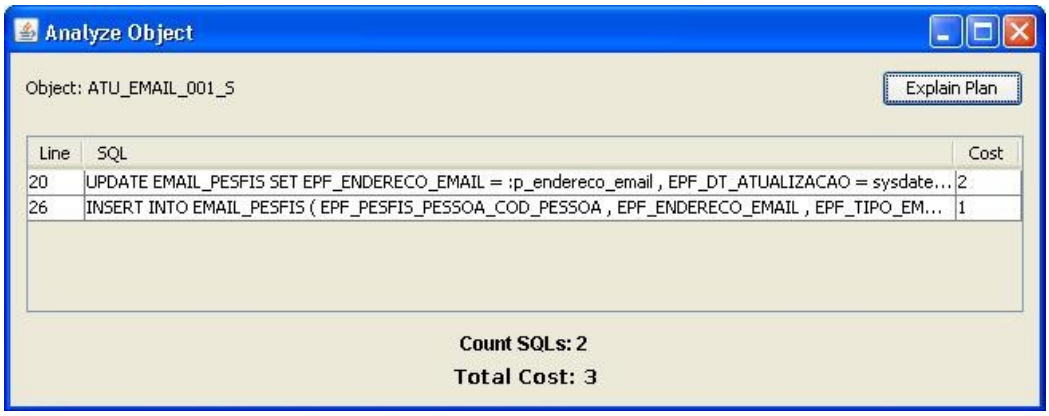

Ilustração 9: Tela de análise do objeto

Na tela “*Analyze Object*” é possível visualizar o plano de execução de cada um dos

SQLs. Para isto, basta dar dois cliques sobre o SQL desejado, ou então selecioná-lo na lista de SQLs e clicar no botão "*Explain Plan*". Ao executar esta ação, surge uma nova tela contendo todas as informações do plano de execução do SQL selecionado, conforme a ilustração 10.

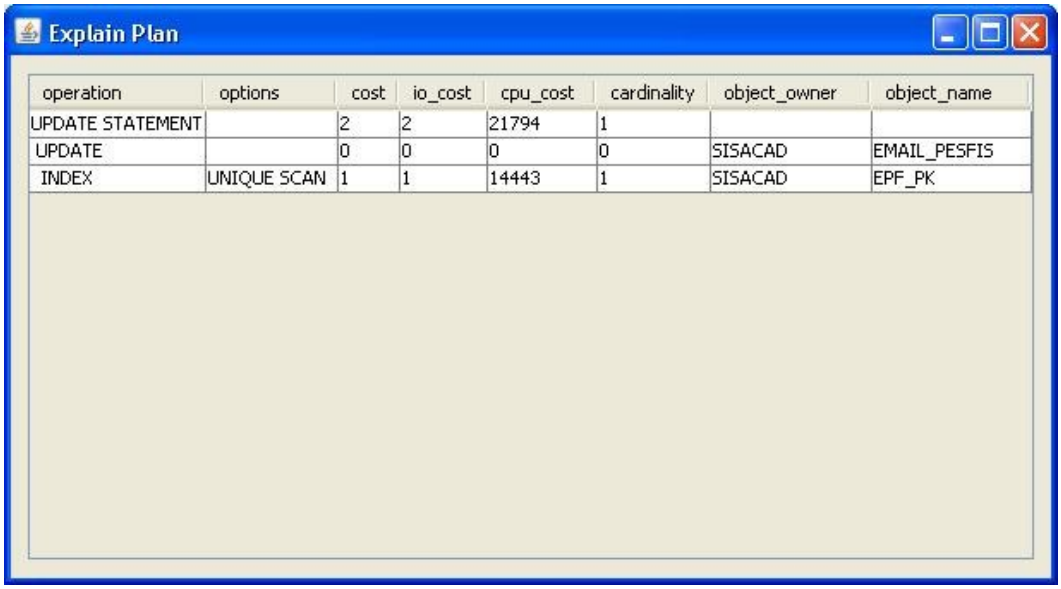

Ilustração 10: Tela do plano de execução do SQL

Os campos visualizados na tela "*Explain Plan*" podem ser customizados através do menu "*Options → Plan Fields*" na tela principal do protótipo. Esta opção abre uma tela onde o usuário pode selecionar quais informações estarão visíveis posteriormente na tela "*Explain Plan*", conforme a ilustração 11. As opções disponíveis fazem referência aos principais campos da tabela PLAN_TABLE.

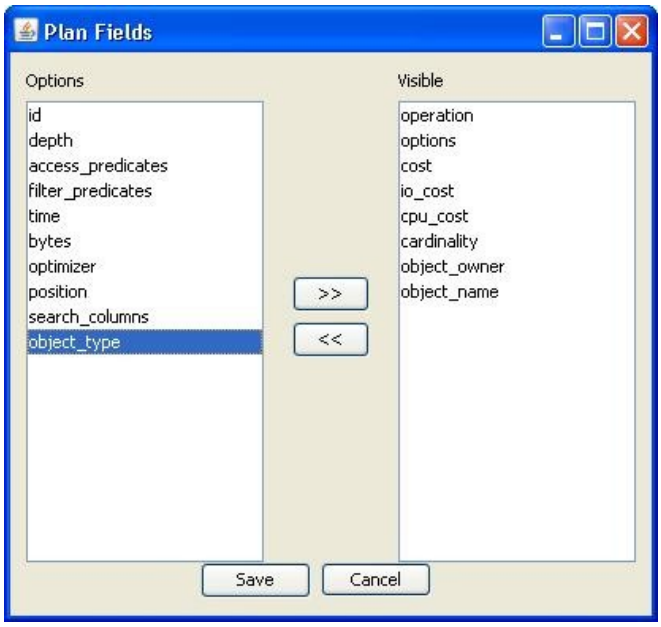

Ilustração 11: Tela de configuração dos campos do Explain Plan

O protótipo guarda sempre a última configuração feita pela usuário, gravando estas informações em um arquivo de configuração do sistema.

Assim como as configurações dos campos do *Explain Plan*, o protótipo também grava no arquivo de configuração os dados da última conexão estabelecida.

O usuário também pode se desconectar do banco de dados a qualquer momento, através do menu "*File* → *Disconnect*", conforme a ilustração 12.

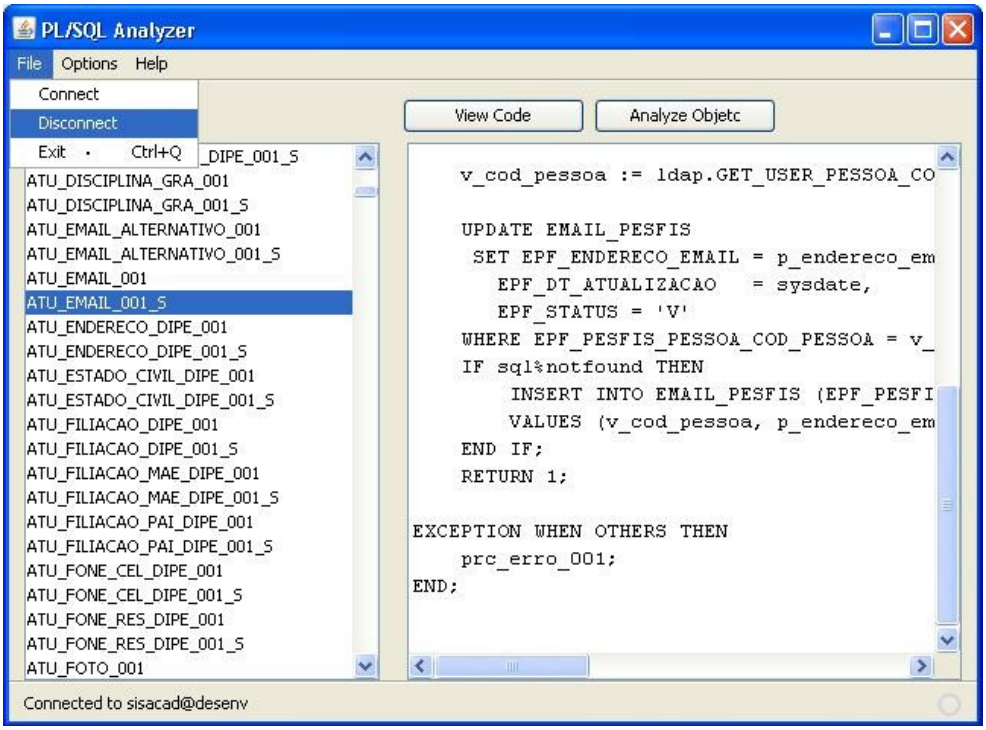

Ilustração 12: Menu Disconnect

4 IMPLEMENTAÇÃO

O protótipo foi desenvolvido com a linguagem de programação orientada a objetos Java, especificamente com o JDK versão 1.5. Para conectar-se ao banco de dados Oracle foi utilizado um JDBC proprietário da Oracle, o pacote OJDBC14.jar, que pode ser baixado gratuitamente no site da Oracle. A versão do SGBD Oracle utilizada foi a 10.2.0.4 *Standard Edition*. Como ferramenta de desenvolvimento java e para a criação das telas foi utilizado o software Netbeans IDE versão 6.0.1.

4.1 Camadas e Classes

No desenvolvimento do sistema, foi utilizada uma arquitetura de três camadas [10], conforme descrito na tabela 10.

Tabela 10: Camadas do sistema

Camada	Descrição
1 - Conexão com o banco de dados	Contém as classes utilizadas para estabelecer a conexão com o banco de dados Oracle e realizar as consultas necessárias à base de dados.
2 - Regras de negócio	Contém todas as classes que implementam a lógica principal do sistema. É nesta camada que encontramos a classes Analisador, que é a principal classe do sistema.
3 - Interface	Contém as classes utilizadas para implementar as telas do sistema.

As camadas do sistema e suas respectivas classes estão ilustradas na ilustração 13.

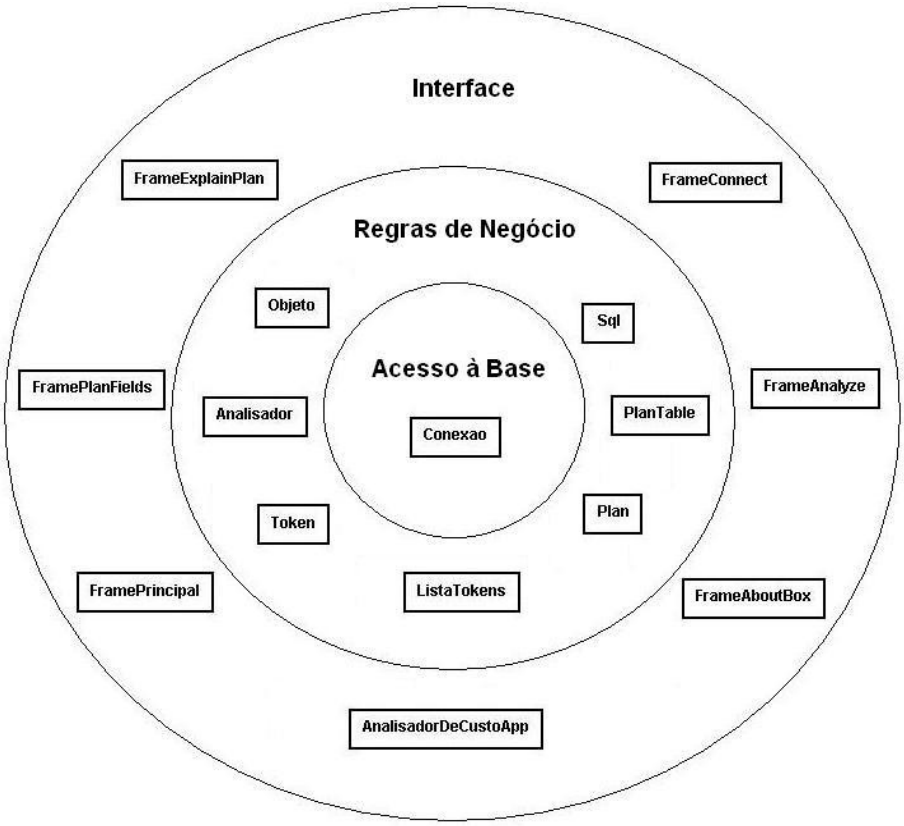

Ilustração 13: Camadas e classes do sistema

O protótipo possui um total de quinze classes java, sendo uma da camada de conexão, sete da camada de regras de negócio, e sete da camada de interface. A tabela 11 mostra um descritivo do papel de cada classe do sistema.

Tabela 11: Descritivo das classes do protótipo

Classe	Descrição
Conexao.java	Estabelece a conexão com o banco de dados e realizar as consultas necessárias à base de dados, tais como buscar os nomes dos objetos PL/SQL, buscar o código fonte dos objetos, executar o *Explain Plan* e buscar as informações da PLAN_TABLE. Esta classe utiliza a biblioteca ojdbc14.jar, que é o JDBC desenvolvido pela Oracle para o Java.
Token.java	Representa um *token* do código fonte de um PL/SQL.
ListaTokens.java	Representa uma lista contendo todos os *Tokens* do código fonte de um PL/SQL.
Sql.java	Mantém as informações referentes a um determinado SQL de um objeto PL/SQL. Responsável por executar o *Explain Plan* do SQL em questão, e buscar as informações da PLAN_TABLE.
Objeto.java	Representa um objeto PL/SQL do banco de dados.
PlanTable.java	Representa as informações contidas na tabela PLAN_TABLE.
Plan.java	Representa um registro da tabela PLAN_TABLE.
Analisador.java	Responsável por fazer a análise dos objetos PL/SQL. Contém todas as regras da gramática utilizada. Faz a busca por instruções SQL contidas no código fonte do objeto, bem como a geração da TS e a identificação, nos SQLs, dos nomes registrados na TS.
FramePrincipal.java	Gera a tela principal do protótipo.
FrameConnect.java	Gera a tela com dados para conexão com o banco de dados.
FrameAboutBox.java	Gera a tela “sobre” do sistema.
FramePlanFields.java	Gera a tela de configuração dos campos visíveis na tela do *Explain Plan*.
FrameAnalyze.java	Gera tela de análise do objeto PL/SQL.
FrameExplainPlan.java	Gera a tela com as informações do *Explain Plan* do SQL.
AnalisadorDeCustoApp.java	Classe responsável por iniciar o protótipo. Contém o método *main* do sistema.

Para entender melhor o funcionamento das classes do sistema como um todo, pode-se observar a relação entre as classes, bem como seus atributos e métodos no diagrama de classes representado na ilustração 14.

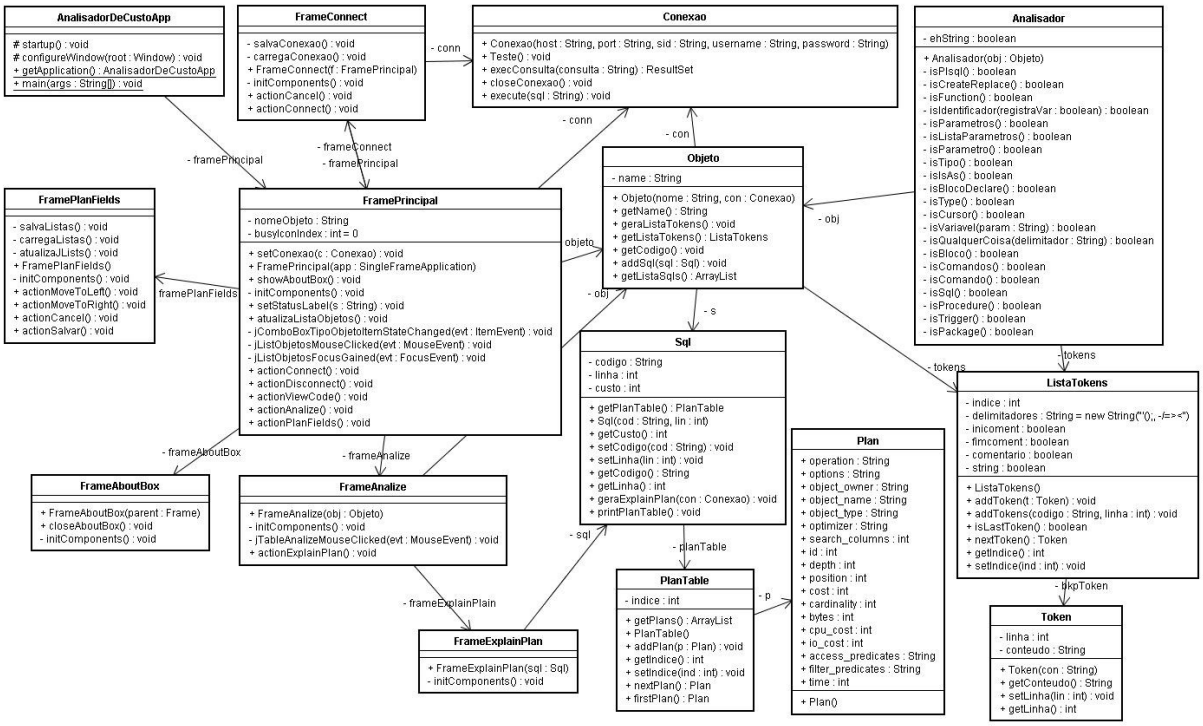

Ilustração 14: Diagrama de classes do sistema

A lógica principal do sistema encontra-se na classe Analisador.java. Ela é responsável por analisar o código fonte do objeto PL/SQL, criar a TS, identificar e separar as instruções SQL do código, e identificar dentro do código SQL todos os nomes existentes na TS. Analisaremos alguns trechos de código para ver como isto foi feito.

Tabela 12: Função java que reconhece a regra de produção <id> da gramática

Código fonte

```
private boolean isIdentificador(boolean registraVar)
{
      Token t = this.tokens.nextToken();
      if (registraVar)
         this.addToTs(t.getConteudo());
      System.out.println("isIdentificador true");
      return true;
}
```

A tabela 12 mostra o código fonte da função responsável por reconhecer a regra de produção <id> da gramática, e também por registrar os nomes na TS. O registro do nome na TS pode ser opcional, pois são registrados apenas os nomes encontrados em blocos de declarações.

Tabela 13: Função java que reconhece a regra de produção <sql> da gramática

Código fonte

```
private boolean isSql() {
        String sql;
        int indice = this.tokens.getIndice();
        Token t = this.tokens.nextToken();
        boolean string = false;
        if (!this.ehString) {
            if (t.getConteudo().toLowerCase().equals("insert") ||
t.getConteudo().toLowerCase().equals("delete") ||
t.getConteudo().toLowerCase().equals("update") ||
t.getConteudo().toLowerCase().equals("select"))
            {
                sql = t.getConteudo();
                int linha = t.getLinha();
                t = this.tokens.nextToken();
                while (!t.getConteudo().equals(";")) {
                    if (t == null) {
                        System.out.println("isSql true");
                        return false;
                    }
                    if (this.isVariavel(t.getConteudo()) && !string){
                        sql = sql + " :"+tiraPontoCursor(t.getConteudo());
                    }
                    else {
                        if (t.getConteudo().equals(" ") || string)
                            sql = sql + t.getConteudo();
                        else
                            sql = sql + " " + t.getConteudo();
                    }
                    if (t.getConteudo().equals("'")) {
                        string = !string;
                    }
                    t = this.tokens.nextToken();
                }
                Sql s = new Sql(sql,linha);
                this.obj.addSql(s);
                System.out.println("isSql true");
                return true;
            }
        }
        this.tokens.setIndice(indice);
        System.out.println("isSql false");
        return false;
}
```

O tabela 13 mostra o código fonte da função que reconhece a regra de produção <sql>

da gramática. Após identificar o trecho de código SQL, a função instancia um novo objeto do tipo Sql e o associa ao Objeto PL/SQL que esta sendo analisado. Além disso, a função verifica se o *token* é uma variável, ou seja, verifica se o nome esta registrado na TS, e caso esteja, adiciona o caractere ':' à frente do nome.

Mais um exemplo de uma função que reconhece a regra de produção <*function*> da gramática, pode ser observado na tabela 14.

Tabela 14: Função java que reconhece a regra de produção <function> da gramática

Código Fonte

```
private boolean isFunction() {
        int indice = this.tokens.getIndice();
        Token t = this.tokens.nextToken();
        if (t.getConteudo().toLowerCase().equals("function")) {
            if (this.isIdentificador(false))
                if (this.isParametros()) {
                    t = this.tokens.nextToken();
                    if (t.getConteudo().toLowerCase().equals("return"))
                        if (this.isTipo())
                            if (this.isIsAs())
                                if (this.isBlocoDeclare())
                                    if (this.isBloco())
                                            return true;
                }
        }
        this.tokens.setIndice(indice);
        return false;
}
```

Assim como as funções demonstradas nas tabelas 12, 13 e 14, para cada regra de produção da gramática existe uma função na classe Analisador.

No PL/SQL existem duas maneiras de se adicionar comentários ao código. Uma delas é utilizando as *tags* '/*' e '*/' para sinalizar, respectivamente, o início e o fim do bloco de comentários. A outra forma é utilizar o *tag* '--' que indica que daquele ponto em diante, até o final da linha, tudo será considerado como comentário. Na tabela 15 podemos observar exemplos das duas formas de utilização de comentários no PL/SQL.

Tabela 15: Exemplo de utilização de comentários em PL/SQL

Código fonte

begin
  aux := 1; -- Comentário
end;

begin
  aux := 1; /* Inicio do comentário
  fim do comentári */
end;

O analisador desconsidera todo o código existente entre comentários, pois o método que adiciona *tokens* à lista de *tokens* do objeto PL/SQL já realiza um tratamento especial que retira, da lista de *tokens*, todo o código que estiver comentado.

No desenvolvimento da camada de interface do protótipo, ou seja, nas classes que montam as telas do sistema, foram utilizados objetos do pacote javax.swing do java, e objetos do pacote org.jdesktop.application que foram incorporados devido a utilização de alguns recursos específicos da ferramenta de desenvolvimento Netbeans, tais como a aparência das telas similares ao sistema de janelas do *Windows*.

4.2 Execução do aplicativo do protótipo

Como dito anteriormente, o protótipo foi desenvolvido com a linguagem de programação Java, e para executá-lo é necessário ter instalado o *runtime* do Java, mais conhecido como JRE. O protótipo é compatível com a versão 1.5 ou posterior do JRE, visto que o seu desenvolvimento foi realizado com o JDK versão 1.5.

O projeto do protótipo possui um arquivo chamado AnalisadorDeCusto.jar, que é pacote java que contém todas as classes do projeto. O projeto também possui dois arquivos de configuração chamados connect.cfg e planFields.cfg que são, respectivamente, os arquivos que guardam as informações da última conexão e a configuração dos campos da tela *Explain Plan*. Além disso, o projeto possui um diretório chamado lib que contém todos os pacotes java (arquivos .jar) das bibliotecas utilizadas, tais como o JDBC da Oracle (OJDBC14.jar).

Para executar o aplicativo do protótipo, é necessário chamar o pacote do projeto através do *runtime* do java, conforme mostra a tabela 16.

Tabela 16: Comando para executar o aplicativo do protótipo

Comando
C:\> java -jar AnalisadorDeCusto.jar

5 TESTES DO PROTÓTIPO

Para validar o protótipo, foram executados diversos testes, afim de realizar o reconhecimento de todas as regras de produção definidas na gramática, verificar se o protótipo estava reconhecendo todas as instruções SQL existentes dentro do código PL/SQL e se os valores dos custos de cada SQL gerados pelo protótipo conferiam com os resultados obtidos utilizando-se o *Explain Plan* diretamente na console de comandos SQL do Oracle.

5.1 Testes de validação da gramática

Para testar o reconhecimento da gramática pelo protótipo, foram criados dois objetos PL/SQL, descritos nos anexos A e B, que englobam todas as regras de produção definidas na gramática.

Como a gramática definida engloba o reconhecimento de apenas dois tipos de objetos PL/SQL, funções e procedimentos, foram criados para os testes um PL/SQL que representa uma função e outro que representa um procedimento. A tabela 17 define os testes realizados.

Tabela 17: Testes realizados para validação da gramática

Teste	Descrição
1	Análise do PL/SQL de tipo função (anexo A)
2	Análise do PL/SQL de tipo procedimento (anexo B)

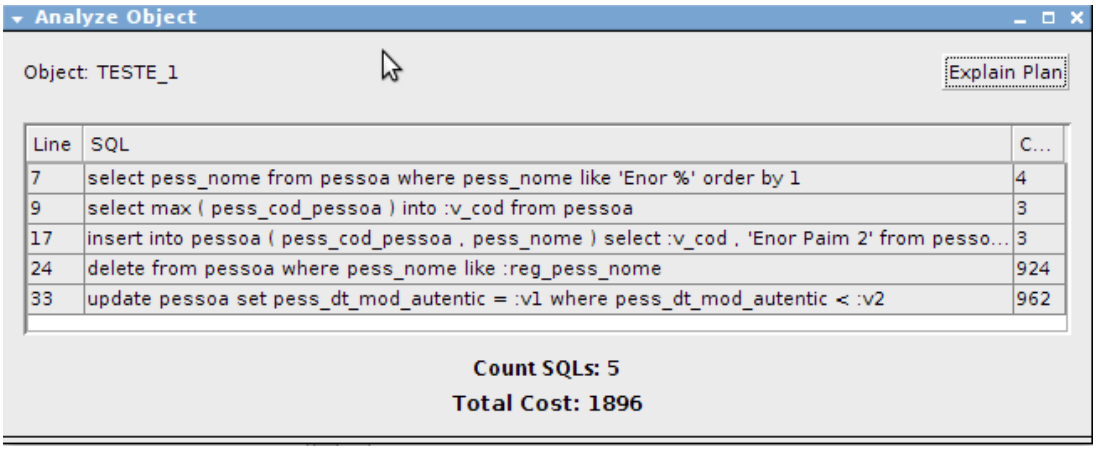

Ilustração 15: Teste 1

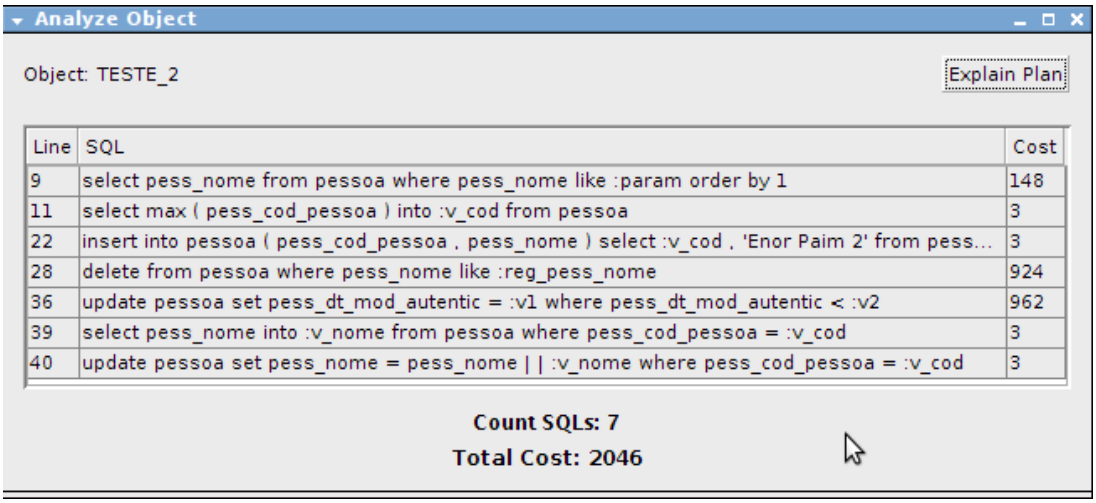

Ilustração 16: Teste 2

Pode-se verificar nas ilustrações 15 e 16 que o protótipo reconheceu perfeitamente o código PL/SQL contido nos dois objetos PL/SQL. Não houveram erros de execução e os SQLs contidos no objeto foram mostrados na tela "*Analyze Object*". Pode-se assim considerar que os testes 1 e 2 foram bem sucedidos.

5.2 Testes de reconhecimento dos SQLs

Os mesmos testes utilizados para validar a gramática foram utilizado para verificar se o protótipo reconheceria todos os SQLs contidos no objeto.

Analisando o código do PL/SQL do teste 1, pode-se verificar a existência de 5 instruções SQL, conforme a tabela 18. É importante salientar que o SQL da linha 12 do Anexo

A não deve ser considerado pelo protótipo, pois ele encontra-se dentro de um comentário.

Tabela 18: SQLs existentes no PL/SQL do teste 1

Linha	SQL
7	select pess_nome from pessoa where pess_nome like 'Enor %' order by 1
9	select max (pess_cod_pessoa) into :v_cod from pessoa
17	insert into pessoa (pess_cod_pessoa , pess_nome) select :v_cod , 'Enor Paim 2' from pessoa where pess_nome = 'Enor Paim'
24	delete from pessoa where pess_nome like :reg_pess_nome
33	update pessoa set pess_dt_mod_autentic = :v1 where pess_dt_mod_autentic < :v2

Comparando a tabela 18 com o resultado obtido no protótipo, demonstrado na ilustração 17, podemos verificar que o protótipo reconheceu todos os SQLs, sendo assim o teste 1 foi bem sucedido.

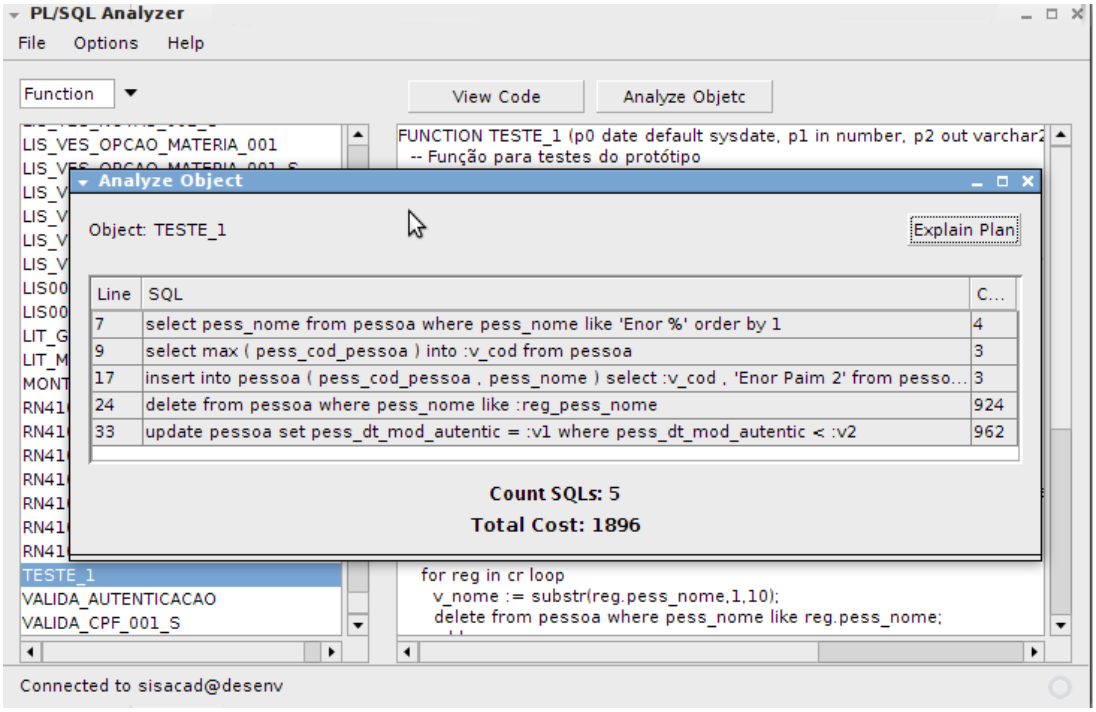

Ilustração 17: SQLs encontrados pelo protótipo no teste 1

O PL/SQL do teste 2 possui 7 instruções SQL, conforme mostra tabela 19. No teste 2 o analisador deve desconsiderar a instrução SQL existente na linha 12, visto que aquele trecho de código faz parte de um *string*.

Tabela 19: SQLs existentes no PL/SQL do teste 2

Linha	SQL
9	select pess_nome from pessoa where pess_nome like :param order by 1
11	select max (pess_cod_pessoa) into :v_cod from pessoa
22	insert into pessoa (pess_cod_pessoa , pess_nome) select :v_cod , 'Enor Paim 2' from pessoa where pess_nome = 'Enor Paim'
28	delete from pessoa where pess_nome like :reg_pess_nome
36	update pessoa set pess_dt_mod_autentic = :v1 where pess_dt_mod_autentic < :v2
39	select pess_nome into :v_nome from pessoa where pess_cod_pessoa = :v_cod
40	update pessoa set pess_nome = pess_nome || :v_nome where pess_cod_pessoa = :v_cod

Comparando a tabela 19 com o resultado obtido no protótipo, demonstrado na ilustração 18, pode-se verificar que o protótipo reconheceu todos os SQLs, sendo assim o teste 2 também foi bem sucedido.

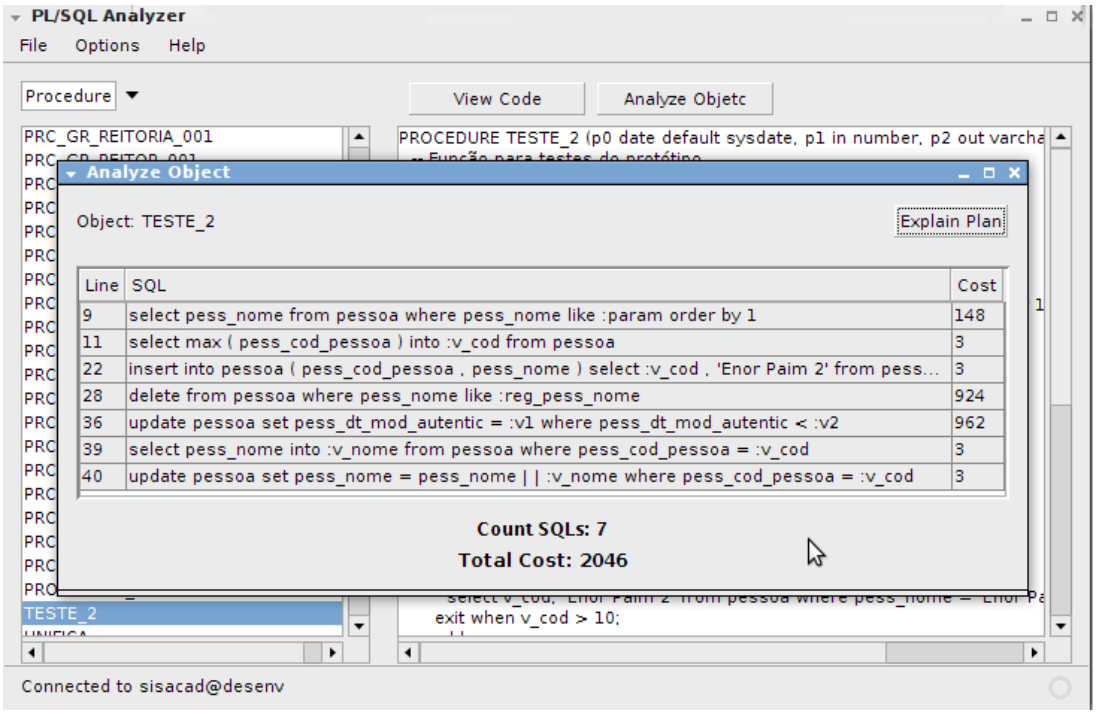

Ilustração 18: SQLs encontrados pelo protótipo no teste 2

5.3 Conferência dos valores dos custos

Para validar se os valores dos custos gerados pelo protótipo na análise dos objetos PL/SQL estão corretos, utilizamos os mesmos testes 1 e 2.

Os valores gerados pelo protótipo foram comparados com os valores gerados pelo *Explain Plan* diretamente na console de comandos SQL do Oracle.

As análises feitas pelo protótipo estão demonstradas nas ilustrações 17 e 18. Os valores gerados pelo *Explain Plan* na console de comandos SQL do Oracle podem ser verificadas nas ilustrações 19 e 20.

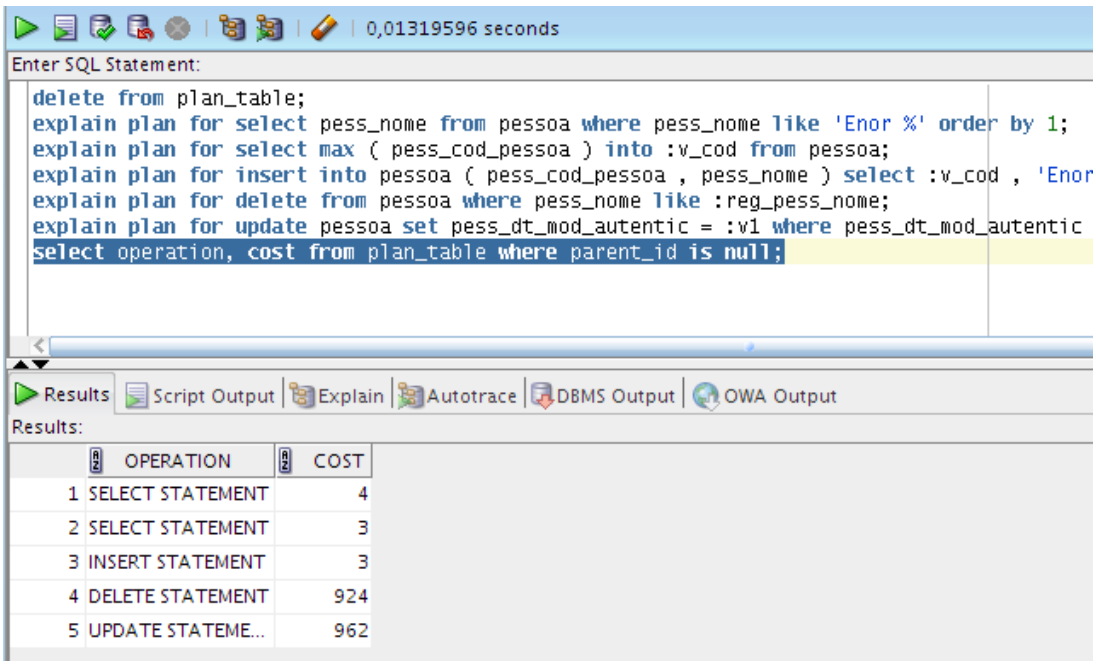

Ilustração 19: Valores gerados pelo Explain Plan para o teste 1

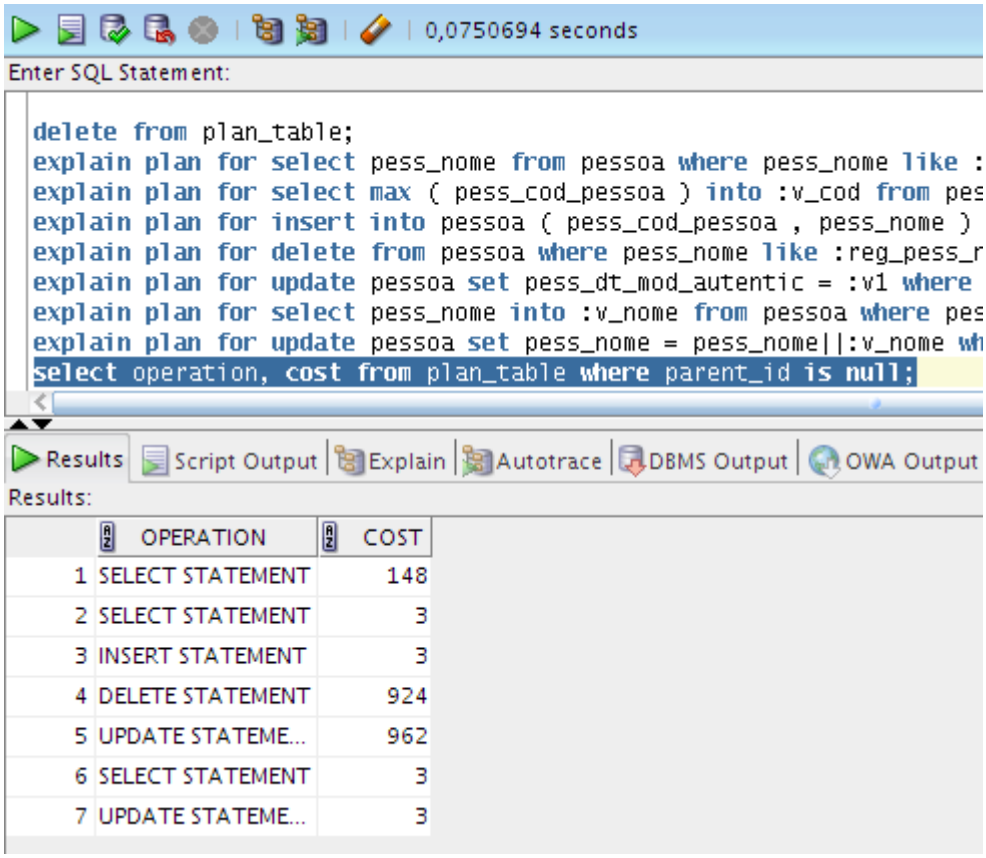

Ilustração 20: Valores gerados pelo Explain Plan para o teste 2

Comparando os valores gerados pelo protótipo com os valores gerados pelo *Explain Plan* pode-se verificar que o protótipo buscou corretamente os valores dos custos de cada SQL dos objetos PL/SQL analisados, comprovando assim o correto funcionamento do protótipo.

5.4 Conferência dos dados dos planos de execução

Para conferir se os valores gerados pelo protótipo para os planos de execução dos SQLs estão corretos, podemos comparar a saída gerada pelo protótipo diretamente com os dados da PLAN_TABLE após executar o *Explain Plan*. Para realizar esta comparação foi utilizado como exemplo os três SQLs presentes no PL/SQL do anexo C, descritos na tabela 20.

Tabela 20: SQLs utilizados para os testes dos planos de execução

	SQL
1	SELECT epf_endereco_email FROM pessoa , pessoa_fisica , email_pesfis WHERE epf_tipo_email = 'I' AND epf_pesfis_pessoa_cod_pessoa = pesf_pessoa_cod_pessoa AND pesf_pessoa_cod_pessoa = pess_cod_pessoa AND pess_nome LIKE 'Enor%'
2	SELECT hisg_conceito FROM pessoa , historico_graduacao WHERE hisg_dadgra_alugra_pesfis_pess = pess_cod_pessoa AND pess_nome LIKE 'Enor%'
3	SELECT logs_data_hora FROM log_senha , pessoa WHERE logs_usuario = pess_cod_pessoa AND pess_nome LIKE 'Enor%'

As ilustrações 21, 22 e 23 mostram os planos de execução gerados pelo o protótipo para os SQLs da tabela 20.

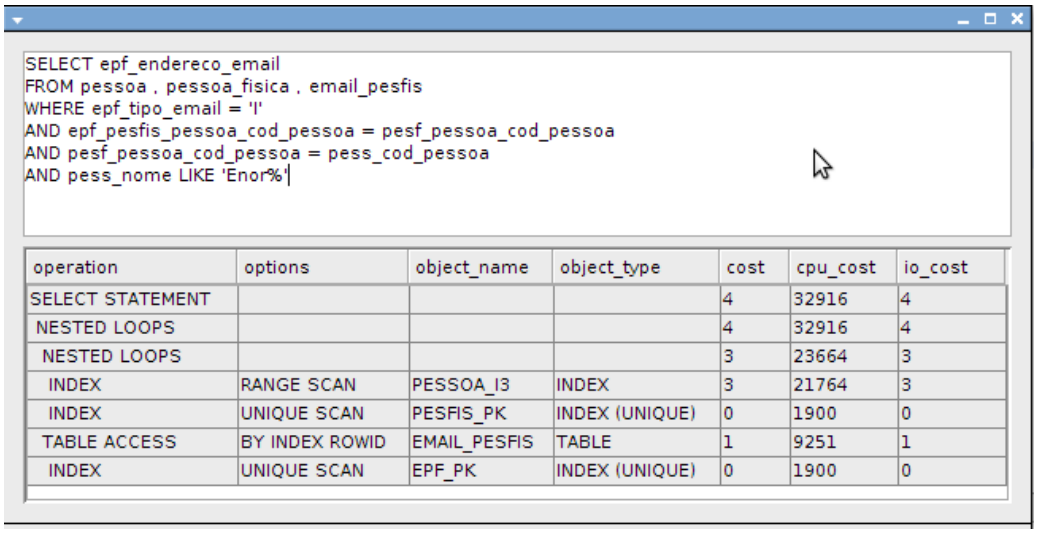

Ilustração 21: Plano de execução gerado pelo protótipo para o SQL 1

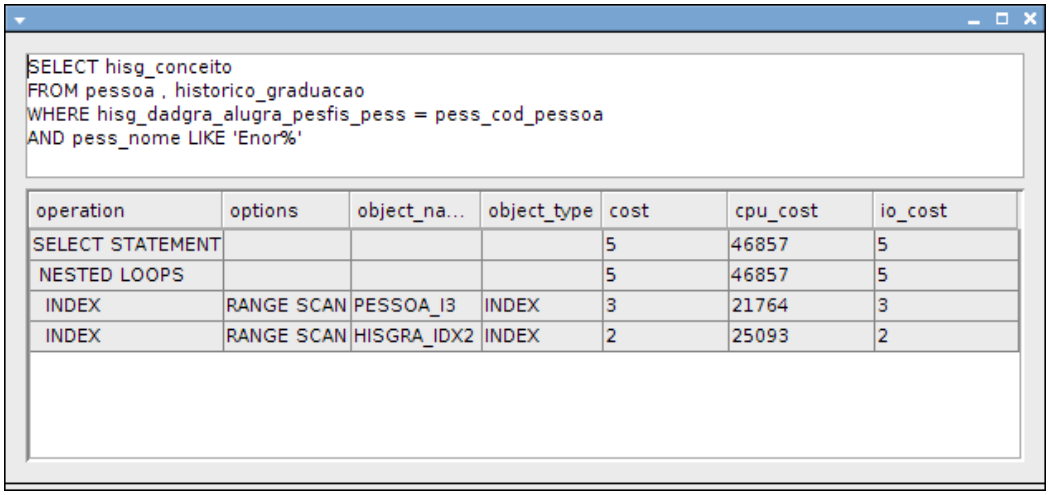

Ilustração 22: Plano de execução gerado pelo protótipo para o SQL 2

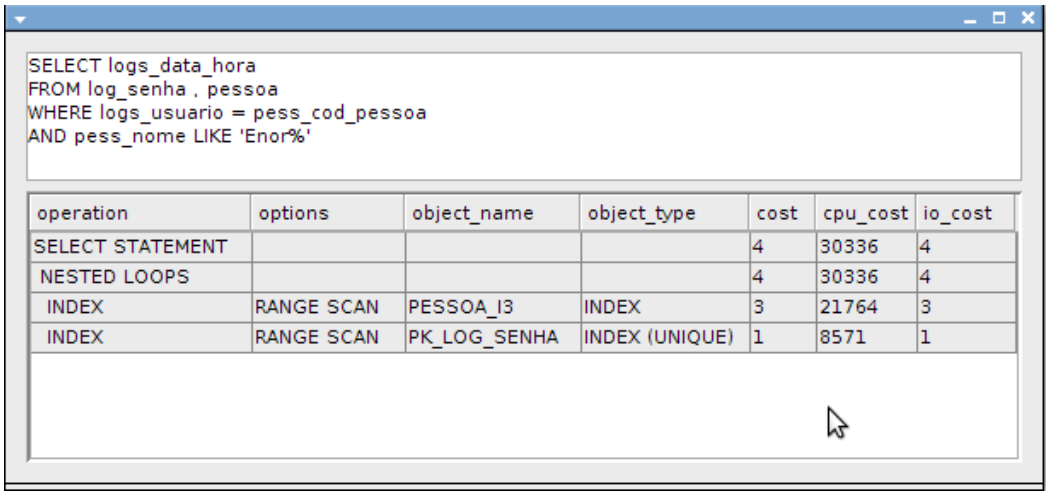

Ilustração 23: Plano de execução gerado pelo protótipo para o SQL 3

As ilustrações 24, 25 e 26 mostram os valores da tabela PLAN_TABLE gerados pelo *Explain Plan* para os SQLs da tabela 10.

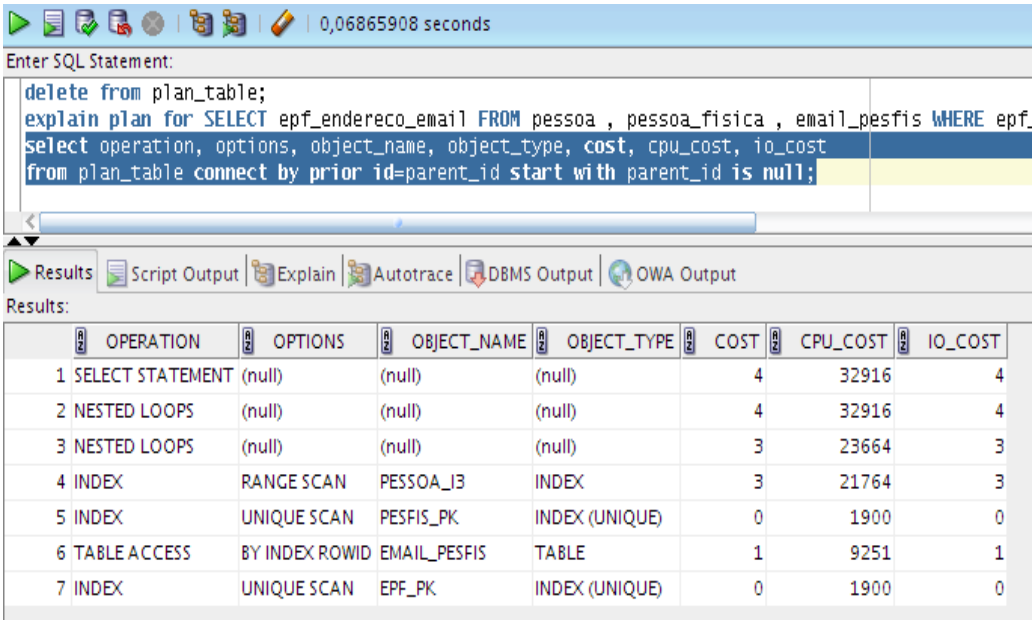

Ilustração 24: Valores gerados pelo Explain Plan para o SQL 1

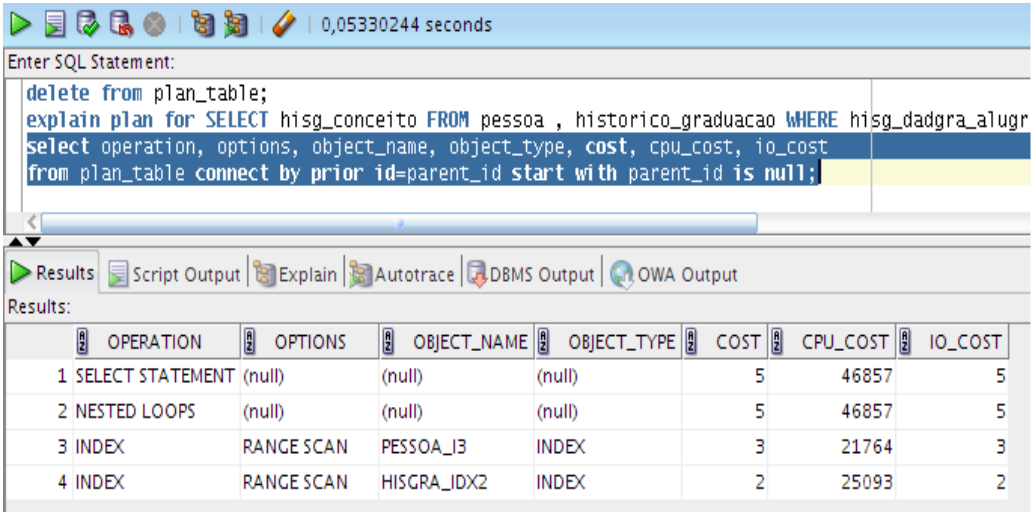

Ilustração 25: Valores gerados pelo Explain Plan para o SQL 2

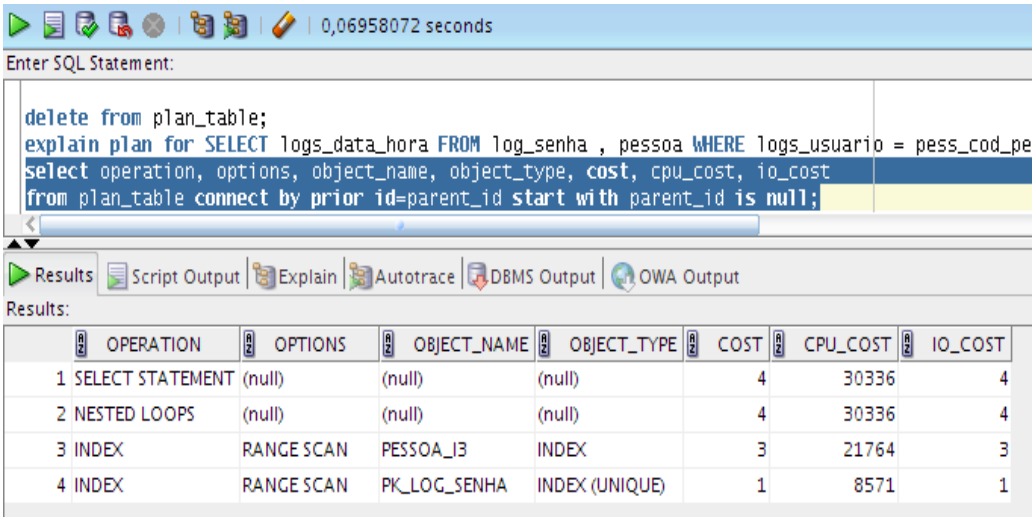

Ilustração 26: Valores gerados pelo Explain Plan para o SQL 3

Comparando os resultados obtidos no protótipo com os resultados gerados pelo *Explain Plan* diretamente na console de comandos SQL do Oracle, podemos observar que não houve divergências, o que valida os testes e comprova o funcionamento do protótipo.

6 CONCLUSÃO

Este trabalho apresentou a implementação de um protótipo de *software* capaz de realizar a análise de custo de execução de um PL/SQL, baseado nos custos dos planos de execução das instruções SQL contidas nele. Para isto foi criado um analisador sintático que identifica, dentro do código PL/SQL, todas as instruções SQL e as submete ao *Explain Plan* para buscar o valor do custo de execução do plano de consulta escolhido pelo CBO. Com base nesta lógica foram criadas interfaces para que o usuário pudesse escolher os objetos PL/SQL a serem analisados e visualizar detalhes sobre a análise destes objetos de forma customizada, bem como os planos de execução dos SQLs contidos neles.

Através dos testes realizados podemos comprovar que o trabalho alcançou seu objetivo, pois o protótipo conseguiu gerar com sucesso, em todos os testes, as informações esperadas para o PL/SQL em questão, simplificando assim a tarefa de análise do custo do PL/SQL.

Como proposta para trabalhos futuros temos a possibilidade de incrementar a gramática reconhecida pelo protótipo, a fim de que ele possa reconhecer todas as regras de produção da gramática oficial do PL/SQL, tais como regras relacionadas a gatilhos e pacotes. Poderiam ser criadas também, opções para análise em lote de PL/SQLs, ou seja, que fosse possível analisar mais do que um PL/SQL de uma só vez. Além disso, poderiam ser criadas algumas funcionalidades extras que auxiliassem o usuário a identificar pontos críticos na análise do PL/SQL. Como exemplo, poderia ser criado um cadastro onde o usuário pudesse definir valores para os campos do *Explain Plan*, que quando detectados pelo sistema, disparassem um gatilho que sinalizasse o usuário de alguma forma, como por exemplo com um mensagem de aviso na tela do protótipo. Assim, cada usuário poderia customizar o *software* de forma que, por exemplo, o sistema pudesse identificar sozinho pontos críticos de performance nos PL/SQLs analisados. Todas estas funcionalidades e melhorias facilitariam a utilização do protótipo, aumentariam sua eficácia e trariam um conjunto maior de opções de análise para o usuário.

7 REFERÊNCIAS

[1] LONEY, K.; KOCH, G. **Oracle 9i** : The Complete Reference. Berkley, California: McGraw-Hill, 2002.

[2] Oracle.com. **A História do Oracle** : Inovação, Liderança e Resultados. Disponível em <http://www.oracle.com/global/br/corporate/story.html>. Acessado em: 20 de outubro 2008.

[3] SILBERSCHATZ, A.; KORTH, F. H.; SUDARSHAN, S. **Sistemas de Banco de Dados** 3. ed. São Paulo: Pearson Makron Books, 1999.

[4] Metalink.oracle.com. **Cost Based Optimizer** (CBO) Overview. Disponível em <https://metalink2.oracle.com/metalink/plsql/ml2_documents.showDocument?p_database_id=NOT&p_id=10626.1>. Acessado em: 30 de setembro 2008.

[5] NIEMIEC, J. R. **Oracle Database 10g Performance** Tuning Tips & Techniques. New York: McGraw-Hill, 2007.

[6] GREENWALD, R.; STARCKWIAK, R.; STERN, J. **Oracle Essentials** – Oracle Database 11g. Beijing: O'Reilly Media, 2008.

[7] AHO, V. A.; SETHI, R.; ULLMAN, D. J. **Compiladores** – Princípios, Técnicas e Ferramentas. Rio de Janeiro, RJ: Guanabara Koogan S.A., 1995.

[8] PRICE, A. M. A.; TOSCANI, S. S. **Implementação de Linguagens de Programação** : Compiladores 2. ed. Porto Alegre: Sagra Luzzatto, 2001.

[9] Oracle.com. **Oracle Database Online Documentation 10g Relesea 2 (10.2)** : Part Number B14237-03. Disponível em <http://download.oracle.com/docs/cd/B19306_01/server.102/b14237/statviews_4222.htm#REFRN29510>. Acessado em: 25 de outubro 2008.

[10] PRESSMAN, S. R. **Engenharia de Software** 6. ed. São Paulo: McGraw-Hill, 2006.

ANEXO A

```
1   create or replace FUNCTION TESTE_1 (p0 date default sysdate, p1 in number, p2 out
    varchar2, p3 in out date) RETURN VARCHAR2 IS
2   -- Função para testes do protótipo
3    v_cod number;
4    v_nome varchar2(80);
5    type t_teste is table of number;
6    cursor cr is
7      select pess_nome from pessoa where pess_nome like 'Enor %' order by 1;
8  BEGIN
9    select max(pess_cod_pessoa) into v_cod from pessoa;
10   /*
11   select min(pess_cod_pessoa) into v_cod from pessoa;
12   Este sql não deve ser reconhecido pelo protótipo pois estas comentado!
13   */
14   if v_cod > 1 then
15     loop
16       v_cod := v_cod + 1;
17       insert into pessoa (pess_cod_pessoa, pess_nome)
18         select v_cod, 'Enor Paim 2' from pessoa where pess_nome = 'Enor Paim';
19       exit when v_cod > 10;
20     end loop;
21
22     for reg in cr loop
23       v_nome := substr(reg.pess_nome,1,10);
24       delete from pessoa where pess_nome like reg.pess_nome;
25     end loop;
26
27     declare
28       v1 date;
29       v2 date;
30     begin
31       v1 := sysdate;
32       v2 := v1 + 1;
33       update pessoa set pess_dt_mod_autentic = v1 where pess_dt_mod_autentic < v2;
34     end;
35   end if;
36   RETURN 'OK';
37 exception when others then
38   RETURN 'erro';
39 END TESTE_1;
```

ANEXO B

```
1  create or replace PROCEDURE TESTE_2 (p0 date default sysdate, p1 in number, p2 out
   varchar2, p3 in out date) AS
2  -- Função para testes do protótipo
3    v_cod number;
4    v_nome varchar2(80);
5    type t_teste is table of varchar2(80);
6    vet t_teste;
7    idx number(9);
8    cursor cr (param in varchar2) is select pess_nome from pessoa where pess_nome like
     param order by 1;
9  BEGIN
10   select max(pess_cod_pessoa) into v_cod from pessoa;
11   -- select min(pess_cod_pessoa) into v_cod from pessoa;
12   v_nome := 'select sysdate from fual';
13   idx := 0;
14   for reg in cr ('Enor%') loop
15     vet(idx) := reg.pess_nome;
16     idx := idx + 1;
17   end loop;
18   if v_cod > 1 then
19     loop
20       v_cod := v_cod + 1;
21       insert into pessoa (pess_cod_pessoa, pess_nome)
22         select v_cod, 'Enor Paim 2' from pessoa where pess_nome = 'Enor Paim';
23       exit when v_cod > 10;
24     end loop;
25     for reg in cr('Enor Paim') loop
26       v_nome := substr(reg.pess_nome,1,10);
27       delete from pessoa where pess_nome like reg.pess_nome;
28     end loop;
29     declare
30       v1 date;
31       v2 date;
32     begin
33       v1 := sysdate;
34       v2 := v1 + 1;
35       update pessoa set pess_dt_mod_autentic = v1 where pess_dt_mod_autentic < v2;
36     end;
37   end if;
38   select pess_nome into v_nome from pessoa where pess_cod_pessoa = v_cod;
39   update pessoa set pess_nome = pess_nome||v_nome where pess_cod_pessoa = v_cod;
40 exception when others then
41   dbms_output.put_line('erro');
42 END TESTE_2;
```

ANEXO C

```
1  CREATE OR REPLACE PROCEDURE TESTE_3 AS
2   aux varchar2(80);
3
4   cursor cr1 is
5    SELECT epf_endereco_email
6    FROM pessoa, pessoa_fisica, email_pesfis
7    WHERE epf_tipo_email = 'I'
8    AND epf_pesfis_pessoa_cod_pessoa = pesf_pessoa_cod_pessoa
9    AND pesf_pessoa_cod_pessoa = pess_cod_pessoa
10   AND pess_nome LIKE 'Enor%';
11
12  cursor cr2 is
13   SELECT hisg_conceito
14   FROM pessoa, historico_graduacao
15   WHERE hisg_dadgra_alugra_pesfis_pess = pess_cod_pessoa
16   AND pess_nome LIKE 'Enor%';
17
18  cursor cr3 is
19   SELECT logs_data_hora
20   FROM log_senha, pessoa
21   WHERE logs_usuario = pess_cod_pessoa
22   AND pess_nome LIKE 'Enor%';
23
24 BEGIN
25
26  for reg in cr1 loop
27   dbms_output.put_line(reg.epf_endereco_email);
28  end loop;
29
30  for reg in cr2 loop
31   dbms_output.put_line(reg.hisg_conceito);
32  end loop;
33
34  for reg in cr3 loop
35   dbms_output.put_line(reg.logs_data_hora);
36  end loop;
37
38 END TESTE_3;
```