UNIVERSIDADE FEDERAL DO RIO DE JANEIRO

ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELETRÔNICA E DE COMPUTAÇÃO

SQL Translator – Biblioteca e framework multiplataforma para desenvolvimento de software com isolamento de camadas para banco de dados

Autores: _____
Cícero Ricardo Máximo Bezerra

_____
Paulo Felipe Braga Gandos

Orientador:_____
Sérgio Barbosa Villas-Boas

Examinador:_____
Aloysio de Castro Pedrosa

Examinador:_____
Edilberto Strauss

DEL
08/2005

à Deus e as nossas famílias

Agradecimentos

Como somos dois temos duas vezes mais pessoas a agradecer:

Cícero

A Deus, por ter sempre estado comigo ao longo de todos esses anos.

Aos meus pais, Expedito e Lenira, que me acompanharam durante toda a minha caminhada até aqui, sempre com muito amor e dedicação.

Ao meu irmão,Luis Felipe, que tantas vezes teve que aturar um irmão ocupado e irritado com a faculdade mas mesmo assim sempre continuou do meu lado.

Aos saudosos amigos, Padre Bruno e Eduardo Ribeiro, pela grande amizade e exemplos de luta que forneceram.

À minha namorada, Marcela Coutinho, por ter mudado o significado desse dois anos em que estamos juntos e ter suportado comigo as dificuldades das recentes etapas da minha vida.

Ao amigo Paulo, que me soube dar a força necessária para a realização deste projeto além de sempre dispor de uma grande paciência.

Aos meus amigos, Tatiana Ramos, André Quadros, Arthur Martinelli, Guilherme Lello, Igor Prata, por terem dado uma ajuda inestimável para a realização deste projeto.

Aos amigos, que ajudaram a tornar estes cinco anos de engenharia uma aventura mais simples.

Ao professor e amigo Osvaldo Pereira, que me deu o animo necessário para concluir a faculdade não permitindo que eu desistisse.

Ao professor Sergio Villas-Boas, pela dedicação e a amizade que teve conosco durante o projeto e em especial nesta fase final.

A banca de projeto final, Aloysio Pedrosa, Edilberto Strauss, por aceitarem o convite para comporem a banca.

Ao povo brasileiro, por ter contribuído com a realização deste curso superior através dos impostos.

Paulo

A minha mãe a ao meu pai, Leila Gandos e Silvino Gandos, pelos bons exemplos, dedicação e amor.

Aos meus avós, pelo grande carinho que sempre demonstram.

Ao meu irmão, Fabrício Gandos, pelo companheirismo e amizade.

Aos meus amigos, Filippe Hallack, Thiago Pelizon, Miguel Balieiro, Patrícia Manso, pelos grandes momentos que passamos juntos.

Aos meus amigos, que sempre confiaram em mim e me apoiaram.

Ao professor Sergio Villas-Boas, grande mestre e amigo.

Aos professores, pela dedicação na admirável tarefa de ensinar.

Ao povo brasileiro, por pagar, via impostos, o curso superior.

Resumo

Um dos grandes obstáculos das aplicações comerciais modernas é a utilização dos diversos tipos de SQL pelos programadores. Por motivos comerciais diversos os fabricantes de SGBD (Sistemas Gerenciais de Banco de Dados) procuram incluir no SQL diretivas que visam otimizar o funcionamento do código em seus programas de Banco. Tal iniciativa no entanto causa a proliferação de diversos sotaques de SQL. A existência destes sotaques dificulta a criação de programas para SGBDs no qual não se tenha costume, onera o desenvolvimento de sistemas e requer da equipe de desenvolvimento constante atenção entre as sutilezas que diferenciam um sotaque de outro.

O objetivo deste trabalho é a construção de uma arquitetura que facilite o desenvolvimento de aplicações em C++ isolando o programa do SGBD. O isolamento consiste em arbitrar uma sintaxe de SQL como padrão. A conexão com um SGBD com SQL não padrão é feita através da passagem da *string* de SQL para o tradutor da SQL Translator, que traduz para o SQL adequado ao SGBD. Escolheu-se o PostgreSQL como SQL padrão. Para ilustrar o funcionamento do SQL Translator foi implementado o tradutos para a linguagem Oracle.

Palavra-Chave

SQL

C++

Tradutores

Oracle

PostgreSQL

Isolamento

Índice

1 Introdução

1.1 Apresentação

Desde o advento da informática, diversas linhas de conhecimento têm se desenvolvido, cada qual agregando mais funcionalidade às atividades que podem ser realizadas pelos computadores. Dentre essas linhas podemos citar a de Banco de Dados que trabalha com informações e formas de armazená-las e recuperá-las. Dentro desta linha podemos destacar diversos sistemas comerciais que fornecem as funcionalidades de armazenamento e recuperação de informações, bem como ferramentas para se manipular propriamente o dado. Para permitir a execução de operações nestes bancos utiliza-se a Structured Query Language (SQL).

Um dos grandes obstáculos das aplicações comerciais modernas é a utilização dos diversos tipos de SQL pelos programadores. Por motivos comerciais diversos os fabricantes de SGBD (Sistemas Gerenciais de Banco de Dados) procuram incluir no SQL diretivas que visam otimizar o funcionamento do código em seus programas de Banco. Tal iniciativa, no entanto, causa a proliferação de diversos sotaques[1] de SQL. A existência destes sotaques dificulta a criação de programas para SGBDs no qual não se tenha costume, onera o desenvolvimento de sistemas e requer do desenvolvedor constante atenção entre as sutilezas que diferenciam um sotaque de outro.

Para contribuir com o esforço de desenvolvimento de sistemas, existem iniciativas que visam diminuir tais dificuldades. O objetivo de tais iniciativas é criar de alguma maneira uma camada de isolamento entre o programa e o acesso ao banco, ou fornecer programas externos que realizem a conversão de SQLs.

1.2 Motivação

O mercado de soluções com foco no auxílio do desenvolvimento de aplicações chama cada vez mais atenção. Pode-se dividir arbitrariamente tal mercado em três categorias: programas auxiliares, soluções de isolamento de query e soluções de conexão ao banco.

O intuito dos programas auxiliares é fornecer um suporte ao desenvolvedor

[1] Ao longo deste trabalho nos referiremos por sotaque como as diferenças existentes entre as sintaxes dos SQLs utilizados nos diferentes SGBDs.

realizando a tradução de sentenças SQL de um padrão para outro. Tais ferramentas, como o SwisSQL, realizam a conversão entre padrões comercias conhecidos, ou em alguns casos a migração de tais padrões para um padrão proprietário. O problema com o uso de tais ferramentas é que elas auxiliam apenas no desenvolvimento de queries fixas, ou seja, as queries têm que estar *hard-coded* para serem utilizadas. Através do uso de tais programas não se consegue a flexibilidade que se conseguiria com, por exemplo, o acesso a um serviço externo para a tradução.

Existem soluções já desenvolvidas que visam contornar este problema, como por exemplo o Java Database Connectivity (JDBC). Este é uma camada que permite a uma aplicação em JAVA executar diferentes sentenças SQL em diferentes SGBDs. A existência desta camada fornece ao desenvolvedor uma independência do SGBD. Tal solução porém possui uma limitação: funciona apenas em soluções desenvolvidas em JAVA.

Algumas destas soluções visam resolver o problema dos sotaques do SQL mas não resolvem um outro problema inerente a diversidade de SGBDs, as diferentes formas de realizar a conexão ao banco. Uma das soluções mais confiáveis para isso é a SQLAPI++ que resolve o problema da conexão com diferentes SGBDs.

No entanto, mesmo com as várias soluções existentes, o mercado ainda carece de uma solução que permita a conexão de uma aplicação em C++ com diversos SGBDs que utilizem diferentes sotaques de SQL. A necessidade desta solução que permita a conexão e execução de sentenças SQL em diferentes SGBDs motivou o desenvolvimento deste trabalho.

1.3 Objetivos

Como descrito anteriormente é de grande importância o desenvolvimento de uma solução que permita a conexão e execução de sentenças SQL em diferentes SGBDs em C++, possibilitando às equipes de desenvolvimento nesta linguagem um isolamento com o SGBD bem como a redução nos custos de desenvolvimento.

Neste contexto, o presente trabalho tem como objetivo descrever o projeto e a implementação de uma solução que permita a conexão e tradução de diferentes sotaques de SQL.

Para alcançar o objetivo principal deste trabalho, as metas estabelecidas são:

- Elaborar após o estudo e compreensão do problema um documento de especificação de requisitos de software da solução, detalhando as funcionalidades necessárias;
- Analisar e projetar as entidades necessárias para o desenvolvimento da solução, apoiado no paradigma da orientação a objetos;
- Implementar a solução capaz de executar as funcionalidades propostas.

A solução deverá permitir ao desenvolvedor o uso ou não do tradutor e a adição de novos tradutores quando estes se tornarem disponíveis sem a necessidade de se alterar ou recompilar o código já desenvolvido. É escolhido um sotaque de SQL como padrão e este será o sotaque de entrada para o SQL Translator. A partir deste sotaque é realizada a tradução para os outras variações de SQL dos outros SGBDs. Além disso, deve ser possível ao desenvolvedor a utilização de queries carregadas diretamente no SQL Translator ao invés de utilizar queries traduzidas através do tradutor.

Esta solução será código livre, podendo ser utilizada livremente por equipes de desenvolvimento em seus projetos.

1.4 Organização do Texto

Além deste capítulo de introdução o presente trabalho está organizado em mais 5 (cinco) capítulos.

No capítulo II, **Diferentes SGBDs e SQLs**, será apresentada a proposta deste trabalho. Serão discutidos diferentes tipos de SQL com especial atenção ao Postgree SQL e ao Oracle SQL.

No capítulo III, **Requisitos gerais do SQL Translator**, será discutida a construção do documento de especificação de requisitos do SQL Translator. Este documento define os requisitos funcionais, os requisitos de dados e as restrições que compõem o sistema, a fim de estabelecer um escopo bem definido para a realização do trabalho, servindo como base para todo o processo de desenvolvimento.

Baseado na especificação de requisitos e nas restrições impostas no documento de requisitos (Apêndice A), no capítulo IV, **Projeto do SQL Translator**, serão descritas as etapas de análise e projeto. Esse capítulo apresentará a arquitetura e o tradutor que compões a solução do SQL Translator descrevendo as soluções apresentadas. Serão discutidos os casos de uso gerados e os digramas de classe e de seqüência. Ainda neste

capitulo serão apresentadas as tecnologias utilizadas para a implementação da solução proposta.

O capítulo V, **Resultados**, apresentará os resultados obtidos, analisando a arquitetura e o tradutor.

No capítulo VI, **Conclusão,** finalmente serão descritas as conclusões obtidas, ressaltando suas contribuições e perspectivas de trabalhos futuros.

No Apêndice A, **Abreviaturas e Siglas,** estão listadas as abreviaturas usadas ao longo deste trabalho.

No Apêndice B, **Documento de requisitos**, é apresentado o documento de requisitos de software do sistema.

O Apêndice C, **Casos de uso**, documenta os casos de uso identificados durante a fase de análise do sistema. O Apêndice D, **Diagramas de Classe**, documenta o diagrama construído nas fases de análise e projeto do sistema. Tal diagrama descreve as classes idealizadas para a construção da ferramenta assim como seus atributos, métodos e relacionamentos.

No Apêndice E, **Exemplo,** é apresentado uma aplicação fictícia com acesso a dois diferentes bancos exemplificando também a tradução realizada pelo tradutor e otimizada pela biblioteca dinâmica.

No Apêndice F, **Manual de Utilização do SQL Translator,** é apresentado os procedimentos que devem ser seguidos para desenvolver uma aplicação utilizando o SQL Translator.

Nos Apêndices G e H, **Manual de Criação de Biblioteca de Query** e **Manual de Criação de Tradutor**, são descritos os procedimentos para a criação de uma biblioteca dinâmica de query ou de tradutor no padrão do SQL Translator.

Finalmente no Apêndice I, **Diagrama de Atividades do Tradutor Oracle**, é exibido todo o conjunto de atividades realizado para se executar a tradução de PostgreSQL para Oracle.

2 Diferentes SGBDs e SQLs

2.1 Introdução

Com o objetivo de facilitar o desenvolvimento de aplicações que utilizem bancos de dados, as diversas empresas que os produzem fazem modificações na estrutura, ou até mesmo na sintaxe do SQL, que vão desde pequenas até profundas alterações, além de implementarem soluções diferentes para os adendos ao SQL padrão (MELO,2002). Tais iniciativas realmente facilitam o desenvolvimento das aplicações, porém, têm a desvantagem de fazerem os desenvolvedores ficarem dependentes de uma determinada tecnologia. Portanto, a migração de uma tecnologia de SGBD para outro torna-se uma tarefa difícil tanto do ponto de vista técnico quanto do ponto de vista cultural, obrigando o desenvolvedor a adotar toda uma nova forma de realizar e pensar seus comandos SQL's.

Baseado neste contexto, este capítulo busca apresentar as diferenças entre os diversos tipos de SQL existente entre os SGBDs e ilustrar um pouco da dificuldade que pode surgir ao migrar de uma linguagem para outra.

No item 2.2 é apresentada uma breve comparação entre os diferentes tipos de SQL. No item 2.3, aprofundamos a discussão entre os SQLs do PostgreSQL e do Oracle. Em seguida fizemos uma consideração final sobre as diferenças entre eles e como isto influenciará na definição e especificação de requisitos.

2.2 Diferentes Tipos de SGBDs

Como se sabe, a diferença entre os SQLs e, em uma última análise, a diferença entre os SGBDs em si pode tornar o trabalho de conversão tão complexo que, para a maioria das aplicações, o investimento não é valido e a migração não é feita. Existe a tentativa de se padronizar o SQL através da versão SQL-92 e posteriormente da SQL-99, mas por diversos motivos este padrão não foi implementado nos SGBDs (MELO,2002).

Serão ilustrados alguns exemplos, abaixo, das diferenças existentes entre as capacidades dos SGBDs e dos sotaques de SQL.

Serão comparados os SGBDs a seguir:

- PostgreSQL versão 7.2;
- Microsoft SQL Server versão 2000;
- MySQL versão 4.1.9;
- Oracle versão 8i.

Vejamos agora alguns exemplos de queries.

2.2.1 Limitação Simples

A query com limitação simples é uma operação de busca que se faz no banco de dados onde se deseja apenas as **n** primeiras linhas do resultado, ignorando qualquer linha acima da limitação pedida. Apesar de ser uma operação relativamente simples, sua codificação nos diferentes SGBDs é realizada de maneira diferente.

Vejamos a seguir a implementação em diferentes SGBDs.

2.2.1.1 PostgreSQL

```
SELECT columns
FROM tablename
ORDER BY key ASC
LIMIT n
```

2.2.1.2 MSSQL

```
SELECT TOP n columns
FROM tablename
ORDER BY key ASC
```

2.2.1.3 MySQL

```
SELECT columns
```

```
FROM tablename
ORDER BY key ASC
LIMIT n
```

2.2.1.4 Oracle

```
SELECT * FROM (
  SELECT
    ROW_NUMBER() OVER (ORDER BY key ASC) AS rownumber,
    columns
  FROM tablename
)
WHERE rownumber <= n
```

2.2.2 Limitação de Resultado com Linhas Ignoradas

A query com limitação de resultado com linhas ignoradas é uma operação de busca que se faz no banco de dados onde se deseja as **n** primeiras linhas do resultado após ignorar (saltar) as **p** primeiras linhas do resultado, ignorando qualquer linha acima da limitação pedida.

Vejamos a seguir a implementação em diferentes SGBDs.

2.2.2.1 PostgreSQL

```
SELECT columns
FROM tablename
ORDER BY key ASC
LIMIT n OFFSET p
```

2.2.2.2 MSSQL

```
SELECT * FROM (
  SELECT TOP n * FROM (
```

```
    SELECT TOP z columns      -- (z=n+p)
    FROM tablename
    ORDER BY key ASC
  ) AS XXX ORDER BY key DESC -- ('XXX' pode representar qualquer coisa)
) AS YYY ORDER BY key ASC    -- ('YYY' pode representar qualquer coisa)
```

2.2.2.3 MySQL

```
SELECT columns
FROM tablename
ORDER BY key ASC
LIMIT n OFFSET skip
```

2.2.2.4 Oracle

```
SELECT * FROM (
 SELECT
   ROW_NUMBER() OVER (ORDER BY key ASC) AS rn,
   columns
 FROM tablename
)
WHERE rn > p AND rn <= (n+p)
```

2.2.3 Conclusão

Tal comparação, apesar de rápida, já é capaz de ilustrar a diferença existente entre os SGBDs em relação à diferença existente entre os SQLs e, conseqüentemente, suas capacidades. Tal diferença de sintaxe é o que neste trabalho é referido por sotaque. Porém não são apenas essas as dificuldades encontradas. Existem entre os SGBDs diferenças de tipos de dados e de recursos que fazem com que o processo de migração entre eles seja ainda mais complexo.

Na próxima seção abordaremos com mais detalhes as características dos dois SGBDs que serão utilizados no projeto. No **Capítulo IV** ficará mais claro o motivo da escolha destas tecnologias. Nosso objetivo é tornar possível ao leitor a familiarização com as diferenças entre eles e com as dificuldades inerentes na conversão.

2.3 PostgreSQL e Oracle

Serão apresentados a seguir, alguns exemplos, aprofundando a explicação sobre a diferença entre os sotaques de SQL do PostgreSQL e do Oracle. Abordaremos o uso dos comandos mais comuns como SELECT, INSERT, UPDATE, DELETE e uma função.

Na notação abaixo, o que estiver entre "[ ]" é uma sentença opcional, não sendo obrigatório seu uso. O que estiver entre "{ }" significa uma cláusula obrigatória. O "( )" agrupa mais de uma cadeia. O "|" significa que "um ou outro" deve ser utilizado.

Vejamos abaixo as diferenças

2.3.1 INSERT

O comando de INSERT é implementado com a seguinte sintaxe no PostgreSQL:

INSERT INTO table [(column [, ...])] { DEFAULT VALUES | VALUES (expression [, ...]) | SELECT query };

Sua implementação no Oracle é:

INSERT [HINT] INTO table_expression_clause [(column[, ...])] {[values_clause] [subquery]};

Deve ser ressaltado que table_expression_clause, values_clauses e subquery são cadeias de comandos com sua própria sintaxe.

2.3.2 UPDATE

O comando de UPDATE é implementado com a seguinte sintaxe no PostgreSQL:

UPDATE [ONLY] table SET col = expression [, ...][FROM fromlist][WHERE condition];

Sua implementação no Oracle é:

UPDATE [hint] table_expression_clause set_clause [where_clause] [returning_clause];

Deve-se ressaltar que table_expression_clause, set_clause, where_clause e returning_clause são cadeias de comando com sua própria sintaxe.

2.3.3 DELETE

O comando de DELETE é implementado com a seguinte sintaxe no PostgreSQL:

DELETE FROM [ONLY] table [WHERE condition];

Sua implementação mo Oracle é:

DELETE [hint] [FROM] table_expression_clause [where_clause] [returning_clause];

Novamente deve-se ressaltar que table_expression_clause, set_clause, where_clause e returning_clause são cadeias de comando com sua própria sintaxe.

2.3.4 SELECT

O comando de SELECT é implementado com a seguinte sintaxe no PostgreSQL:

SELECT [ALL | DISTINCT [ON (expression [, ...])]] | expression [AS output_name] [, ...][FROM from_item [, ...]][WHERE condition][GROUP BY expression [, ...]][HAVING condition [, ...]][{ UNION | INTERSECT | EXCEPT } [ALL] select][ORDER BY expression [ASC | DESC | USING operator] [, ...]] [FOR UPDATE [OF tablename [, ...]]][LIMIT { count | ALL }][OFFSET start]*

Sua implementação no Oracle é:

SELECT [hint] [{ALL|DISTINCT|UNIQUE} {|{(EXPR[[AS] c_alias])| ([SCHEMA]{TABLE|VIEW|SNAPSHOT}.*)}} FROM table_expression_clause [where_clause] [([group_by_clause] [hierarchical_query])] [{MINUS |INTERSECT| UNION[ALL]}(subquery)] [order_by_clause];*

Deve ser ressaltado que o from_item (da implementação em PostgreeSQL), table_expression_clause, where_clause, group_by_clause, hierarchical_query, subquery e oreder_by_clause são cadeias de comando com sua própria sintaxe.

2.3.5 Outras diferenças

Dentre outras diferenças, podemos citar por exemplo o DBLINK. No PostgreSQL este é implementado como uma função que pode ser utilizada pelo usuário (*SELECT * FROM dblink('dbname=mydb', 'select proname, prosrc from pg_proc') AS t1(proname name, prosrc text) WHERE proname LIKE 'bytea%';*) enquanto no Oracle este é substituído pelo caracter '@', fazendo parte da própria sintaxe da query (*select * from (select from name, prosrc from pg_proc@mydb) t1*).

O caso do DBLINK é apenas um dentre as várias diferenças existentes entre os sotaques de SQL e o que cada um implementa como função ou como parte da sintaxe. Este serve para ilustrar a dificuldade existente na tarefa de migração entre diferentes SGBDs.

2.4 Conclusão

Neste capítulo, em um primeiro momento foram abordados exemplos sobre a diferença existente entre alguns sotaques de SQL, e em um segundo momento, a diferença entre o PostgreSql e o Oracle foi vista mais a fundo.

A diferença entre os sotaques foi a primeira motivação da execução deste projeto. A escolha dos SGBDs alvos deste projeto foi feita levando-se em conta essa diferença e a posição deles no mercado, sua representatividade e sua capacidade. No que diz respeito aos requisitos, o estudo entre os diferentes sotaques contribui para a elaboração de alguns dos seus requisitos funcionais. O documento de especificação de requisitos pode ser encontrado no Apêndice B e será discutido no capítulo a seguir.

3 Requisitos gerais do SQL Translator

3.1 Introdução

Os modelos de desenvolvimento de software têm como entrada o requisito de um sistema e como saída o produto. O modelo de desenvolvimento utilizado foi o seqüencial linear, que inicia na especificação de requisitos e segue pela análise do projeto, codificação e teste (Pressman,2002). Deste modo, apenas quando se completa uma etapa se inicia a etapa seguinte. Assim apenas quando tivermos os requisitos descritos de forma completa e consistente é que poderemos iniciar a etapa seguinte.

Neste capítulo será discutida a especificação de requisitos e a construção do documento de requisitos do SQL Translator. No item 3.2 serão abordadas as restrições do projeto com a apresentação das tecnologias utilizadas para o desenvolvimento, justificando a sua escolha. Nos itens 3.3 e 3.4 serão discutidos os requisitos de software e o processo de identificação e análise dos requisitos. Trataremos nesta seção dos requisitos de dados e requisitos funcionais.

O objetivo principal deste capítulo é apresentar ao leitor o documento de especificação de requisitos produzidos, discutindo a metodologia utilizada para definição e construção do documento. A versão final do documento de requisitos de software utilizado neste projeto encontra-se no Apêndice B.

3.2 Restrições de Projeto

Geralmente, as restrições do sistema a ser construído determinam as tecnologias a serem usadas, mas sua escolha só será feita mais adiante, na fase de projeto do sistema. No entanto, como o presente trabalho visa suprir a carência do mercado por um determinado tipo de solução tecnológica, as escolhas de algumas tecnologias foram definidas em sua fase de concepção (como a utilização de C++ e SQLAPI++). Outras escolhas de tecnologia (PostgreSQL e Oracle SQL) foram realizadas a partir do que foi julgado mais representativo em termos de mercado, e, que pudesse ser mais rapidamente assimilado pelas equipes de desenvolvedores. Baseado no documento de requisitos do Apêndice B, as seguintes seções apresentam as tecnologias que serão utilizadas no

desenvolvimento da ferramenta, com uma descrição e justificativa de sua escolha.

3.2.1 PostgreSQL

O PostgreSQL é um dos mais conhecidos e poderosos SGBDs relacionais *open source* disponíveis no mercado. Curiosamente chamado pelos brasileiros de "Postgree", o PostgreSQL já consolidou sua boa reputação, demonstrando sua versatilidade tanto em recursos quanto ao suporte a maioria dos sistemas operacionais de destaque como: Linux, BeOS, Windows e Unix.

O PostgreSQL é totalmente multitransacional e oferece total suporte à *foreign keys, joins, views, triggers,* e *stored procedures*. Ele inclui a maioria dos tipos de dados do SQL-92 e SQL-99 e mais funções como otimização de *query,* replicação assíncrona, backup on-line e suporte ao conjunto de caracteres internacionais (Unicode).

Sua escolha para o projeto foi feita devido o seu suporte à maioria das funcionalidades disponíveis para banco de dados e a implementação a maioria dos tipos de dados existentes. O fato de ser *open source* também influenciou sua escolha para uso neste projeto bem como a escolha dele para ser a linguagem padrão de entrada do sistema. O PostgreSQl foi escolhido como o SQL padrão para entrada de queries no SQL Translator e a versão utilizada do PostgreSQL foi a 7.2.

3.2.2 Oracle

O Oracle é o líder de mercado no segmento de SGBD sendo também a referência para o mercado em inovação tecnológica e funcionalidades. Contendo uma vasta gama de aplicações integradas como processamento em GRID e alta disponibilidade com *desaster recovery*, sua estrutura permite adequação às necessidades específicas do cliente, seja ele uma pessoa física ou uma grande empresa multinacional.

A tradição e *know-how* fazem com que programadores procurem aprender o seu sotaque de SQL em adição a qualquer outro que esteja sendo efetivamente implementado em seus projetos. A licença, contudo, possui um custo muito elevado, o que dificulta uma maior popularização do software e o seu acesso por empresas menores.

Sua escolha para o projeto foi feita devido à sua grande representatividade no

mercado e o desejo por parte dos programadores de aprenderem seu sotaque de SQL para uso nas aplicações. O Oracle é o primeiro SGBD a ter o tradutor implementado.

3.2.3 C++

A linguagem de programação C++ é uma linguagem orientada a objetos (OO), sendo atualmente a linguagem hegemônica na implementação de projetos que utilizam este paradigma. Os conceitos de orientação a objetos como classes, herança, polimorfismo, sobrecarga de métodos, sobrecarga de operadores entre outros, podem ser perfeitamente implementados com esta linguagem, além de sua maior velocidade e melhor desempenho quando comparado com outras soluções que também utilizam o paradigma OO, oferecendo um ambiente ideal para desenvolver aplicativos.

O Software Development Kit utilizado no desenvolvimento do SQL Translator foi o *Visual C++ 6.0* da Microsoft. O *Visual C++* fornece um ambiente completo e profissional para desenvolvimento, além de fornecer um ambiente completo para *debug* e execução dos aplicativos construídos.

Devido ao seu intuito de preencher uma lacuna existente no conjunto de soluções de desenvolvimento existentes em C++, esta foi a escolha óbvia para o desenvolvimento do projeto.

3.2.4 SQLAPI++

SQLAPI++ é uma biblioteca para acessar diversos tipos de SGBD como Oracle, PostgreSQL, SQL Server, MySQL, etc. A biblioteca utiliza as APIs nativas dos SGBDs de forma que as aplicações desenvolvidas com SQLAPI++ rodem rápido e eficientemente. Ela também oferece uma interface de baixo nível para acesso a características específicas dos SGBDs. Através do encapsulamento de APIs, o SQLAPI++ age como camada intermediária para os programas, criando portabilidade para os SGBDs. Para utilizar a SQLAPI++ é necessário adquirir sua licença.

Sua escolha foi feita devido à sua característica multiplataforma e à sua já reconhecida confiabilidade que se mostram imprescindíveis para implementação deste projeto.

3.2.5 VBLIB

A VBLib é uma biblioteca desenvolvida em C++ para uso em aplicações em geral (VILLAS-BOAS,2005). Ela implementa diversas funcionalidades que visam facilitar o desenvolvimento de aplicações em C++. A VBLib é uma biblioteca gratuita que pode ser utilizada nos projetos com as restrições previstas no *GNU Lesser General Public Licens.*

A VBLib é utilizada no seu projeto devido à implementação de uma poderosa classe de manipulação de *strings* (a VBString) e às suas outras funcionalidades que auxiliarão no desenvolvimento do projeto.

3.3 Requisitos de Software

Todo o projeto possui uma finalidade, um objetivo. Quando um contratante requisita a construção de um sistema, ele possui uma noção em alto nível do que o sistema deve fazer. Porém, o contratante nem sempre sabe exatamente como expressar o que quer, ou não possui uma idéia completa do que necessita. Para que se possa identificar corretamente as funcionalidades do sistema pedido, faz-se necessária a definição e construção de um documento de requisitos de software, com o intuito de formalizar as funcionalidades e características do sistema.

Um requisito pode ser uma característica do sistema ou uma ação que o sistema deve realizar. Ele deve ser capaz de descrever suas características e ações de forma clara, de modo que tanto o cliente quanto o desenvolvedor possam ter a mesma compreensão do objetivo do sistema e de como ele deve funcionar. O conjunto de requisitos irá compor um documento, assegurando ao desenvolvedor de que estará atendendo às expectativas do contratante,e a este, serve de instrumento para cobrança e validação do que foi desenvolvido. A especificação de requisitos não descreve como o sistema será implementado, apenas o comportamento do sistema, identificando "*o que"* o sistema deve fazer, e não "*como fazer*", sem se preocupar com os detalhes de implementação.

Um dos principais requisitos define que o SQL Translator trabalhe com múltiplos SGBDs. Para que isto seja possível o SQL Translator não realiza a tradução para as diretivas de otimização específicas de cada banco. Com isso uma query que tenha sido

otimizada será traduzida com perda de desempenho. Para que o SQL Translator possa ser utilizado em conjunto com essas diretivas de otimização é permitido ao desenvolvedor a criação de bibliotecas de queries otimizadas para um SGBD específico de forma que não haja perda no desempenho. Este tipo de query será referido ao longo do texto como query traduzida manualmente. Este procedimento é descrito no Apêndice E. Além disso, o SQL Translator deve ser capaz de reconhecer a adição de um novo tradutor automaticamente sempre que for reiniciado.

Esses requisitos estão descritos de forma detalhada no documento de especificação de requisitos do sistema, no Apêndice B, sendo importante sua leitura ao fim deste capítulo.

3.4 Identificação e Especificação dos Requisitos

A identificação de requisitos de software envolve uma metodologia que visa a correta identificação e especificação das características e ações a serem realizadas. Deve-se identificar todas as necessidades dos usuários e descrever o problema a fim de verificar se estamos utilizando o ponto de vista correto. Cada requisito lida com objetos ou entidades, definindo os estados que estes podem possuir e as ações que devem ser realizadas para modificar suas características.

Com o intuito de entender o problema a ser resolvido, foi levantado com o professor Villas-Boas um grupo de características iniciais que o projeto deverá atender. Este conjunto de características foi levantado primeiramente com um alto nível de abstração. Em seguida, foi realizado um aprofundamento na literatura indicada pelo professor, bem como uma pesquisa ampla de mercado das soluções existentes. Após algumas reuniões onde os requisitos foram discutidos, estes foram praticamente concluídos. Para finalizar, foi feita ainda uma etapa de validação com o objetivo de garantir que os requisitos estivessem completos, corretos e consistentes.

Para descrever os requisitos de forma completa, testável e procurando evitar ambigüidades, foram observadas as seguintes técnicas para especificação dos requisitos:

- Cada cláusula deve conter somente um requisito;
- Os requisitos semelhantes devem ser colocados juntos;
- Deve-se evitar que um requisito faça referência a outro requisito;
- Cada advérbio e adjetivo deve ter uma descrição quantitativa, de modo que o

significado dos qualificadores seja claro e não ambíguo;

- Os pronomes devem se r trocados pelos nomes específicos das entidades;
- Todo nome deve ser definido em um local exato nos documentos dos requisitos;

(TRAVASSOS,2004)

O documento de especificação de requisitos foi construído com base nestes conceitos. Tal documento é iniciado com uma descrição do objetivo a que ele se propõe. As seções seguintes tratam da definição dos requisitos. Para tanto, procurou-se agrupá-los em três classes distintas, de acordo com suas características: Requisitos de Dados, Requisitos Funcionais e Restrições.

Os **Requisitos de Dados** buscam identificar as entidades envolvidas, descrevendo seus atributos de forma quantitativa. Essa visão é estática, porque não descreve como os relacionamentos se modificam ao longo do tempo (PFLEEGER, 2004).

Na especificação dos requisitos de dados foi adotado o seguinte padrão: primeiro é feita uma descrição da entidade com seus atributos. Em seguida, é feita uma especificação quantitativa dos atributos, procurando determinar tipos e limites para cada atributo. Em especial, no caso da cadeia de caracteres, este limite não foi definido devido o uso da VBLIB (anteriormente descrita), que permite a manipulação de uma cadeia de caracteres ilimitada. A seguir, temos um exemplo da descrição de uma query:

Uma query deve possuir query Id, Tipo do banco de dados, Texto SQL, Tipo de Query.

Descrição de dados:

Query Id	*Inteiro positivo*
Tipo do BD	*Tipos pré-definidos em uma lista*
Texto SQL	*String*
Tipo de Query	*Assume os valores: Saída, Entrada*

Os **Requisitos Funcionais** procuram capturar as ações a serem realizadas pelo sistema. Um requisito funcional descreve uma interação entre o sistema e seu ambiente. Além disso, os requisitos funcionais descrevem como o sistema deve se comportar, considerando um certo estímulo. (PFLEEGER, 2004). Para determinação dos requisitos funcionais, foram decididas quais ações são aceitáveis para o SQL Translator.

Dentre os principais requisitos funcionais identificados, podemos citar:

- O SQL Translator deve permitir a habilitação ou não do tradutor;
- O SQL Translator deve permitir a utilização de uma ou mais queries traduzidas de forma manual pelo usuário, sobrepondo assim uma query traduzida;

- O SQL Translator deve carregar automaticamente qualquer novo tradutor que venha a ser desenvolvido.

As **Restrições** limitam nossas opções para criar uma solução para o problema. Podem impor quais as tecnologias devem ser utilizadas, como os sistemas operacionais que devem ser suportados, as ferramentas de desenvolvimento e as bibliotecas a serem utilizadas e a metodologia de desenvolvimento.

A seguir, são apresentadas algumas restrições impostas no documento de requisitos:

- A ferramenta deve ser implementada utilizando a linguagem de programação orientada a objetos C++;
- O acesso aos bancos de dados deve ser realizado através da SQLAPI++;
- Para as funcionalidades em geral e classe de *string* devem ser utilizadas as soluções da VBLIB, quando estas existirem.

Estas seções compõem o documento de requisitos, que se encontra no Apêndice B.

3.5 Conclusão

Em qualquer projeto de software, uma das etapas mais importantes é a especificação de requisitos. Requisitos mal especificados são uma das maiores fontes de erro durante a implementação de projetos. Estudos realizados afirmam que quanto mais tarde se detecta um erro em um requisito do sistema, maior será o custo para a correção do mesmo (PRESSMAN,2002). Desta forma, procurou-se neste trabalho dar atenção especial aos requisitos da ferramenta, destacando um capítulo inteiro para sua discussão.

Através de pesquisa bibliográfica, o documento de requisitos foi construído baseado na metodologia do padrão IEEE. Procurou-se construir o documento de acordo com esta metodologia, identificando e especificando os requisitos, com o objetivo de descrevê-los de forma completa e consistente, evitando assim possíveis ambigüidades. Além disso, foram identificadas também as tecnologias a serem utilizadas, fornecendo um cenário completo para o projeto e implementação da ferramenta.

Antes do prosseguimento aos próximos capítulos, é importante que o leitor se reporte ao Apêndice B, **Documento de Requisitos,** para um melhor entendimento dos requisitos do sistema, uma vez que o projeto e implementação da ferramenta são a

realização desses requisitos de software.

4 Projeto do SQL Translator

4.1 Introdução

Neste capítulo serão descritas as fases de análise e projeto do SQL Translator. Serão apresentados a definição do problema, o diagrama de classes do SQL Translator e a solução proposta para construção da arquitetura e do tradutor. Em seguida serão discutidos os casos de uso identificados e os diagramas de seqüência desenvolvidos para representar as ações a serem realizadas pelo sistema. Finalmente serão discutidos alguns aspectos de implementação.

4.2 Definição do Problema

O SQL Translator deve ser a arquitetura híbrida que forneça flexibilidade para o desenvolvedor trabalhar com diferentes SGBDs. Ele deve permitir a adição posterior de novos tradutores bem como a desabilitação total ou parcial do tradutor (através do uso de queries otimizadas) e a conexão com múltiplos SGBDs, inclusive simultaneamente. A conexão com os SGBDs deve ser feita através da SQLAPI++ que é uma biblioteca comprovadamente eficiente e consistente. Todas essas funcionalidades devem ser multiplataforma.

Baseando-se nessas características do sistema e no documento de especificação de requisitos, a seguir serão apresentados a seguir o diagrama de classes e a solução proposta para a sua construção.

4.3 Diagrama de Classes

O diagrama de classes foi projetado a partir das necessidades impostas pelo documento de requisitos (Apêndice B) e está representado pelos diagramas da figura 1.

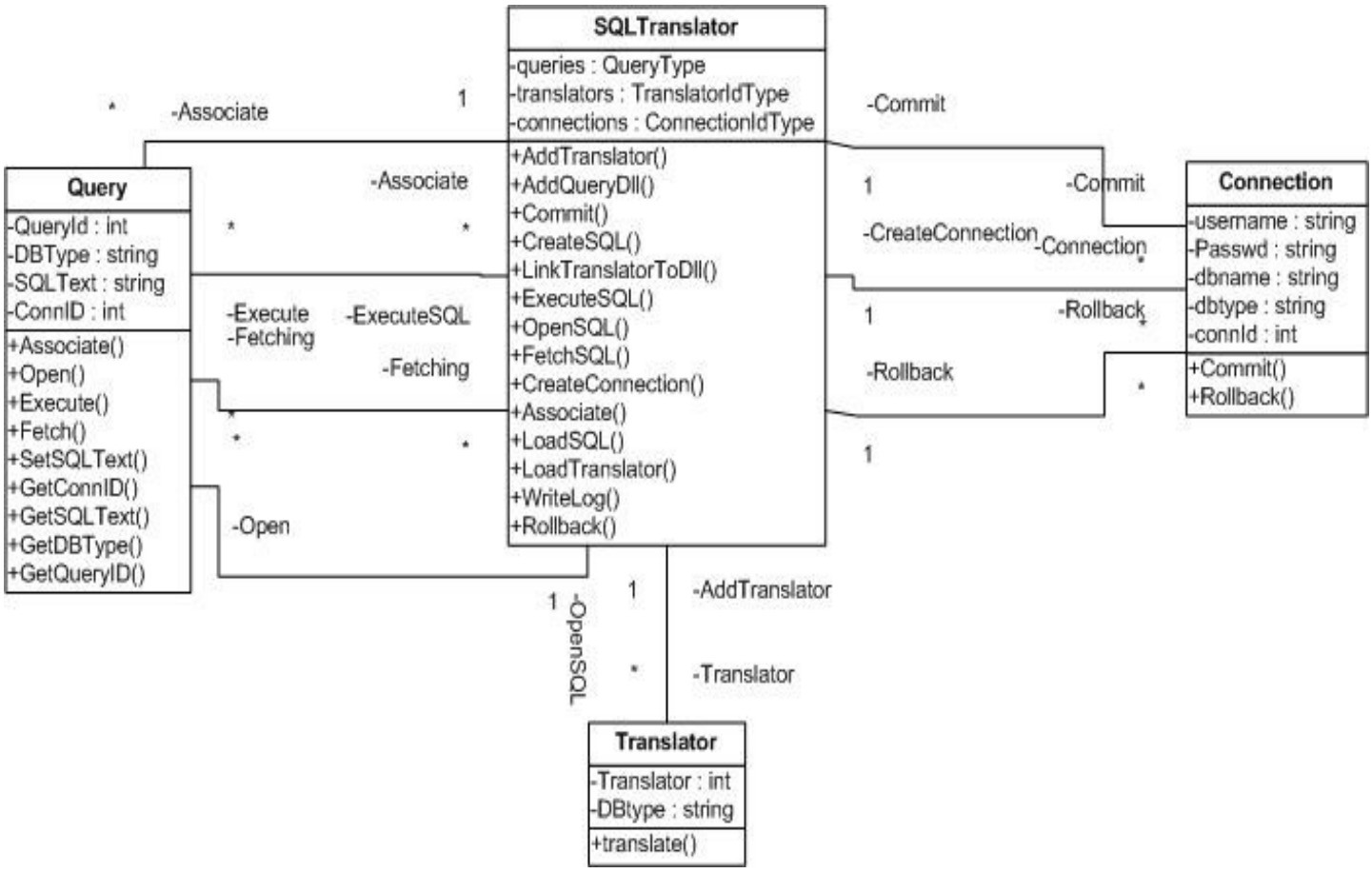


Figura 1 – Diagrama de classes estático do SQL Translator

O diagrama de classes estático pode ser observado na figura 1 acima (ou no Apêndice D). Devemos chamar a atenção para a classe de interface SQLTranslator. Esta classe é responsável por realizar a interface com o programa e acessar todos os serviços internos.A classe SQLTranslator também é responsável por gerenciar o *pool* de queries, tradutores e conexões.

Além da identificação dos objetos, foram identificados também os atributos de cada objeto, que são capazes de descrever seus possíveis estados ao longo do tempo. Esse diagrama de classe foi construído de acordo com as restrições impostas pelos "Requisitos de Dados", item 3 do Apêndice B.

4.4 Decisões de projeto

Os SGBDs que serão utilizados neste projeto são o PostgreSQL (versão 7.2) e o Oracle (versão 8i). O PostgreSQL foi escolhido como SGBD padrão para o sistema por

ser a opção *open source* mais completa. Além disso, o PostgreSQL foi escolhido devido a uma tentativa de se estimular o uso de soluções *open source*. Deve ser lembrado que se a query de entrada do SQL Translator for uma query em PostgreSQL esta não será traduzida, sendo diretamente executada no banco.

O SGBD escolhido como destino para o primeiro tradutor foi o Oracle, por ser o mais representativo no mercado, fornecendo às equipes de desenvolvimento que não estão acostumados com o uso de tal sotaque de SQL uma maneira fácil de iniciar o seu trabalho. Com a evolução deste projeto outros tradutores para outras linguagens serão acrescentados.

Para este projeto inicial decidiu-se trabalhar com dois sistemas operacionais, Windows e Linux, sendo que o suporte a outros sistemas será adicionado com a evolução natural do projeto.

4.5 Solução proposta

A seguir é descrita a solução adotada para a arquitetura e para o tradutor.

4.5.1 Arquitetura

A solução proposta para a arquitetura baseia-se nos seus requisitos e nas funcionalidades fornecidas pela linguagem C++. Devemos lembrar também que o PostgreSQL foi escolhido como SQL padrão o que significa que todas as queries recebidas pelo SQL Translator devem ser deste formato. Para ilustrar o funcionamento da arquitetura é implementado um tradutor para o SQL do SGBD Oracle.

Em primeiro lugar, deve-se analisar o seu objetivo de facilitar o desenvolvimento de outras aplicações. Para que isto seja possível foi escolhido disponibilizar o SQL Translator através do seu próprio fonte (arquivos .cpp e .h) para que o desenvolvedor possa utilizá-lo livremente. Esta solução tem a vantagem de permitir que desenvolvedores externos estudem e sugiram modificações auxiliando assim no amadurecimento dela. A interface e o isolamento entre o programa em desenvolvimento e o SQL Translator é feito pela classe de mesmo nome, “SQLTranslator”.

Também foi escolhido utilizar os componentes disponibilizados pela SQLAPI++. A reutilização de código ou de componentes é prática comum e benéfica na

programação, sendo prevista pelos paradigmas da OO. A SQLAPI++ é uma das soluções multiplataforma mais completas e robustas para uma conexão em bancos de dados. Também foi levada em consideração a experiência já adquirida no uso desta ferramenta.

Outra biblioteca utilizada no desenvolvimento da arquitetura foi a VBLib. Esta contém várias soluções de propósito geral que visam facilitar diversas tarefas durante a programação.

Para que seja possível a adição de tradutores e de queries sem que seja necessário a recompilação do código, estes devem ser implementados através de bibliotecas dinâmica com ligação explicita. A vantagem desta solução consiste no fato de permitir tanto a adição quanto a substituição de novos tradutores e queries. As queries criadas desta maneira são as queries traduzidas manualmente. Uma query traduzida manualmente é uma query em que o desenvolvedor optou por traduzir para o SGBD de destino de maneira manual, ou seja, sem a passagem pelo tradutor. Tal situação pode ser útil quando o desenvolvedor achar necessário inserir deretrizes de otmização de SQL na query traduzida. Deve ser lembrado que o tradutor não é capaz de otimizar queries nem traduzir otimizações.

Para a carga das bibliotecas de tradutores e de queries, a arquitetura faz uma varredura no diretório passado pelo usuário ou no caminho indicado pela variável de ambiente “SQLTranslator_Home”. Após carregadas, as queries e os tradutores são armazenados em um *pool* gerenciado pela classe de interface, que é acessada durante o tempo de execução sempre que for necessário a busca de alguma informação. As bibliotecas devem estar no padrão especificado para a arquitetura. Mais informações sobre este padrão podem ser obtidas através do Apêndice G ou H.

4.5.2 Tradutor

A solução proposta para o tradutor levou em conta, principalmente, o requisito de que este pode ser tanto adicionado posteriormente quanto substituído, sem a necessidade de recompilação do código. Para se alcançar este objetivo foi escolhido implementar o tradutor como biblioteca dinâmica (DLL, no caso do Windows ou, SO, no caso do Linux).

O tradutor é responsável por receber como parâmetro uma *string*, que corresponde

a query a ser traduzida, e retornar uma *string* com o resultado da tradução. Por esta ser uma operação de manipulação de *strings,* foi decidido utilizar a VBLib e sua a classe de manipulação de strings VBString.

Com relação ao universo das queries a serem traduzidas foi decidido dedicar um foco maior aos comandos de manipulação de dados em detrimento dos comandos que manipulassem e alterassem o banco de dados em si. Esta decisão foi tomada porque, em sua operação diária, as operações de manipulação de dados como INSERT,UPDATE,SELECT, etc, são muito mais comuns que operações de alteração do banco de dados como CREATE TABLE, ALTER TABLE, CREATE USER, etc.

4.6 Comportamento Dinâmico

A visão dinâmica do SQL Translator se preocupa em descrever as ações que serão desenvolvidas e os elementos envolvidos em cada ação. Este cenário é baseado nos requisitos funcionais, descritos no item 4 do documento de requisitos, contido no Apêndice B. Como se utiliza no desenvolvimento da ferramenta o paradigma OO, os requisitos funcionais são representados por casos de usos. Um caso de uso descreve a funcionalidade específica que um sistema, supostamente, deve desempenhar ou exibir, por meio da modelagem do diálogo que um usuário, um sistema externo ou outra entidade terá como o sistema a ser desenvolvido. Cada caso descreve um possível cenário de como uma entidade externa interage com o sistema, identificando-se todos os possíveis eventos (PFLEEGER, 2004). Além de ampliar e esclarecer os requisitos, os casos de uso podem ajudar a encontrar defeitos nos mesmos, caso estes não tenham sido bem definidos.

Entre os principais casos de uso podemos destacar:

Carregar Queries via Biblioteca Dinâmica com Ligação Explicita:

1. Verifica se há alguma biblioteca dinâmica no diretório especificado;
2. Verifica as bibliotecas dinâmicas uma a uma;
3. Para cada biblioteca dinâmica que está no padrão, uma query é carregada;

Associar Query a Conexão:

1. Verifica se existe a conexão;

2. Verifica se existe a Query a ser associado;
3. Chama o caso de uso "Traduz Query" caso seja necessário traduzir;
4. Associa a Query a conexão;

Traduzir Query:

1. Interpreta SQL de Entrada;
2. Gera SQL de saída para uso em outro banco;

Além dos casos de uso aqui apresentados, existem outros casos de usos. Os diagramas de caso de uso completos, criados para auxiliar a construção do sistema, encontram-se no Apêndice C.

Os casos de uso fornecem uma visão que está basicamente focada na funcionalidade do sistema sem se preocupar com os detalhes de implementação ou com o projeto do sistema. Faz parte da análise do sistema, auxiliando na identificação dos objetos envolvidos nas ações a serem realizadas pelo mesmo.

Após a construção dos casos de uso e identificação das classes envolvidas nas ações do sistema, devemos ser capazes de descrever tais ações em métodos de classes.

Também foi levantado o diagrama de estados da classe query. Esta é a única classe que sofrerá modificação em seu estado ao longo da execução do programa. O diagrama de estados pode ser observado abaixo:

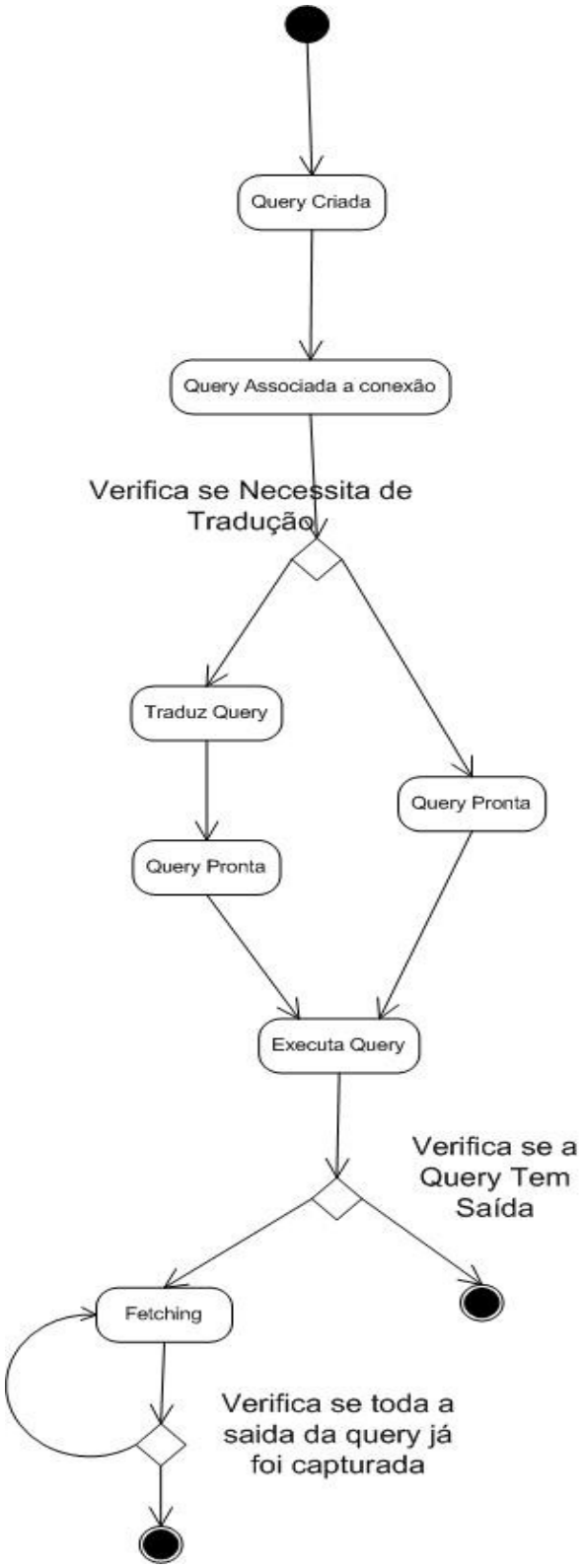


Figura 2 - Diagrama de Estados da Classe Query

4.7 Conclusão

Neste capítulo discutimos a análise e projeto do sistema, identificando as classes necessárias para a realização do projeto, assim como seus relacionamentos, atributos, métodos e possíveis estados. A solução proposta buscou explorar as vantagens do paradigma OO, no intuito de facilitar o desenvolvimento e a manutenção futura do código. Também foram utilizados os recursos que a linguagem C++ dispõe, alcançando assim uma solução técnica satisfatória para os requisitos levantados.

5 Resultados

5.1 Introdução

Os resultados finais do projeto, como as principais funcionalidades e características implementadas, serão apresentados neste capítulo. Será apresentado o resultado obtido com a arquitetura, com o tradutor e como deve ser feita a sua utilização. Também demonstraremos um exemplo de aplicação utilizando o SQL Translator.

5.2 Resultados Obtidos com a Arquitetura

O principal resultado com a arquitetura foi o desenvolvimento do código fonte que pode ser utilizado em projetos de terceiros, auxiliando no desenvolvimento de aplicações que utilizem SGBD. O fonte da arquitetura pode ser utilizado tanto no sistema operacional Windows quanto no sistema operacional Linux. A classe "SQLTranslator" fornece a interface necessária para o uso do tradutor através de seus métodos.

Também foi conseguido que, ao se iniciar um programa que tenha sido desenvolvido utilizando o fonte, este faça um levantamento de todos os tradutores e queries que estão disponíveis para uso e os armazene em uma *pool*. Com isto, obtém-se duplo benefício pois, além de permitir a substituição de tradutor ou query com a simples adição da nova biblioteca em substituição da antiga, permite a adição de novos tradutores ou queries. As bibliotecas do tradutor e da query são bibliotecas dinâmicas com ligação explicita. No caso específico dos tradutores, estes só serão carregados efetivamente em memória se forem utilizados durante a execução, objetivando assim uma melhora no desempenho.

Também foi elaborado um procedimento para ensinar o processo de criação de bibliotecas para a adição de query. As bibliotecas do tradutor e da query são bibliotecas dinâmicas com ligação explicita. O procedimento para a criação das bibliotecas pode ser obtido nos apêndices G e H. Com isso o desenvolvedor ganha a flexibilidade de substituir qualquer tradução que possa vir a ser realizada pelo tradutor por uma versão

mais otimizada de query feita manualmente.

5.3 Resultados Obtidos com o Tradutor

O principal resultado com o tradutor foi uma biblioteca que realiza a tradução para o SQL do Oracle 8i, com a abrangência de um número bastante representativo de casos. Além das sintaxes mais comuns deve-se destacar a implementação da tradução para o DBLINK e para os diferentes tipos de *JOINS*. Estes se mostraram os maiores desafios durante a implementação do tradutor e, possivelmente, serão também a maior dificuldade para a implementação da tradução dos futuros tradutores. A estrutura na qual o tradutor foi desenvolvido permite que este seja atualizado futuramente, acompanhando assim a evolução natural dos SGBDs.

Um diagrama das tividades necessárias para a implementação do tradutor Oracle pode ser obtido no Apêndice I.

5.4 Exemplo

O Apêndice F, **Exemplo,** apresenta um exemplo de aplicação utilizando o SQL Translator. Este exemplo apresenta as mensagens de retorno de um programa que realiza a conexão a um SGBD PostgreeSQL e um banco Oracle. Pode ser visto a carga do tradutor e de queries implementadas manualmente. O Programa também realiza a tradução de algumas queries para exemplificar o funcionamento do tradutor.

5.5 Conclusão

Neste capítulo apresentamos os resultados obtidos pela arquitetura e pelo tradutor. Pudemos verificar através do exemplo que o SQL Translator se comportou de forma satisfatória, atendo as expectativas. Devemos lembrar que o exemplo foi elaborado com as queries mais comumente usadas no dia-a-dia de um sistema. Portanto, podemos afirmar que o SQL Translator suporta um universo representativo das solicitações que são comumente realizadas.

6 Conclusão

Desde sua criação, os diversos tipos de SGBDs foram, ao passar do tempo, incrementando suas funções e formas de uso. Estas modificações apesar de benéficas para os SGDBs em si, culminaram na criação de sotaques e, com isso, problemas quando se tem como objetivo desenvolver ferramentas que trabalhem com um SGBD que o desenvolvedor não esteja familiarizado ou quando se busca realizar uma migração de um SGBD para outro.

Por outro lado, com o objetivo de facilitar o desenvolvimento de aplicações, diversas soluções de software gratuitas são produzidas e disponibilizadas para que as equipes de desenvolvimento possam reduzir esforço e custo, bem como minimizar os riscos, reutilizando uma solução comprovadamente eficiente. Como fruto desta filosofia de pensamento podemos citar, por exemplo, as bibliotecas VBLIB e VBMCGI (VILLAS-BOAS, 2005) (entre muitas outras bibliotecas disponíveis na Internet). Também existem soluções disponíveis comercialmente, como é o caso da SQLAPI++, em que há o pagamento de uma licença para a utilização da mesma. Ambas as soluções auxiliam o trabalho das equipes de desenvolvimento, sendo a decisão de se produzir uma solução comercial ou gratuita uma decisão meramente estratégica.

Neste contexto, o objetivo deste projeto foi criar uma camada de isolação entre o programa e o SGBD, fornecendo meios de realizar uma unificação dos SQLs no sotaque do PostgreSQL, deixando a cargo da ferramenta fazer a tradução para o SGBD de destino. Esta camada é o SQL Translator, que é responsável tanto pela tradução das queries quanto pela conexão com o SGBD, utilizando para isso a SQLAPI++. O desenvolvimento desta ferramenta seguiu os requisitos propostos atendendo assim a uma necessidade pela comunidade de desenvolvedores em C++ de uma ferramenta que auxiliasse o desenvolvimento de sistemas voltados para SGBD.

6.1 Resultados

Como resultado deste projeto temos o SQL Translator, que é composto pelo seu código fonte, representando a sua arquitetura, e uma biblioteca dinâmica de um tradutor. O SQL Translator também permite a atualização de um tradutor ou query pela simples

substituição de uma biblioteca dinâmica, sem a necessidade da recompilação de nenhuma parte do código. Também foi elaborado um processo de como devem ser desenvolvidas as bibliotecas de queries para serem acrescentadas ao sistema.

Foi implementado o primeiro tradutor de SQL, de PostgreSQL para Oracle, com o intuito de demonstrar a flexibilidade do SQL Translator, bem como ampliar a sua usabilidade na primeira versão. Com o uso do tradutor, o desenvolvedor ganha uma isolação do SGBD utilizado pois este só se preocupará em aprender e implementar o sistema em um sotaque de SQL, o do PostgreSQL, sendo o trabalho de tradução deixado para o tradutor específico.

Em termos de aprendizado, o desenvolvimento deste projeto possibilitou o aprofundamento dos conceitos em engenharia de software orientada a objetos, permitindo o contato com todos os aspectos da construção de um sistema utilizando este paradigma, aprimorando os conhecimentos em especificação de requisitos e análise, permitindo ainda aprimorar os conhecimentos das principais tecnologias utilizadas (C++, Visual C++ 6.0, SQLAPI++). Também foi possível ampliar o conhecimento em SQL, em especial nos sotaques de Oracle e PostgreSQL.

6.2 Metodologias Utilizadas

As principais metodologias utilizadas para o desenvolvimento deste trabalho foram:

- Pesquisa Bibliográfica;
- O ciclo de vida do processo de desenvolvimento de software adotado foi o modelo seqüencial linear;
- Para identificação dos requisitos, foram feitas reuniões com o orientador com o intuito de extrair o máximo de informações sobre o sistema a ser desenvolvido. Os requisitos foram separados em requisitos de dados, requisitos funcionais e restrições. Foram feitas ainda revisões periódicas do documento de requisitos pelos autores deste trabalho, buscando eliminar requisitos potencialmente redundantes, incompletos ou ambíguos.
- Na análise e projeto do sistema utilizaram-se as metodologias abordadas pela engenharia de software, especificamente os conceitos ensinados nas

aulas de Engenharia de Software OO, ministrada pelo professor Guilherme Horta Travassos, e abordados nos livros, “Engenharia de Software”, Pressman - 2002 e “Engenharia de Software - Teoria e Prática”, Pfleeger - 2004.

6.3 Trabalhos Futuros

O desenvolvimento do projeto consiste em uma progressiva expansão de sua capacidade, buscando estender sua compatibilidade a mais sistemas operacionais e outros SGBDs, bem como suas futuras atualizações.

Portanto, nas próximas atualizações, será implementado o tradutor para MySQL, SGBD *open source* como o PostgreSQL, e acrescentado o suporte ao sistema FreeBSD.

Referências

ARVIN, T., *Comparison of different SQL implementations*, Disponível em: <http://troels.arvin.dk/db/rdbms/>. Acesso em: 17/04/2005.

DEXTRA, *Banco de Dados PostgreSQL,* Disponível em: <www.dextra.-com.br/postgres/>. Acesso em: 13/04/2005

GONZÁLEZ, V. H. D., LAMAS, F. M., FABRE, J. N. J., *Curso de Bases de Datos y PostgreSQL,* Disponíve em: <www.apresendergratis.com>. Acesso em: 09/06/2005.

ISOTTON, A., *C++ dlopen mini HOWTO*, Disponével em: <http://www.isotton.-com/howtos/C++-dlopen-mini-HOWTO/C++-dlopen-mini-HOWTO.html>. Acesso em: 29/07/2005.

LONEY, K. KOCH, G., *Oracle 91: The complete Reference*, 1 ed., California: McGraw-Hill, 2002.

MAILSOFTWARE, *Open Source Database Software Comparison,* Disponível em: <http://www.geocities.com/mailsoftware42/db/>. Acesso em: 15/04/2005

MELO, D., *É possível padronizar o acesso a banco de dados?,*. Disponível em: <http://phpbrasil.com/articles/article.php/id/240>. Acesso em: 13/04/2005.

MOMJIAN, B., *PostgreSQL: Introduction and Concepts*, 1 ed., New Jersey: Addison-Wesley, 2001

ORACLE, *SQL Reference, Release 8.1.5,* 1999

_____, *Introduction to Oracle 9i: Volume 1,2001*

_____, *Introduction to Oracle 9i: Volume 2,2001*

_____, *Technology Solutions*,Disponível em: <http://www.oracle.-com/technologies/technology_solutions.html>. Acesso em: 16/04/2005.

PFLEEGER, S. L., *Engenharia de Software: teoria e prática* , 2. ed , São Paulo: Prentice Hall. , 2004.

POSTGRESQL, *About PostgreSQL*, Disponível em: <www.postgresql.org>. Acesso em: 13/04/2005

PRESSMAN, R. S., *Engenharia de Software*, 5. ed,. Rio de Janeiro: McGraw-Hill , 2002.

SQLAPI, *SQLAPI++ Library*, Disponível em: <http://www.sqlapi.com/>. Acesso em 23/05/2005.

THE POSTGRESQL GLOBAL DEVELOPMENT GROUP, *PostgreSQL 7.2 Reference Manual, 2001*

TRAVASSOS, G. H., *Notas de Aula do Curso Engenharia de Software Orientada a Objetos*, Rio de Janeiro:DEL - EE/UFRJ. , 2004.

VILLAS-BOAS, S. B., *The General Purpose Villas-Boas Library in C++,* Disponível em: <http://www.sbvb.com.br/>. Acesso em: 12/08/2005.

Apêndice A – Abreviaturas e Siglas

C++	Linguagem de Programação Orientada a Objetos
DLL	Dynamic Link Library
JDBC	Java DataBase Connectivity
ODBC	Open DataBase Connectivity
OO	Orientado a Objetos
SO	Shared Object
SGBD	Sistema de Gerenciamento de Banco de Dados
SQL	Structured Query Language
SQLAPI++	Biblioteca multiplataforma para conexão em múltiplos bancos

Apêndice B – Documento de Requisitos

DOCUMENTO DE REQUISITOS DA ARQUITETURA DO SQL TRANSLATOR

1. **Objetivo**
   Desenho e definição da arquitetura hibrida do SQL Translator.Esta arquitetura será usada no trabalho de desenvolvimento com objetivo de isolar camadas e para facilitar a manutenibilidade dos projetos.

2. **Requisitos de Dados**
   Requisito de Dados 1
   Uma query deve possuir query Id, Tipo do banco de dados, Texto SQL, Tipo de Query.
   - **Descrição de dados:**

Query Id	Inteiro positivo
Tipo do BD	Tipos pré-definidos em uma lista
Texto SQL	String
Tipo de Query	Assume os valores: Saída, Entrada

   Requisito de Dados 2
   Um tradutor deve possuir Tipo de banco de Dados para qual ele traduz. Assume como linguagem de origem Postgree.
   - **Descrição de dados:**

Tipo do BD	Tipos pré-definidos em uma lista

   Requisito de Dados 3
   Uma Conexão deve possuir o nome do usuário, senha, nome do banco de dados e tipo de banco de dados.
   - **Descrição de dados:**

Usuário	String
Password	String
DB Name	String
DB Type	String

3. **Requisitos Funcionais**

   Requisito Funcional 1:
   Escolher entre utilizar ou não o tradutor
   O SQL Translator deve fornecer ao desenvolvedor a opção de habilitar ou não o tradutor

   Requisito Funcional 2

Carregar biblioteca de tradutor
O SQL Translator deve realizar a carga do tradutor apenas quando for necessário seu uso durante a execução do programa. Tal medida tem como objetivo aumentar o desempenho durante o inicio do programa.

Requisito Funcional 3
Conectar ao SGBD
O SQL Translator deve realizar a conexão com o SGBD alvo quando for executar a query. A query pode ser traduzida, não-traduzida ou carregada por biblioteca dinâmica.

Requisito Funcional 4
Associar query a conexão
O SQL Translator deve associar uma query a uma conexão. Após realizada esta associação a query só pode ser executada no SGBD a ela associada.

Requisito Funcional 5
Traduzir query
O SQL Translator deve realizar a tradução da query que esteja escrita no SQL Padrão para o SQL destino. Para isso é necessário que haja um tradutor para este linguagem

Requisito Funcional 6
Carregar query sem recompilação
O SQL deve permitir a carga de query para sistemas já desenvolvidos. A carga deve ser sta através Translator deve verificar no diretório padrão, e qualquer outro diretório especificado pelo desenvolvedor, a existência de bibliotecas dinâmicas que contenham query. Caso existam deve carrega-las.

Requisito Funcional 7
Carregar query através de string
O SQL Translator deve ser capaz de carregar query através de string. A string com a query deve ser fornecida pelo desenvolvedor em tempo de execução.

Requisito Funcional 8
Executar query
O SQL Translator deve ser capaz de executar a query no SGBD requisitado. Toda a parte de conexão ao SGBD deve ser encapsulada pelo SQL Translator.

Requisito Funcional 9
Abrir query
O SQL Translator deve permitir a abertura de query.

Requisito Funcional 10
Executar Fetch na query
O SQL Translator deve permitir a execução de Fetch na query

Requisito Funcional 11
Recuperar parâmetro de saída
O SQL Translator deve permitir a recuperação dos parâmetros de saída

Requisito Funcional 12

Utilização de um diretório padrão

O SQL Translator deve utilizar um diretório padrão para o armazenamento dos tradutores e das queries

Requisito Funcional 13

Identificação de tradutores

O SQL Translator deve,ao iniciar a execução, percorrer o diretório padrão, e qualquer outro diretório especificado pelo desenvolvedor, identificando novos tradutores.

4. Restrições

Restrição 1

O SQL Translator deve ser implementado utilizando-se C++.

Restrição 2

O SQL Translator deve utilizar a SQLAPI++ para realizar a conexão com os SGBDs.

Restrição 3

O SQL Translator deve utilizar as funções da VBLib para manipulação de string , arquivos e bibliotecas dinâmica.

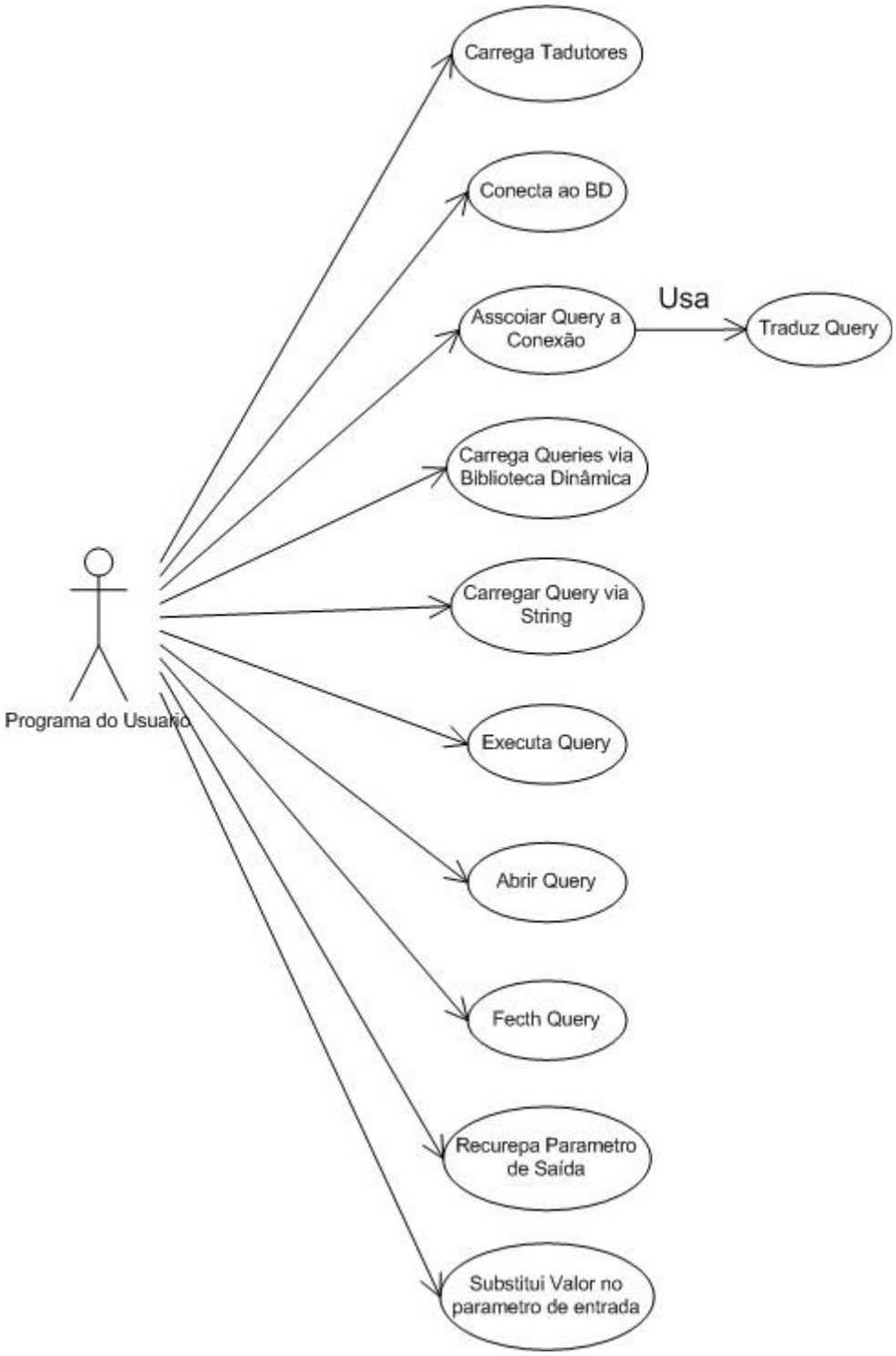
Restrição 4

O SQL padrão deve ser o do PostgreSQL.

Restrição 5

O primeiro tradutor a ser implementado deve ser o para o SGBD Oracle.

Apêndice C – Diagramas de Casos de Uso



Carrega Tradutores

Pré-condições:Nenhuma

Invariante: Nenhuma

Pós-condições: Nenhuma, um ou mais tradutores carregados

1. Verificar se há tradutor a ser carregado
    a. Não havendo não carregar nenhum tradutor
2. Carregar tradutores disponíveis

Conecta ao BD

Pré-condições: Existir um BD para se comunicar; Dados de conexão devem estar corretos; Ambiente deve estar configurado para o BD em questão

Invariante: Nenhuma

Pós-condições: Conectado ao BD

1. Verifica se a máquina do BD está acessível no canal informado
    a. Caso não esteja retornar mensagem de erro
2. Verifica se o BD está acessível
    a. Caso não esteja retornar mensagem de erro
3. Tenta conectar com os dados fornecidos
    a. Caso não seja possível retornar mensagem de erro
4. Testar Conexão

Associar Query a Conexão

Pré-condições: Existir o Handle de conexão; Existir a Query

Invariante: Nenhuma

Pós-condições: Query associada a conexão e pronta para ser executada ou aberta

1. Verifica se existe a conexão
    a. Se não existir retornar uma mensagem de erro
2. Verifica se existe a query a ser associado
    a. Se não existir retornar uma mensagem de erro
3. Chama a Traduz a Query caso necessário traduzir
    a. Não sendo possível a tradução retorna a mensagem de erro

4. Associar a Query a conexão

Traduz Query

Pré-condições: Existir o tradutor para o BD em questão

Invariante: Nenhuma

Pós-condições: Query traduzida

1. Interpreta o SQL de entrada
    a. Caso a query possua algum erro, gera uma mensagem de erro para o programa
2. Gera SQL de saída, para o outro banco

Carrega Queries via Biblioteca Dinâmica

Pré-condições: Nenhum

Invariante: Nenhuma

Pós-condições: Query reconhecida e carregada em memória

1. Verifica se há alguma DLL no diretório especificado
    a. Caso não haja, Nenhuma query é carregada
2. Começa a ver todas as DLL uma a uma
    a. Caso alguma DLL não esteja no padrão,esta DLL será ignorada
3. Para cada DLL que está no padrão, uma Query é carregada

Carrega Queries via String

Pré-condições: Nenhum

Invariante: Nenhuma

Pós-condições: Query reconhecida e carregada em memória

1. Verifica Parâmetros
2. Carrega a Query em memória

Executa Query

Pré-condições: Nenhum

Invariante: Nenhuma

Pós-condições: Numero de linhas afetadas

1. Verifica se a Query esta pronta para executar

    a. Caso o usuário não tenha comprido todos os passos necessários como por exemplo associar a query a uma conexão. O programa deve gerar um mensagem de erro

2. Executa a Query
    a. Caso ocorra algum erro neste ponto, o programa deve mostrar para o usuário o erro do banco. Por exemplo: nome da tabela invalida.
3. Retorna para o usuário o numero de linha afetada

Abrir Query

Pré-condições: Nenhum

Invariante: Nenhuma

Pós-condições: Query pronta para fetching

1) Verifica se a Query esta pronta para executar
    a) Caso o usuário não tenha comprido todos os passos necessários como por exemplo associar a query a uma conexão. O programa deve gerar um mensagem de erro
2) Abre a Query
    a. Caso ocorra algum erro neste ponto, o programa deve mostrar para o usuário o erro do banco. Por exemplo: nome da tabela invalida.

Fetch Query

Pré-condições: Nenhuma

Invariante: Nenhuma

Pós-condições: Parâmetros de saída carregados em memória

1) Verifica se a Query esta aberta
    a) Caso contrario mostra o erro para o usuário
2) Carrega a saída em memória

Recupera um parâmetro de saída da Query

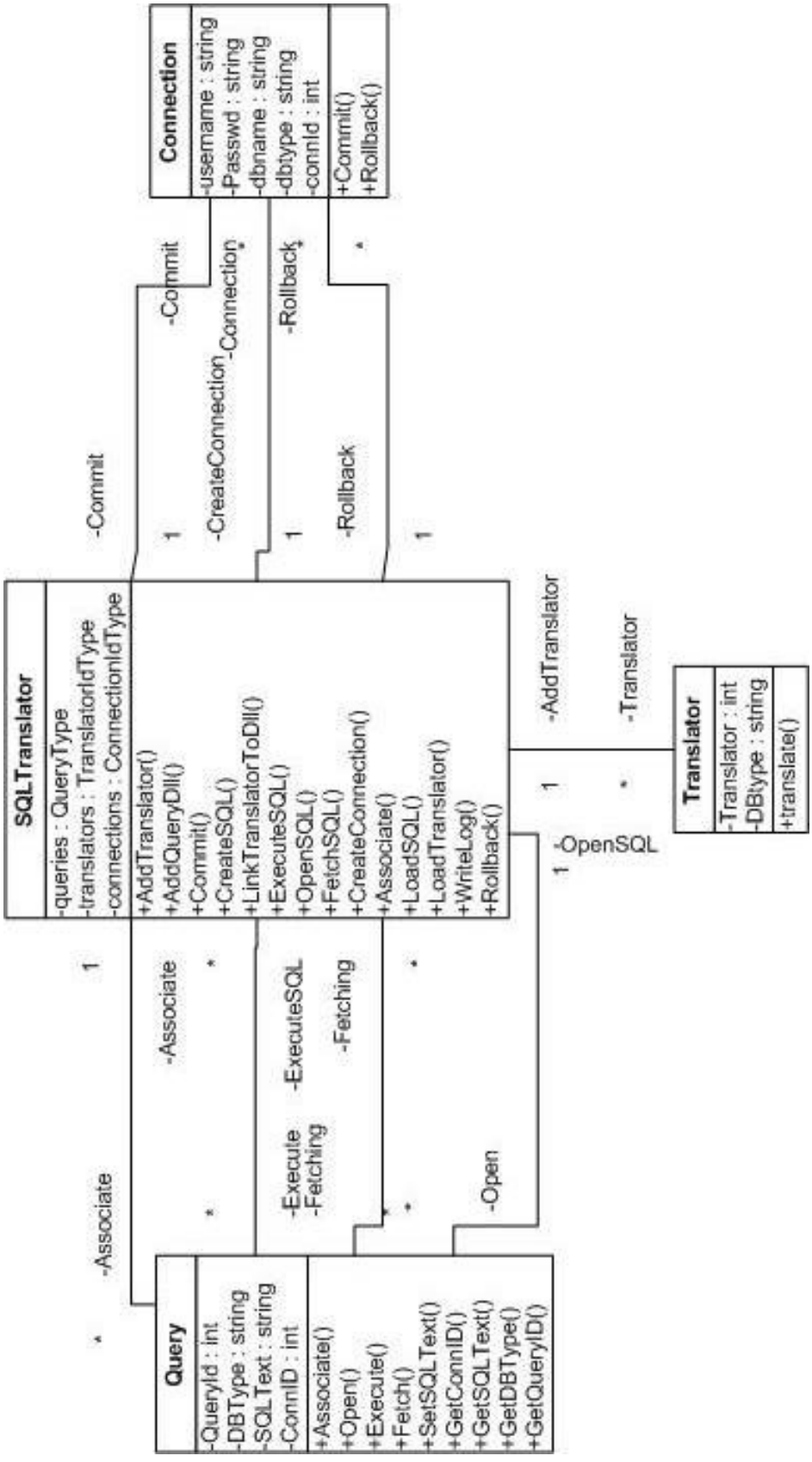
Pré-condições: Nenhuma

Invariante: Nenhuma

Pós-condições: Valor do parâmetro recuperado

1) Verifica se teve um fetching para esta query
    a) Caso contrario mostra o erro para o usuário
2) Verifica se o parâmetros requisitado, é saída da query
    a) Caso contrario mostra o erro para o usuário
3) Recupera o valor do parâmetro

Apêndice D – Diagramas de Classe



Apêndice E – Exemplo

Abaixo podemos ver a saída de um programa que utiliza o SQL Translator. Nele é realizado a carga de queries via DLL, carga de tradutores, conexão ao Banco de Dados e tradução de queries.

```
************************************************************
Carregando DLLs
Files:

----------


C:/Arquivos de programas/Microsoft Visual Studio/MyProjects/SQLtranslatorExample/SQLTranslatorExample/Query//Query.dll
C:/Arquivos de programas/Microsoft Visual Studio/MyProjects/SQLtranslatorExample/SQLTranslatorExample/Query//Query2.dll
Files:

----------


C:/Arquivos de programas/Microsoft Visual Studio/MyProjects/SQLtranslatorExample/SQLTranslatorExample/Tradutores//2.dll
C:/Arquivos de programas/Microsoft Visual Studio/MyProjects/SQLtranslatorExample/SQLTranslatorExample/Tradutores//oracle.dll
Tradutor jß existente, Serß sobre escrito
************************************************************

************************************************************
Criando Querys
************************************************************

************************************************************
Criando conexoes
Conectando com o banco [teste],com o usuario [teste], 10
************************************************************

************************************************************
Associando conexoes
************************************************************

************************************************************
Executando Querys
Executando [Insert into teste (name,str,num) values ('Alo2','STR',1)]
Realizando o Rolback
```

```
************************************************************

************************************************************
Setando Parametros
Parametro setado
************************************************************

************************************************************
Abrindo Query
Abrindo Query [Select teste.name as alo from teste where str = :1 and num = :2]
************************************************************

************************************************************
Fetching Query
Fetching Query [Select teste.name as alo from teste where str = :1 and num = :2]
************************************************************

************************************************************
Recuperando Valores
Paulo
Fetching Query [Select teste.name as alo from teste where str = :1 and num = :2]
************************************************************

************************************************************
Traduþ§es
************************************************************
Conectando com o banco [teste],com o usuario [scott], 2
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Caso com @ -5
De:
 SElect @ -5 a from tabela
Para:
 SELECT abs(-5) A FROM TABELA
Abrindo Query [ SELECT abs(-5) A FROM TABELA ]
Fetching Query [ SELECT abs(-5) A FROM TABELA ]
5
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Caso com nextval('text')
De:
 SELECT nextval('text') from tabela
Para:
 SELECT text.nextval FROM TABELA
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Caso com curval('seq')
De:
 SELECT curval('seq') from tabela
Para:
```

SELECT seq.curval FROM TABELA

++

++

Caso com Select sem from

De:

SELECT curval('seq')

Para:

SELECT seq.curval FROM DUAL

++

++

Caso usando a function DBLINK

De:

SELECT * FROM dblink('dbname=mydb', 'select proname, prosrc, curval("SEQ") from pg_proc') AS t1(proname nam

e, prosrc text) WHERE t1.name LIKE 'bytea%'

Para:

SELECT * FROM (SELECT PRONAME NAME, PROSRC TEXT, SEQ.curval FROM PG_PROC@mydb PG_PROC)

T1 WHERE T1.NAME LI

KE 'bytea%'

++

++

Caso com a clausula LIMIT

De:

SELECT * FROM table LIMIT 10

Para:

SELECT * FROM TABLE WHERE rownum < 10

++

++

Casos com Usando OUTER JOIN De:

SELECT * FROM table LIMIT 10

Para:

SELECT * FROM TABLE WHERE rownum < 10

De:

select * from a teste left outer join b teste3 on teste.name <> teste3.name and teste3.str = teste.str, t

este2

Para:

SELECT * FROM A TESTE , B TESTE3, TESTE2 WHERE TESTE.NAME (+) <>TESTE3.NAME AND TESTE.STR

(+) =TESTE3.STR

De:

SELECT * FROM table LIMIT 10

Para:

SELECT * FROM TABLE WHERE rownum < 10

Query Jß existente

De:

select abc from tabela

Para:

```
 SELECT ABC FROM TABELA
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Caso com Claussula EXCEPT

De:

```
 SElect coluna from tabela EXCEPT select coluna2 from tabela2
```

Para:

```
 SELECT COLUNA FROM TABELA MINUS SELECT COLUNA2 FROM TABELA2
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Caso com Claussulas ISNULL e NOTNULL

De:

```
 SElect coluna from tabela where coluna3 isnull EXCEPT select coluna2 from tabela2 where coluna4 is notnull
```

Para:

```
 SELECT COLUNA FROM TABELA WHERE COLUNA3 IS NULL MINUS SELECT COLUNA2 FROM TABELA2
WHERE COLUNA4 IS NOT NULL
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Caso com Funcoes NOW() e TIMEOFDAY()

De:

```
 SElect now()
```

Para:

```
 SELECT (select sysdate from dual)  FROM DUAL
----------------------------------------------------------------
```

De:

```
 select timeofday()
```

Para:

```
 SELECT (select to_char(sysdate,'DY MON DD HH24::MI:SS.000000 YYYY TZ') from dual)  FROM DUAL
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Caso com comentarios

De:

```
  Select /* Carro */3 * 4 from a , Bn where a.col = ' ' and Bn.col2 like ' hjhjhj' Bn --Esquece
 /* To nem ai*/
```

Para:

```
 SELECT 3 * 4 FROM A , BN WHERE A.COL = ' ' AND BN.COL2 LIKE ' hjhjhj' BN
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Caso com divesas substrings

De:

```
  Select 3 * 4, (select (select (select time),alo),alo, nextval('Paulo')), timeofday(), @ -5, timeofday( ), @6
, nextval('kiko') from         ' ' a ' hjhjhj' Bn  limit 10 except select * from   a   teste right outer join
b teste3 on teste.name <> teste3.name and teste3.str = teste.str, teste2
```

Para:

```
 SELECT 3 * 4, ( SELECT ( SELECT ( SELECT TIME FROM DUAL ) , ALO FROM DUAL ) , ALO, Paulo.nextval
```

FROM DUAL)

, (select to_char(sysdate,'DY MON DD HH24::MI:SS.000000 YYYY TZ') from dual) , abs(-5), (select to_char(sysda

te,'DY MON DD HH24::MI:SS.000000 YYYY TZ') from dual) , abs(6), kiko.nextval FROM ' ' A ' hjhjhj' BN WHERE row

num < 10 MINUS SELECT * FROM A TESTE , B TESTE3, TESTE2 WHERE TESTE.NAME <>TESTE3.NAME (+) AND TESTE.STR=TEST

E3.STR (+)

++

++

Caso com ''

De:

SELECT * FROM dblink('dbname=mydb', 'select proname, prosrc, curval("SEQ") from pg_proc, table where pg_pro

c.id = table.id ') AS t1(proname name, prosrc text) WHERE t1.name LIKE 'bytea%'

Para:

SELECT * FROM (SELECT PRONAME NAME, PROSRC TEXT, SEQ.curval FROM PG_PROC@mydb PG_PROC, TABLE@mydb TABLE WHE

RE PG_PROC.ID = TABLE.ID) T1 WHERE T1.NAME LIKE 'bytea%'

++

****

Finalizando

Apagando a Query: DBtype - [2] , QueryId - [1] e SQLText [SELECT abs(-5) A FROM TABELA]

Apagando a Query: DBtype - [2] , QueryId - [2] e SQLText [SELECT text.nextval FROM TABELA]

Apagando a Query: DBtype - [2] , QueryId - [3] e SQLText [SELECT seq.curval FROM TABELA]

Apagando a Query: DBtype - [2] , QueryId - [4] e SQLText [SELECT seq.curval FROM DUAL]

Apagando a Query: DBtype - [2] , QueryId - [5] e SQLText [SELECT * FROM (SELECT PRONAME NAME, PROSRC TEXT,

SEQ.curval FROM PG_PROC@mydb PG_PROC) T1 WHERE T1.NAME LIKE 'bytea%']

Apagando a Query: DBtype - [2] , QueryId - [6] e SQLText [SELECT * FROM TABLE WHERE rownum < 10]

Apagando a Query: DBtype - [2] , QueryId - [7] e SQLText [SELECT * FROM TABLE WHERE rownum < 10]

Apagando a Query: DBtype - [2] , QueryId - [8] e SQLText [SELECT * FROM A TESTE , B TESTE3, TESTE2 WHERE TES

TE.NAME (+) <>TESTE3.NAME AND TESTE.STR (+) =TESTE3.STR]

Apagando a Query: DBtype - [2] , QueryId - [9] e SQLText [SELECT * FROM TABLE WHERE rownum < 10]

Apagando a Query: DBtype - [2] , QueryId - [10] e SQLText [SELECT ABC FROM TABELA]

Apagando a Query: DBtype - [2] , QueryId - [11] e SQLText [SELECT COLUNA FROM TABELA MINUS SELECT COLUNA2 FRO

M TABELA2]

Apagando a Query: DBtype - [2] , QueryId - [12] e SQLText [SELECT COLUNA FROM TABELA WHERE COLUNA3 IS NULL MI

NUS SELECT COLUNA2 FROM TABELA2 WHERE COLUNA4 IS NOT NULL]

Apagando a Query: DBtype - [2] , QueryId - [13] e SQLText [SELECT (select sysdate from dual) FROM DUAL]

Apagando a Query: DBtype - [2] , QueryId - [14] e SQLText [SELECT (select to_char(sysdate,'DY MON DD HH24::MI

:SS.000000 YYYY TZ') from dual) FROM DUAL]

Apagando a Query: DBtype - [2] , QueryId - [15] e SQLText [SELECT 3 * 4 FROM A , BN WHERE A.COL = ' ' AND BN.

COL2 LIKE ' hjhjhj' BN]

Apagando a Query: DBtype - [2] , QueryId - [16] e SQLText [SELECT 3 * 4, (SELECT (SELECT (SELECT TIME FROM

DUAL) , ALO FROM DUAL) , ALO, Paulo.nextval FROM DUAL) , (select to_char(sysdate,'DY MON DD

```
HH24::MI:SS.0
        00000 YYYY TZ') from dual) , abs(-5), (select to_char(sysdate,'DY MON DD HH24::MI:SS.000000 YYYY TZ') from
dua
        l) , abs(6), kiko.nextval FROM ' ' A ' hjhjhj' BN WHERE rownum < 10 MINUS SELECT * FROM A TESTE , B TESTE3,
TE
        STE2 WHERE TESTE.NAME <>TESTE3.NAME (+)  AND TESTE.STR=TESTE3.STR  (+)  ]
        Apagando a Query: DBtype - [2] , QueryId - [17] e SQLText [ SELECT * FROM (   SELECT PRONAME NAME,
PROSRC TEXT,
         SEQ.curval FROM PG_PROC@mydb PG_PROC, TABLE@mydb TABLE WHERE PG_PROC.ID = TABLE.ID   ) T1
WHERE T1.NAME LIKE '
        bytea%']
        Apagando a Query: DBtype - [10] , QueryId - [1] e SQLText [ SElect @ -5 a from tabela ]
        Apagando a Query: DBtype - [10] , QueryId - [2] e SQLText [ SELECT nextval('text') from tabela ]
        Apagando a Query: DBtype - [10] , QueryId - [3] e SQLText [ SELECT curval('seq') from tabela ]
        Apagando a Query: DBtype - [10] , QueryId - [4] e SQLText [ SELECT curval('seq') ]
        Apagando a Query: DBtype - [10] , QueryId - [5] e SQLText [ SELECT * FROM dblink('dbname=mydb', 'select pronam
        e, prosrc, curval("SEQ") from pg_proc') AS t1(proname name, prosrc text) WHERE t1.name LIKE 'bytea%']
        Apagando a Query: DBtype - [10] , QueryId - [6] e SQLText [ SELECT * FROM table LIMIT 10 ]
        Apagando a Query: DBtype - [10] , QueryId - [7] e SQLText [ SELECT * FROM table LIMIT 10 ]
        Apagando a Query: DBtype - [10] , QueryId - [8] e SQLText [ select * from   a   teste left outer join b teste3
         on teste.name <> teste3.name and teste3.str = teste.str, teste2 ]
        Apagando a Query: DBtype - [10] , QueryId - [9] e SQLText [ SELECT * FROM table LIMIT 10 ]
        Apagando a Query: DBtype - [10] , QueryId - [10] e SQLText [select abc from tabela]
        Apagando a Query: DBtype - [10] , QueryId - [11] e SQLText [ SElect coluna from tabela EXCEPT select coluna2 f
        rom tabela2]
        Apagando a Query: DBtype - [10] , QueryId - [12] e SQLText [ SElect coluna from tabela where coluna3 isnull EX
        CEPT select coluna2 from tabela2 where coluna4 is notnull]
        Apagando a Query: DBtype - [10] , QueryId - [13] e SQLText [ SElect now() ]
        Apagando a Query: DBtype - [10] , QueryId - [14] e SQLText [ select timeofday() ]
        Apagando a Query: DBtype - [10] , QueryId - [15] e SQLText [  Select /* Carro */3 * 4 from a , Bn where a.col
        = ' ' and Bn.col2 like ' hjhjhj' Bn --Esquece
         /* To nem ai*/]
        Apagando a Query: DBtype - [10] , QueryId - [16] e SQLText [  Select 3 * 4, (select (select (select time),alo)
        ,alo, nextval('Paulo')), timeofday(), @ -5, timeofday( ), @6, nextval('kiko') from         ' ' a ' hjhjhj' Bn
         limit 10 except select * from   a   teste right outer join b teste3 on teste.name <> teste3.name and teste3.s
        tr = teste.str, teste2]
        Apagando a Query: DBtype - [10] , QueryId - [17] e SQLText [ SELECT * FROM dblink('dbname=mydb', 'select prona
        me, prosrc, curval("SEQ") from pg_proc, table where pg_proc.id = table.id ') AS t1(proname name, prosrc text
        ) WHERE t1.name LIKE 'bytea%']
        Apagando a Query: DBtype - [10] , QueryId - [66] e SQLText [Insert into teste (name,str,num) values ('Alo2','S
        TR',1)]
        Apagando a Query: DBtype - [10] , QueryId - [98] e SQLText [Select teste.name as alo from teste where str = :1
         and num = :2]
        Apagando a Query: DBtype - [10] , QueryId - [99] e SQLText [Select teste.name as alo from teste]
        Apagando a Query: DBtype - [16] , QueryId - [11] e SQLText [select abc from tabela]
        Apagando o Tradutor: DBtype - [2] e DLL [C:/Arquivos de programas/Microsoft Visual Studio/MyProjects/SQLtransl
        atorExample/SQLTranslatorExample/Tradutores//oracle.dll]
        Realizando o commit
        Apagando a Connexao : DBtype - [10] e ConnId [1]
        Realizando o commit
        Apagando a Connexao : DBtype - [2] e ConnId [2]
```

Apêndice F – Manual de Utilização do SQL Translator

Abaixo descreverem,os alguns exemplos de utilização do SQL Translator.

No primeiro exemplo estaremos mostrando um software que realiza um SELECT de todas as linhas de uma tabela e imprime na tela.

Exemplo1.cpp

```
#include “SQLtranslator.h”
#include <iostream>
using namespace std;

int main ()
{
    SQLTranslator myApp;
    myApp.CreateSQL(1,“select nome from clientes”);  //Cria a query com o seu respective Query ID
     myApp.CreateConnection(3, “usuário”, “senha”, “Nome do Banco”); /*Cria uma conexão com seu respectivo ID no PostgreSQL*/
    myApp.associate(1,3);
    myApp.OpenSQL(1) //Query ID
    While (myApp.FetchSQL(1))
    {
        cout<<myApp.GetOutputParameter(1,”nome “).asString()<<endl;
    }
}
```

Agora faremos a mesma operação do exemplo acima mas no banco de dados Oracle e carregando a query via biblioteca dinâmica.

Exemplo2.cpp

```
#include “SQLtranslator.h”
#include <iostream>
using namespace std;

int main ()
{
    SQLTranslator myApp;
     myApp.LoadSQL(“path da biblioteca dinâmica”);  /*A biblioteca deste diretório possui ID 2, é do tipo Oracle e possui o seguinte texto:” SELECT nome, salario FROM funcionários”*/
    myApp.CreateConnection(3, “usuário”, “senha”, “Nome do Banco”, SA_Oracle_Client); /*Cria uma conexão com seu respectivo ID no Oracle*/
    myApp.associate(2,3);
    myApp.OpenSQL(2,SA_Oracle_Client) //Query ID
    While (myApp.FetchSQL(2,SA_Oracle_Client))
    {
                cout<<myApp.GetOutputParameter(1,”nome  “,SA_Oracle_Client).asString() << “-“ << myApp.GetOutputParameter(1,”salario “,SA_Oracle_Client).asString()  <<endl;
    }
```

```
}
```

Agora mostraremos um exemplo de uma query nativa de PostgreSQL sendo executada em Oracle com o auxilio do tradutor.

Example3.cpp

```
#include “SQLtranslator.h”
#include <iostream>
using namespace std;

int main ()
{
    SQLTranslator myApp;
     myApp.LoadSQL(“path da biblioteca dinâmica”);  /*A biblioteca deste diretório possui ID 2, é do tipo PostgreSQL e possui o seguinte texto:” SELECT origem, destino,@distancia deslocamento FROM cidades”*/
    myApp.LoadTranslator(“path da biblioteca do tradutor Oracle”);
     myApp.CreateConnection(3, “usuário”, “senha”, “Nome do Banco”, SA_Oracle_Client); /*Cria uma conexão com seu respectivo ID no Oracle*/
    myApp.associate(2,3); //Neste momento será realizada a tradução já queo tipo da conexão é diferente do tipo da query.
    myApp.OpenSQL(2,SA_Oracle_Client) //Query ID
    While (myApp.FetchSQL(2,SA_Oracle_Client))
    {
                cout<<myApp.GetOutputParameter(1,”origem“,SA_Oracle_Client).asString() << “-“ << myApp.GetOutputParameter(1,”destino “,SA_Oracle_Client).asString() << “= “<<myApp.GetOutputParameter(1,”deslocamento “,SA_Oracle_Client).asLong()<<endl;
    }
}
```

Apêndice G – Manual de Criação de Biblioteca de Query

O presente manual tem como objetivo explicar o procedimento de criação de uma biblioteca dinâmica de query para o SQL Translator.

Para gerar uma biblioteca dinâmica no padrão Windows (DLL) deve-se preencher os arquivos abaixo. Deve ser lembrado que o nome é de livre escolha do desenvolvedor.

Query.cpp

```
typedef
enum
{
        SA_Client_NotSpecified,
        SA_ODBC_Client,
        SA_Oracle_Client,
        SA_SQLServer_Client,
        SA_InterBase_Client,
        SA_SQLBase_Client,
        SA_DB2_Client,
        SA_Informix_Client,
        SA_Sybase_Client,
        SA_MySQL_Client,
        SA_PostgreSQL_Client,
        _SA_Client_Reserverd = (int)(((unsigned int)(-1))/2)
} SAClient_t;

#define QUERY_TEXT "select abc from tabela" // Texto da query a ser preenchido pelo usuário
#define QUERY_ID 10                         // Query ID de livre escolha do usuário
#define QUERY_DB_TYPE SA_PostgreSQL_Client      /*Refere-se ao tipo de banco nativo da query,
preendhido conforme o enumerator acima */

unsigned int QueryID()
{
  return QUERY_ID;
}

unsigned int DBtype()
{
  return QUERY_DB_TYPE;
}

const char * SQL_TEXT()
{
   static char retorno[] = QUERY_TEXT;
        return retorno;
}
```

O arquivo def deve ser usado conforme abaixo. Não é necessária nenhuma

modificação.

Query.def

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; query.def : Declares the module parameters for the DLL.
LIBRARY "QUERY"
DESCRIPTION 'QUERY Windows Dynamic Link Library'
EXPORTS
; Explicit exports can go here
QueryID @1
DBtype @2
SQL_TEXT @3
```

Para gerar uma biblioteca dinâmica no padrão Linux (SO) deve-se preencher o arquivo abaixo.

```
typedef
enum
{
        SA_Client_NotSpecified,
        SA_ODBC_Client,
        SA_Oracle_Client,
        SA_SQLServer_Client,
        SA_InterBase_Client,
        SA_SQLBase_Client,
        SA_DB2_Client,
        SA_Informix_Client,
        SA_Sybase_Client,
        SA_MySQL_Client,
        SA_PostgreSQL_Client,
        _SA_Client_Reserverd = (int)(((unsigned int)(-1))/2)
} SAClient_t;

#define QUERY_TEXT "select abc from tabela" // Texto da query a ser preenchido pelo usuário
#define QUERY_ID 10                         // Query ID de livre escolha do usuário
#define QUERY_DB_TYPE SA_PostgreSQL_Client      /*Refere-se ao tipo de banco nativo da query,
preendhido conforme o enumerator acima */

extern "C" {
unsigned int QueryID()
{
  return QUERY_ID;
}
unsigned int DBtype()
{
  return QUERY_DB_TYPE;
}

const char * SQL_TEXT()
{
   static char retorno[] = QUERY_TEXT;
        return retorno;
}
```

```
}
```

Apêndice H – Manual de Criação de Tradutor

O presente manual tem como objetivo explicar o procedimento de criação de uma biblioteca dinâmica de tradutor para o SQL Translator.

Para gerar uma biblioteca dinâmica no padrão Windows (DLL) deve-se preencher os arquivos abaixo. Deve ser lembrado que o nome é de livre escolha do desenvolvedor.

Translator.cpp

```
typedef
enum
{
        SA_Client_NotSpecified,
        SA_ODBC_Client,
        SA_Oracle_Client,
        SA_SQLServer_Client,
        SA_InterBase_Client,
        SA_SQLBase_Client,
        SA_DB2_Client,
        SA_Informix_Client,
        SA_Sybase_Client,
        SA_MySQL_Client,
        SA_PostgreSQL_Client,
        _SA_Client_Reserverd = (int)(((unsigned int)(-1))/2)
} SAClient_t;

#define DB_TYPE SA_Oracle_Client        /*Refere-se ao tipo de banco alvo da tradução, preenchido conforme o enumerator acima */

unsigned int DBtype()
{
        return DB_TYPE;

}

const char * Translate (const char* Texto)
{
//Acresecente aqui o código de tradução
}
```

O arquivo def deve ser usado conforme abaixo. Não é necessária nenhuma modificação.

Query.def

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 16.def : Declares the module parameters for the DLL.
LIBRARY "16"
DESCRIPTION '16 Windows Dynamic Link Library'
EXPORTS
```

```
; Explicit exports can go here
DBtype @1
       Translate @2
```

Para gerar uma biblioteca dinâmica no padrão Linux (SO) deve-se preencher o arquivo abaixo.

```
typedef
enum
{
        SA_Client_NotSpecified,
        SA_ODBC_Client,
        SA_Oracle_Client,
        SA_SQLServer_Client,
        SA_InterBase_Client,
        SA_SQLBase_Client,
        SA_DB2_Client,
        SA_Informix_Client,
        SA_Sybase_Client,
        SA_MySQL_Client,
        SA_PostgreSQL_Client,
        _SA_Client_Reserverd = (int)(((unsigned int)(-1))/2)
} SAClient_t;

#define DB_TYPE SA_PostgreSQL_Client        /*Refere-se ao tipo de banco alvo da tradução, preenchido conforme o enumerator acima */

extern "C"
{

        unsigned int DBtype()
        {
                return DB_TYPE;
        }

        const char * Translate (const char* Texto)
        {
            //Acrescente aqui o código de tradução
        }


}
```

Apêndice I – Diagrama de atividades do Tradutor Oracle