UNIVERSIDADE FEDERAL DO PIAUÍ
CAMPUS SENADOR HELVIDIO NUNES DE BARROS
CURSO DE BACHARELADO EM SISTEMAS DE INFORMAÇÃO

Leonílio Rodrigues de Sousa

# Protótipo de uma Ferramenta de conversão de consultas SQL para Álgebra Relacional

PICOS – PI
2014

Leonílio Rodrigues de Sousa

# Protótipo de uma Ferramenta de conversão de consultas SQL para Álgebra Relacional

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação Campus Senador Helvídio Nunes de Barros da Universidade Federal do Piauí como parte dos requisitos para obtenção do Grau de Bacharel em Sistemas de Informação, sob orientação da Profª. MSc. Patricia Medyna Lauritzen de Lucena Drumond.

PICOS – PI

2014

Eu, **Leonílio Rodrigues de Sousa,** abaixo identificado(a) como autor(a), autorizo a biblioteca da Universidade Federal do Piauí a divulgar, gratuitamente, sem ressarcimento de direitos autorais, o texto integral da publicação abaixo discriminada, de minha autoria, em seu site, em formato PDF, para fins de leitura e/ou impressão, a partir da data de hoje.

Picos-PI, 07 de agosto de 2014.

Leonílio Rodrigues de Sousa
Assinatura


**FICHA CATALOGRÁFICA**
**Serviço de Processamento Técnico da Universidade Federal do Piauí**
**Biblioteca José Albano de Macêdo**

**S725p** Sousa, Leonílio Rodrigues de.
Protótipo de uma ferramenta para conversão de consultas SQL para álgebra relacional / Leonílio Rodrigues de Sousa. - 2014.
CD-ROM : il. ; 4 ¾ pol. (54 p.)

Monografia(Bacharelado em Sistemas de Informação) – Universidade Federal do Piauí. Picos-PI, 2014.
Orientador(A): Profa. MSc. Patrícia Medyna Lauritzen de L. Drumond

1. Interpretador de Comandos. 2. Expressão Algébrica. 3. Bancos de Dados Relacionais. 4. Parser. I. Título.

**CDD 005.75**

Leonílio Rodrigues de Sousa

# Protótipo de uma Ferramenta de conversão de consultas SQL para Álgebra Relacional

Trabalho de Conclusão de Curso apresentado ao Curso de Bacharelado em Sistemas de Informação Campus Senador Helvídio Nunes de Barros da Universidade Federal do Piauí como parte dos requisitos para obtenção do Grau de Bacharel em Sistemas de Informação, sob orientação da Profª. MSc. Patricia Medyna Lauritzen de Lucena Drumond.

Data de Aprovação:

05/08/2014

Prof. MSc. Patricia Medyna Lauritzen de Lucena Drumond _____ UFPI

Prof. Esp. Francisco das Chagas Imperes Filho _____ UFPI

Prof. MSc. Frank Cesar Lopes Veras _____ UFPI

PICOS – PI

2014

Dedico este trabalho a toda minha família em especial aos meus pais, Maria da Conceição Rodrigues de Sousa, Miguel Arcanjo de Sousa, Maria Júlia, Francisco do Rego Mendes, Márcia Valéria e aos meus irmãos por me ajudarem sempre.

Agradeço a Deus por ser o criador de todos as coisas, por ter compartilhado um pouco da sua inteligência, proporcionando assim o desenvolvimento humano, a busca por melhorias e a descoberta de novas tecnologias.

Agradeço à toda minha família por toda a ajuda proporcionada nessa jornada, aos amigos da turma da universidade, aos amigos do trabalho e à minha noiva e companheira: Márcia Váleria. Aos professores por sua dedicação e satisfação em compartilhar conhecimento. Desejo a todos muita saúde, paz e determinação.

Agradeço ao professor Arlindo Henrique Magalhães de Araújo por ensinar-me a base necessária para o desenvolvimento deste trabalho e por toda sua atuação como professor. Agradeço muito aos professores que aceitaram participar da banca de apresentação deste trabalho.

Amo a liberdade, por isso as coisas que amo deixo-as livres. Se voltarem é porque as conquistei. Se não voltarem é porque nunca as tive.

(Bob Marley)

Que os vossos esforços desafiem as impossibilidades, lembrai-vos de que as grandes coisas do homem foram conquistadas do que parecia impossível.

(Charles Chaplin)

O mundo é um lugar perigoso de se viver, não por causa daqueles que fazem o mal, mas sim por causa daqueles que observam e deixam o mal acontecer.

(Albert Einstein)

# RESUMO

Realizar a conversão de consultas complexas de forma manual pode acarretar em sucessivas falhas humanas além de um grande desperdício de tempo. Desta forma, surge a necessidade de realizar a conversão de consultas SQL para a linguagem álgebra relacional com o menor desperdício de tempo e melhor eficiência possível. Esse trabalho apresenta o desenvolvimento de uma ferramenta que realiza a conversão de consultas SQL para a linguagem álgebra relacional, mostrando a consulta convertida ao usuário no próprio programa. A conversão de consultas da *Structured Query Language* (SQL) para a linguagem da álgebra relacional é um processo em que o *script* da linguagem SQL é interpretado através de um analisador escrito em *Java* que faz a análise da expressão em SQL. Para identificar os comandos presentes na consulta o programa instancia métodos implementados no interpretador de comandos ZQL *Parser*, tomando como entrada declarações SQL tais como, *SELECT, INSERT, UPDATE, DELETE, COMMIT, ROLLBACK* e TRANSAÇÃO e preenche as estruturas *Java* que representam as declarações analisadas.O desenvolvimento do trabalho foi realizado utilizando a linguagem *Java* versão 1.7 em conjunto com o compilador *Netbeans* 7.3.1. Possibilitando retornar ao usuário expressões na linguagem álgebra relacional correspondentes às inseridas em SQL.

**Palavras-chave:** Interpretador de Comandos, Expressão Algébrica, Bancos de Dados Relacionais, *Parser*.

# ABSTRACT

Perform the conversion of complex queries manually may result in successive human error plus a big waste of time. Thus arises the need for converting SQL queries to relational algebra language with the least waste of time and optimum efficiency. This paper presents the development of a tool that performs the conversion of SQL queries to relational algebra language, showing the converted query the user in the program itself. The conversion of visits to Structured Query Language (SQL) into the language of relational algebra is a process in which the script of the SQL language is interpreted by a parser written in Java that makes the analysis of expression in SQL. To identify the query commands on the program instantiates methods implemented in shell ZQL Parser, taking as input SQL statements such as SELECT, INSERT, UPDATE, DELETE, COMMIT, ROLLBACK TRANSACTION and fills and Java structures that represent the statements analyzed . The development work was carried out using the Java language version 1.7 together with Netbeans 7.3.1 compiler. Allowing the user to return expressions in relational algebra corresponding to the inserted language in SQL.

**Keywords:** Command Interpreter, Algebraic expression, Relational Databases, Parser.

# Lista de Figuras

# Lista de Tabelas

# Lista de abreviaturas e siglas

SGBDR - Sistema Gerenciador de Banco de Dados Relacionais
IBM - International Business Machines
SQL - Structured Query Language
SDK - Software Development Kit
UML - Unified Modelling Language
CSDC - Ferramenta de Conversão de Script SQL
DDL - Definition Data Language
DML - Data Manipulation Language
XML - EXtensible Markup Language
SGBD - Sistema Gerenciador de Banco de Dados
MVS - Multiple Virtual Storage
RDB - Receive Deposite Bank
DB2 - Databse Software
TI - Tecnologia da Informação
ISO - International Organization for Standardization
CPF - Cadastro de Pessoas Físicas
ER - Entidade Relacionamento
SQL/DS- Structured Query Language/Data System
ANSI - American National Standards Institute
JDK - Java Development Kit
IDE - Integrated Development Environment
API - Application Programming Interface
GNU - General Public License
HD - Hard Disk
GHz - Giga Hertz
PHP - Hypertext Preprocessor
HTML - HyperText Markup Language
DHTML - Dynamic HyperText Markup Language

# Lista de símbolos

$\sigma$ –Sigma

$\cup$ –União

$\cap$ –Intersecção

= –Igual

< –Menor

$\leq$ –Menor ou Igual

> –Maior

$\geq$ –Maior ou Igual

$\neq$ –Diferente

$\pi$ –PI

$\rho$ –Renomear

$\times$ –Produto Cartesiano

$\bowtie$–Junção Natural

# Sumário

# 1 INTRODUÇÃO

A *Structured Query Language* (SQL) e álgebra relacional são linguagens para banco de dados relacionais utilizadas para realizar consultas. A SQL também permite realizar outras operações tais como criar, alterar e remover dados da base de dados. Essa linguagem é hoje um padrão na realização de consultas em bancos de dados em razão da sua simplicidade e facilidade de uso.

Consultas construídas em SQL podem tornar-se bastante complexas e combinadas com o alto nível da linguagem geralmente dificulta a assimilação dos estudantes. O uso da álgebra relacional pode facilitar o aprendizado fornecendo outras maneiras de resolução conforme evidenciado por Garcia (2001), "uma das vantagens de usar álgebra relacional é que ela torna fácil explorar formas alternativas de uma consulta".

A álgebra relacional não apresenta a simplicidade da linguagem SQL. Esta, por sua vez, é uma linguagem de consulta mais formal, ou seja, o usuário fornece informações ao sistema e o mesmo realiza uma sequência de operações no banco de dados. A SQL é representada por comandos tais como *"select"*, *"from"*, *"where"* etc, enquanto que a álgebra relacional é representada por símbolos, onde cada símbolo representa uma operação no modelo relacional, um conjunto de operações forma uma expressão algébrica que realiza uma consulta em um banco de dados relacional.

Conhecimento sobre álgebra relacional nos permite entender a execução e otimização de consultas em Sistemas Gerenciadores de Banco de Dados Relacionais (SGBDR), característica importante apontada por (SUMATHI, 2007).

Segundo Elmasri e Shamkant (2011), uma consulta expressa em uma linguagem de consulta de alto nível, como SQL, primeiro precisa ser lida, analisada e validada. A varredura identifica os *tokens* de consulta – como as palavras-chave SQL, nomes de atributo e nomes de relação – que aparecem no texto da consulta, enquanto o analisador sintático verifica a sintaxe da consulta para determinar se ela está formulada de acordo com as regras de sintaxe (regras de gramática) da linguagem de consulta.

A ferramenta de conversão faz a análise da consulta SQL e a converte para uma expressão em álgebra relacional, sendo um dos pontos mais importantes no desenvolvimento do software é a implementação do compilador que é responsável por fazer a comparação dos símbolos de uma linguagem para outra. O programa que faz a análise da consulta e a divide em operadores, comandos e atributos é o ZQL *Parser*.

Uma alternativa de pontencializar a utilização da álgebra relacional é a união dos con-

ceitos de álgebra relacional e SQL em uma única ferramenta. Esta ferramenta é útil para o estudo de consultas SQL utilizadas em SGBDs. Segundo Elmasri e Shamkant (2011), uma consulta SQL é primeiro traduzida para uma expressão equivalente da álgebra relacional estendida – representada como uma estrutura de dados de árvore de consulta – que é então, otimizada. Normalmente, as consultas SQL são decompostas em blocos de consulta, que formam as unidades básicas que podem ser traduzidas em operadores algébricos e otimizadas. Uma consulta pode conter uma única expressão *SELECT-FROM- WHERE*, bem como cláusulas *GROUPBY* e *HAVING*, se estas fizerem parte do bloco. As consultas aninhadas em uma consulta são identificadas como blocos de consulta separados. Como a SQL inclui operadores de agregação – como *MAX*, *MIN*, *SUM* e *COUNT* – esses operadores também precisam ser incluídos na álgebra estendida.

O presente trabalho está organizado da seguinte forma: No capitulo 2 é abordada a fundamentação teórica para o desenvolvimento da ferramenta, dentre elas, as principais características dos assuntos de banco de dados relacionais, da linguagem SQL, da linguagem álgebra relacional e do *parser* ZQL. No capitulo 3 são apresentados os trabalhos relacionados e as características e funcionalidades de cada um. O desenvolvimento da ferramenta e seu funcionamento são mostrados no capítulo 4. Os testes realizados e os resultados obtidos são apresentados no capítulo 5 e, por fim, as conclusões obtidas com a análise dos resultados encontram-se no capítulo 6.

# 2 BANCO DE DADOS

Este capítulo apresenta os conceitos básicos sobre os bancos de dados relacionais, as linguagens álgebra relacional e SQL. Além disso, mostra a ferramenta ZQL *parser* para a manipulação de comandos SQL utilizando a linguagem *Java*.

Para Date (2003), "Um banco de dados é uma coleção de dados persistentes, usada pelos sistemas de aplicação de uma determinada empresa". Isto significa que um banco de dados é um local onde são armazenados dados necessários à manutenção das atividades de determinada organização, sendo este repositório a fonte de dados para as aplicações atuais e as que vierem a existir. "A expressão Banco de Dados originou-se do termo inglês *Databanks*. Este foi trocado pela palavra *Databases* – Base de Dados – devido possuir significação mais apropriada" (SETZER; CORRÊA DA SILVA, 2005).

## 2.1 Conceitos Básicos de Bancos de Dados Relacionais

O modelo de dados relacional foi introduzido inicialmente por Ted Codd, da IBM *Research*, em 1970, em um artigo clássico Codd (1970), que atraiu atenção imediata devido a sua simplicidade e base matemática. O modelo usa o conceito de relação matemática – que se parece com uma tabela de valores – como seu bloco de montagem básico, e sua base teórica reside em uma teoria de conjunto e lógica de predicado de primeira ordem (ELMASRI; SHAMKANT, 2011).

As primeiras implementações comerciais do modelo relacional se tornaram disponíveis no início da década de 1980, como o sistema SQL/DS no sistema operacional MVS, da IBM, e o SGBD, da *Oracle*. Desde então, o modelo foi implementado em uma grande quantidade de sistemas comerciais. Os SGBDs Relacionais (SGBDRs) populares atuais incluem o DB2 e *Informix Dinamic Server* (da IBM), o *Oracle* e Rdb (da *Oracle*), o *Sybase* SGBD (da *Sybase*) e o *SQLServer* e *acess* (da *Microsoft*). Além disso vários sistemas de código aberto, como *MySQL* e *PostgreSQL* , estão disponíveis (ELMASRI; SHAMKANT, 2011).

O cálculo relacional é considerado a base para a linguagem SQL, e a álgebra relacional é usada nos detalhes internos de muitas implementações de banco de dados para processamento e otimização de consulta. O modelo relacional representa o banco de dados como uma coleção de relações. Informalmente, cada relação é semelhante a uma tabela de valores ou, até certo ponto, a um arquivo plano de registros. Ele é chamado de arquivo porque cada registro tem uma

simples estrutura linear e plana (ELMASRI; SHAMKANT, 2011).

O modelo relacional é muito simples e elegante: um banco de dados é uma coleção de uma ou mais relações, em que cada relação é uma tabela com linhas e colunas. Essa representação tabular simples permite que até usuários iniciantes entendam o conteúdo de um banco de dados e possibilita o uso de linguagens de alto nível simples para consultar os dados. As principais vantagens do modelo relacional em relação aos modelos de dados mais antigos são sua representação de dados simples e a facilidade com que mesmo consultas complexas podem ser expressas (RAMAKRISHNAN; RAGHU, 2008).

Os bancos de dados relacionais foram desenvolvidos para prover acesso facilitado aos dados, possibilitando que os usuários utilizassem uma grande variedade de abordagens no tratamento das informações. Pois, enquanto em um banco de dados hierárquico os usuários precisam definir as questões de negócios de maneira específica, iniciando pela raiz do mesmo, nos bancos de dados relacionais os usuários podem fazer perguntas relacionadas aos negócios através de vários pontos. A linguagem padrão dos bancos de dados relacionais é a SQL (RAMAKRISHNAN; RAGHU, 2008).

O modelo relacional formulado originalmente por Codd usou deliberadamente certos termos, como o próprio termo relação, que não eram familiares nos círculos de tecnologia da informação (TI) da época (embora os conceitos fossem familiares em alguns casos). O problema era que muitos dos termos mais familiares eram bastante vagos – eles não tinham a precisão necessária a uma teoria formal do tipo que Codd estava propondo. Por exemplo, considere o termo registro. Em diferentes momentos e em diferentes contextos, esse único termo pode significar uma ocorrência de um registro ou um tipo de registro; um registro lógico ou um registro físico; um registro armazenado ou um registro virtual; e talvez outros significados além destes. Por essa razão, o modelo relacional não usa de forma alguma o termo registro; em vez disso, ele utiliza o termo tupla (que rima com “dupla”), que recebe uma definição muito precisa. O termo tupla corresponde aproximadamente à noção de linha (assim como o termo relação corresponde aproximadamente à noção de tabela). Da mesma maneira, o modelo relacional não utiliza o termo campo; em vez disso, ele utiliza o termo atributo, que, para nossos propósitos nesse momento, podemos dizer que corresponde aproximadamente à noção de uma coluna de uma tabela (DATE, 2003).

### 2.1.1 Tabela

Uma tabela nada mais é do que um conjunto de tuplas ou linhas contendo atributos de uma entidade. Uma tabela é um conjunto não ordenado de linhas (tuplas na terminologia acadêmica). Assim sendo, quando descrito que uma entidade tem como característica permitir o armazenamento, na realidade é uma representação, do que ela poderá armazenar, quando for

uma tabela em um banco de dados relacional (HEUSER, 2002).

Tabelas no modelo de banco de dados relacionais também são chamadas de entidades. Segundo Elmasri e Shamkant (2011), o objeto básico do modelo ER é representar uma entidade, que é algo no mundo real com uma existência independente. Uma entidade pode ser um objeto com uma existência fisíca (por exemplo, uma pessoa em particular, um carro, uma casa ou um funcionário), ou pode ser um objeto com uma existência conceitual (por exemplo, uma empresa, um cargo ou um curso universitário).

Elmasri e Shamkant (2011), diz ainda que cada entidade possui atributos – as propridades que as descrevem. Por exemplo, uma entidade FUNCIONARIO pode ser descrita pelo nome, idade, endereço, salário e cargo do funcionário. Uma entidade em particular terá um valor para cada um de seus atributos. Os valores de atributo que descrevem cada entidade tornam-se uma parte importante dos dados armazenados no banco de dados. Um banco de dados em geral contem grupos de entidades que são semelhantes. Por exemplo, uma empresa que emprega centenas de funcionáriospode querer armazenar informações semelhantes com relação a cada um dos funcionários. Essas entidades de funcionário compartilham os mesmos atributos, mas cada uma tem o(s) próprio(s) valor(es) para cada atributo. A Tabela 1 mostra a organização de uma tabela no modelo de banco de dados relacionais, onde a coluna **CodFuncionario** é chave primária, a coluna **Nome** é atributo e **CodEndereco** é chave estrangeira. Os termos: chave primária e chave estrangeira são conceituados no item 2.1.3.

***Tabela 1*** *– Exemplo de uma tabela no modelo relacional.*

| $CodFuncionario$ | $Nome$ | $CodEndereco$ |
|---|---|---|
| 1 | João | 2 |
| 2 | Francisco | 4 |
| 3 | José | 5 |
| 4 | Maria | 3 |
| 6 | Conceição | 1 |

### 2.1.2 Atributos

O atributo é uma informação que se encontra em um campo da tabela, lembrando que a tabela é formada por um conjunto de linhas e as linhas são formadas por uma série de campos. Uma campo nada mais é do que uma das informações da linha. Cada atributo deve possuir um tipo, vários campos (com ou sem atributos) formam uma linha, cada linha representará a

informação completa de algo que se deseja armazenar, e todas as linhas formam uma tabela (HEUSER, 2002).

Os atributos não divisíveis são chamados atributos simples ou atômicos. Os atributos compostos podem formar uma hierárquia, o valor de uma atributo composto é a concatenação dos valores de seus componentes atributos simples. Na Tabela 2, os campos *"4"*, *"7 de setembro"*, *"64602435"* e os demais dados representam os atributos que vem em uma tabela no modelo relacional, nela o campo **CodEndereco** representa uma chave primária e os campos **Rua** e **CEP** são atributos.

***Tabela 2** – Exemplo de tabela com Atributos.*

| $codEndereco$ | $Rua$ | $CEP$ |
|---|---|---|
| 4 | 7 de setembro | 64602435 |
| 5 | alvarenga | 64523255 |
| 1 | Expedito lopes | 53344646 |
| 2 | getulio vargas | 63543745 |
| 3 | presidente dutra | 65734345 |

Segundo Elmasri e Shamkant (2011), atributos compostos são úteis para modelar situações em que um usuário às vezes se refere ao atributo composto como uma unidade, mas outras vezes se refere especificamente a seus componentes. Se o atributo composto for referenciado apenas como um todo, não é necessário subdividi-lo em atributos componentes. Por exemplo, se não for preciso referenciar os componentes individuais de um endereço (CEP, rua etc), então o endereço inteiro pode ser designado como um atributo simples.

### 2.1.3 Chaves

Segundo Heuser (2002), uma chave é o conceito básico para identificar linhas e estabelecer relações entre linhas de tabelas de um banco de dados. As chaves podem ser primária, estrangeira ou alternativa, a chave primária é uma coluna, ou ainda uma combinação de colunas (campos), cujos valores distinguirão uma linha das demais da tabela, que de forma simplificada é como se fosse um identificador de uma linha, aonde a unicidade de valores desse campo, ou campos, que é a chave primária deverá ser mantida.

Uma chave estrangeira é uma coluna ou uma combinação de colunas, cujos valores aparecem necessariamente na chave primária de outra tabela. A chave estrangeira é o mecanismo que permite a implementação de relacionamentos em um banco de dados relacional. Na Tabela 3 contém os campos **codDependente** e **codFuncionario**, ambos são campos chave, o primeiro

é chave primária da tabela DEPENDENTE e o segundo chave estrangeira.

Uma restrição importante das entidades de um tipo de entidade é a chave ou restrição de exclusividade sobre os atributos. Um tipo de entidade normalmente tem um ou mais atributos cujos valores são distintos para cada entidade individual no conjunto de entidades. Esse atributo é denominado **atributo-chave**, e seus valores podem ser usados para identificar cada entidade de maneira exclusiva. Por exemplo, o atributo **NOME** é uma chave do tipo de entidade **EMPRESA**, pois duas pessoas podem ter o mesmo nome. Para o tipo de entidade *PESSOA*, um atributo-chave típico é o CPF (Cadastro de Pessoa Física). Às vezes, vários atributos juntos formam uma chave, significando que a combinação dos valores de atributo deve ser distinta para cada entidade (ELMASRI; SHAMKANT, 2011).

***Tabela 3** – Tabela com chave primária **codDependente** e chave estrangeira **codFuncionario**.*

| $codDependente$ | $Nome$ | $codeFuncionario$ |
|---|---|---|
| 1 | Leonardo | 3 |
| 2 | Lucas | 10 |
| 3 | Junior | 4 |
| 4 | Pedro | 6 |
| 5 | Alex | 9 |

### 2.1.4 Relacionamentos

Relacionamentos representam a associação entre uma chave estrangeira e uma primária, seja a primária de outra tabela ou da mesma. Uma relação permite maior integridade para o banco de dados. Segundo Elmasri e Shamkant (2011), sempre que um atributo de um tipo de entidade se refere a outro tipo de entidade, existe um relacionamento. O grau de um tipo de relacionamento é o número dos tipos de entidade participantes. Um tipo de relacionamento de grau dois é chamado de binário, e um tipo de grau três é chamado de ternário. Os relacionamentos podem ser de qualquer grau, mas os mais comuns são relacionamentos binários. Relacionamentos de grau mais alto geralmente são mais complexos do que binários.

Os relacionamentos são utilizados para estabelecer um padrão a uma tupla especifica em uma entidade no banco dados. Se não existirem não haveria como encontrar uma informação no banco de dados com facilidade. A Figura 1 demonstra o exemplo de uma tabela com falta de relação entre os dados.

Na Figura 1, observa-se que se for feita uma consulta das cidades de um mesmo estado, não é possível retornar um resultado correto, pois há dois nomes diferentes para representar

um mesmo estado. Para resolver esse problema é necessário criar uma tabela para armazenar os estados e através de relacionamentos implementar as chaves primárias para representar cada estado.

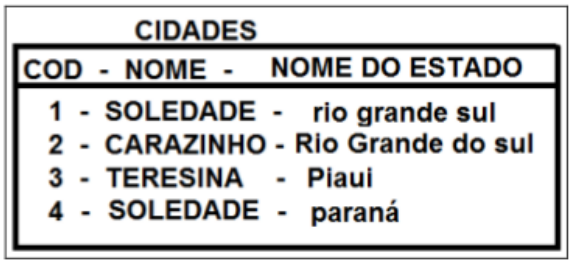



***Figura 1*** *– Falta de relacionamentos.*

## 2.2 Álgebra Relacional

Conforme Ramakrishnan e Raghu (2008), álgebra relacional é uma das duas linguagens formais de consulta associadas ao modelo relacional. Consultas em álgebra são construídas utilizando uma coleção de operadores. Cada operador aceita uma ou duas relações como parâmetro e retorna uma relação como resultado, esta é principal propriedade e torna fácil a composição de consultas complexas.

O conjunto básico de operações para o modelo relacional é a álgebra relacional. As operações de álgebra produzem, assim, novas relações, que podem ser manipuladas, usando-se as operações da mesma álgebra. Uma sequência de operações de álgebra relacional forma uma expressão de álgebra relacional cujos resultados também serão uma relação que representa o resultado de uma consulta de banco de dados (ou solicitação de recuperação) (ELMASRI; SHAMKANT, 2011).

As consultas na álgebra relacional são compostas usando uma coleção de operadores. Uma propriedade fundamental é a de que todo operador na álgebra aceita (uma ou duas) instâncias de relação como argumentos e retorna uma instância de relação como resultado. Esta propriedade facilita a composição de operadores para formar uma consulta complexa – uma expressão de álgebra relacional é recursivamente definida como uma relação, um operador de álgebra unário aplicado a uma única expressão, ou um operador de álgebra binário aplicado a duas expressões. Cada consulta em álgebra relacional descreve um procedimento passo a passo para computar a resposta desejada, baseado na ordem em que os operadores são aplicados na consulta (RAMAKRISHNAN; RAGHU, 2008).

Segundo Elmasri e Shamkant (2011) a álgebra relacional é muito importante por diversos motivos. Primeiro, ela oferece um alicerce formal para as operações do modelo relacional. Segundo, e talvez o mais importante, ela é usada como base para a implementação e otimização

de consultas nos módulos de otimização e processamento de consultas, que são partes integrais dos SGBDRs. Terceiro, alguns de seus conceitos são incorporados na linguagem de consulta padrão SQL para SGBDRs.

Aplicações com SGBDRs utilizam a álgebra relacional para representar internamente as consultas SQL construídas por usuários. Outra característica importante informada por Ramakrishnan e Raghu (2008), é que a álgebra relacional "é usada como uma base para desenvolver e otimizar as consultas em SGDBRs."

Elmasri e Shamkant (2011), diz ainda que, embora a maioria dos SGBDRs comerciais em uso não ofereçam interfaces de usuário para consultas da álgebra relacional, as operações e funções essenciais nos módulos internos da maioria dos sistemas relacionais são baseadas nas operações da álgebra relacional.

### 2.2.1 Operadores

Em relação às operações utilizadas na álgebra relacional, pode-se defini-las em dois grupos. Um grupo inclui as operações derivadas da teoria dos conjuntos da matemática, quais sejam: união, intersecção, diferença e produto cartesiano. O outro grupo compõe as operações desenvolvidas diretamente para os SGBDRs, como seleção, projeção, junção, renomear e divisão. (ELMASRI; NAVATHE, 2005).

Existem ainda os operadores classificados como estendidos ou adicionais como ordenação, funções de agrupamento e agregação. Nas seções subsequentes são apresentados os operadores. Inicialmente são abordados os tradicionais, na sequência os desenvolvidos para SGBDRs e em seguida os que compreendem a álgebra relacional estendida.

### 2.2.2 União, intersecção e diferença

Elmasri e Shamkant (2011) descreve que os operadores união, intersecção e diferença na álgebra relacional clássica têm seu funcionamento igual ao da teoria dos conjuntos da matemática. Estas são operações binárias, isto é, cada uma é aplicada a duas relações e são definidas da seguinte forma:

a) união: o resultado desta operação, indicada por R $\cup$ S, é uma relação que inclui todas as tuplas que estão em R, ou em S, ou em ambas, R e S. As tuplas repetidas são eliminadas;

b) intersecção: o resultado desta operação indicada por R $\cap$ S, é uma relação que inclui todas as tuplas que estão em ambas, R e S;

c) diferença: o resultado desta operação indicada por R - S, é uma relação que inclui todas as tuplas que estão em R, mas não estão em S.

Quando qualquer uma das três operações for utilizada deve-se também respeitar a regra

de compatibilidade da união, ou seja, as duas relações devem ter o mesmo número de atributos e estes devem ser do mesmo tipo.

### 2.2.3 Selecionar ($\sigma$)

A operação seleção é usada para escolher um subconjunto das tuplas de uma relação que satisfaça uma condição de seleção. Pode-se considerar que a operação seleção seja um filtro que mantém apenas as tuplas que satisfaçam uma condição qualificadora. Como alternativa, podemos considerar que essa operação restringe as tuplas em uma relação para apenas aquelas que satisfazem a condição. A operação seleção também pode ser visualizada como uma partição horizontal da relação em dois conjuntos de tuplas – aquelas que satisfazem a condição e são selecionadas, e aquelas que não satisfazem a condição e são descartadas (ELMASRI; SHAMKANT, 2011).

Segundo Elmasri e Shamkant (2011), uma consulta em álgebra relacional para selecionar a tupla funcionario cujo departamento é 4, ou aquelas cujo salário é maior do que 30.000, podemos especificar individualmente cada uma dessas duas condições com uma operação seleção da seguinte maneira:

$$\sigma\text{Dnr=4(FUNCIONARIO)}$$

$$\sigma\text{Salario>30.000(FUNCIONARIO)}$$

Em geral a operação seleção é indicada por:

$$\sigma\text{<condição seleção> (R)}$$

Onde o símbolo $\sigma$ (sigma) é usado para indicar o operador seleção e a condição de seleção é uma expressão *booleana* (condição) especificada nos atributos da relação R. Observe que R costuma ser uma expressão da álgebra relacional cujo resultado é uma relação – a mais simples expressão desse tipo é apenas o nome de uma relação de banco de dados. A relação resultante da operação seleção tem os mesmos atributos de R.

Elmasri e Shamkant (2005) diz ainda que a expressão *booleana* especificada em <condição seleção> é composta de uma série de cláusulas da forma <nome atributo> <op comparação> <valor constante> ou <nome atributo> <op comparação> <nome atributo> onde <nome atributo> é o nome de um atributo R, <op comparação> em geral é um dos operadores (=, <, $\leq$, >, $\geq$, $\neq$) e <valor constante> é um valor constante do domínio do atributo. As cláusulas podem ser conectadas pelos operadores *booleanos* padrão *and*, *or* e *not* para formar uma condição de seleção geral. Por exemplo, para selecionar as tuplas para todos os funcionários que ou trabalham no departamento 4 e ganham mais de 25.000 por ano, ou trabalham no departamento 5 e ganham mais de 30.000, podemos especificar a seguinte operação seleção:

$\sigma$(Dnr=4 AND Salario>25.000) OR (Dnr=5 AND Salario>30.000)(FUNCIONARIO)

O operador seleção tem como objetivo selecionar um determinado conjunto de tuplas de uma relação respeitando uma condição de seleção. Pode ser visualizada como uma divisão de uma relação em dois conjuntos, aquele conjunto que satisfaça a condição e é apresentado, e o conjunto que não satisfaz a condição e é descartado. O símbolo utilizado para esta operação é o sigma ($\sigma$) (SANTOS, 2010).

A especificação da operação seleção tem o formato $\sigma$ <condição de seleção> (R) onde símbolo $\sigma$ (sigma) indica a operação, e a condição de seleção é uma expressão *booleana* definida a partir dos atributos da relação R. Verifica-se que R no seu formato mais simples é diretamente o nome de uma relação do banco de dados, contudo pode ser uma expressão de álgebra relacional cujo resultado é uma relação (ELMASRI; SHAMKANT, 2005). A Figura 2 apresenta um exemplo da operação seleção, onde há uma seleção usando a chave primária **a**, das tuplas em que o campo **c** é maior do que 20.

Seja a relação **R** (a,b,c):

| a | b | c |
|---|---|---|
| 1 | João | 20 |
| 2 | Marcos | 21 |
| 3 | Maria | 23 |

O resultado de $\sigma c > 20$ (**R**):

| a | b | c |
|---|---|---|
| 2 | Marcos | 21 |
| 3 | Maria | 23 |

***Figura 2** – Exemplo do Operador seleção.*

### 2.2.4 Projeção ($\pi$)

Para se obter determinados atributos de uma relação é utilizada a operação projeção, a operação é executada em apenas uma relação e o resultado é uma nova relação contendo apenas os atributos especificados, eliminando-se as eventuais duplicidades entre as tuplas.

A especificação da operação projeção segue o formato $\pi$ <lista de atributos> (R) onde, o símbolo $\pi$ (pi) indica a operação e, a lista de atributos, informa quais colunas devem ser apresentadas entre aquelas da relação R. Verifica-se que R no seu formato mais simples é diretamente o nome de uma relação do banco de dados, contudo pode ser uma expressão de álgebra relacional cujo o resultado é uma relação. (NAVATHE, 2005). A figura 3 demonstra um exemplo deste operador, dos campos **a, b e c** projeta-se **a e b**.

Seja a relação R (a,b,c) :

| a | b | c |
|---|---|---|
| 1 | João | 20 |
| 2 | Marcos | 21 |
| 3 | Maria | 23 |

O resultado de $\pi_{a,b}$ (R):

| a | b |
|---|---|
| 1 | João |
| 2 | Marcos |
| 3 | Maria |

***Figura 3*** *– Exemplo do Operador projeção.*

### 2.2.5 Renomear ($\rho$)

A operação renomear tem como objetivo rebatizar o nome de uma relação, ou os nomes de seus atributos. Esta técnica pode auxiliar no uso das operações como união e junção. Para renomear os atributos em uma relação, simplesmente listamos os novos nomes de atributos entre parênteses.

A operação renomear ($\rho$) pode ser usada também para resolver os possíveis conflitos de nomes que podem surgir em alguns casos. Assim é conveniente sermos capazes de nomear explicitamente os campos de uma instância de relação que é definida por uma expressão de álgebra relacional. De fato, normalmente é conveniente dar um nome à instância propriamente dita, de maneira que possamos dividir uma grande expressão algébrica em partes menores, nomeando os resultados das expressões (RAMAKRISHNAN; RAGHU, 2008). A Figura 4 demonstra um exemplo deste operador, nesta operação passa-se como paramentro de **R** os campos **a, b e c** e usando o símbolo renomear ($\rho$) altera-se os nomes de **a, b e c** para **código, nome e idade** respectivamente.

Seja a relação R (a,b,c) :

| a | b | c |
|---|---|---|
| 1 | João | 20 |
| 2 | Marcos | 21 |
| 3 | Maria | 23 |

O resultado de ρ(código,nome,idade) (R):

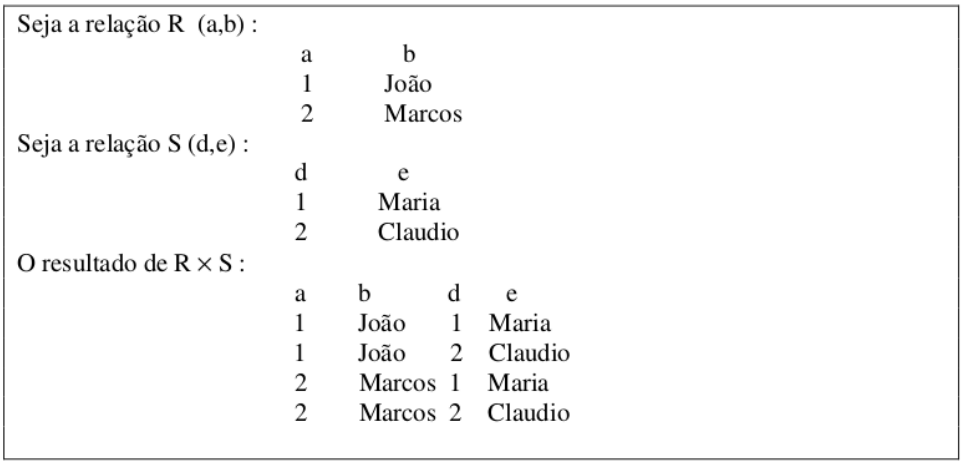
| código | nome | idade |
|---|---|---|
| 1 | João | 20 |
| 2 | Marcos | 21 |
| 3 | Maria | 23 |

***Figura 4*** *– Exemplo do Operador Renomear.*

As seguintes operações padrão sobre conjuntos também estão disponíveis na álgebra relacional:

### 2.2.6 Produto Cartesiano ($\times$)

Segundo (ELMASRI; SHAMKANT, 2005), a operação produto cartesiano é referenciada pelo símbolo $\times$, a mesma também atua sobre duas relações, contudo não é necessária a compatibilidade da união para sua utilização. Sua função é combinar as tuplas de duas relações na forma combinatória, desta forma o resultado de R $\times$ S é uma relação Q que terá a soma dos atributos de R e S e o número de tuplas igual a quantidade de tuplas de R multiplicada pela quantidade de tuplas de S. A figura 5 demonstra um exemplo deste operador.


Seja a relação R (a,b) :

| a | b |
|---|---|
| 1 | João |
| 2 | Marcos |

Seja a relação S (d,e) :

| d | e |
|---|---|
| 1 | Maria |
| 2 | Claudio |

O resultado de R $\times$ S :

| a | b | d | e |
|---|---|---|---|
| 1 | João | 1 | Maria |
| 1 | João | 2 | Claudio |
| 2 | Marcos | 1 | Maria |
| 2 | Marcos | 2 | Claudio |

***Figura 5*** *– Exemplo do Operador Produto Cartesiano.*

### 2.2.7 Junções

A operação junção é uma das mais úteis na álgebra relacional e a maneira mais comumente usada para combinar informações de duas ou mais relações. Uma junção pode ser definida como um produto cartesiano seguido de seleções e projeções. Na prática as junções aparecem com muito mais frequência do que os produtos cartesianos normais.

Segundo Elmasri e Shamkant (2011), cada SGBD normalmente tem uma série de algoritmos gerais de acesso de banco de dados que implementam operações da álgebra relacional como SELEÇÃO ou JUNÇÃO ou combinações dessas operações. Somente estratégias de execução que podem ser implementadas pelos algoritmos de acesso do SGBD e que se aplicam à consulta em particular, bem como ao projeto de banco de dados físico em particular, podem ser consideradas pelo módulo de otimização de consulta.

### 2.2.8 Junção Natural ($\bowtie$)

A operação junção natural é uma operação binária que permite combinar as operações de seleção e produto cartesiano dentro de uma única operação.

A operação junção theta é utilizada para combinar tuplas relacionadas em duas relações

dentro de uma tupla única. Pode ser definida por um produto cartesiano seguido de uma Seleção.

A especificação da operação junção theta segue o formato R <condição de junção> S onde, o símbolo ⋈ indica a operação e, a condição de junção, informa quais colunas devem ser levadas em consideração na combinação das tuplas de R e S. Apenas as tuplas que satisfazem a condição de junção são apresentadas no resultado. Verifica-se que R no seu formato mais simples é diretamente o nome de uma relação do banco de dados, contudo pode ser uma expressão de álgebra relacional cujo resultado é uma relação (NAVATHE, 2005). A Figura 6 demonstra um exemplo deste operador.

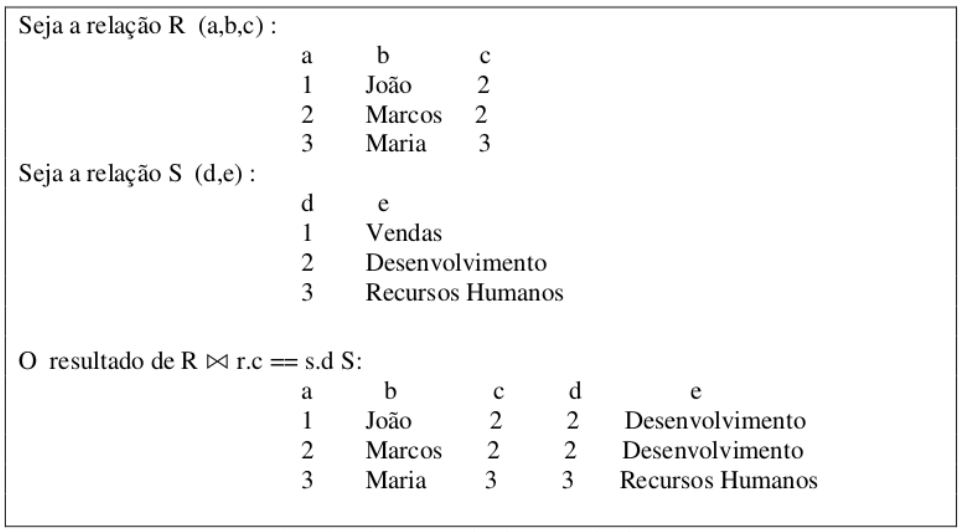



***Figura 6*** *– Exemplo do Operador Junção Natural.*

### 2.2.9 Operador Divisão (/)

Esse operador é útil para expressar certos tipos de consultas. A operação de divisão, simbolizada por "/", é usada nas consultas nas quais se exige que todos os valores dos atributos do divisor estejam presentes no dividendo (RAMAKRISHNAN; RAGHU, 2008).

As operações da álgebra relacional são muitos importantes para entender os tipos de solicitação que podem ser especificadas em um banco de dados relacional. Elas também são importantes para processamento e otimização de consulta em um SGBD Relacional. As operações da álgebra relacional são consideradas muito técnicas para a maioria dos usuários de SGBDs comerciais, pois uma consulta em álgebra relacional é escrita como uma sequência de operações que, quando executadas, produz o resultado exigido. O usuário precisa especificar como – ou seja, em que ordem – executar as operações de consulta. Por sua vez, a linguagem SQL oferece uma interface de linguagem declarativa de nível mais alto, de modo que o usuário apenas especifica qual deve ser o resultado, deixando a otimização real e as decisões sobre como

executar a consulta para o SGBD. Embora a SQL inclua alguns recursos da álgebra relacional, ela é baseada em grande parte no cálculo relacional de tupla. Porém a sintaxe SQL é mais fácil de ser utilizada do que qualquer uma das duas linguagens formais (ELMASRI; SHAMKANT, 2011).

## 2.3 Linguagem SQL

Originalmente desenvolvida como linguagem de consulta do SGBD Relacional pioneiro da IBM, o *System-R*, a linguagem de consulta estruturada (SQL, de *Structured Query Language*) tornou-se a mais usada para criar, manipular e consultar SGBDs Relacionais. Como muitos fabricantes oferecem produtos SQL, há necessidade de um padrão que defina a "SQL oficial". A existência de um padrão permite aos usuários medir a consistencia da versão SQL de determinados fabricantes. O padrão também permite aos usuários distinguir recursos da SQL específicos de um produto daqueles que são padronizados; um aplicativo que conta com recursos não padronizados é menos portátil (RAMAKRISHNAN; RAGHU, 2008).

O primeiro padrão SQL foi desenvolvido em 1986 pelo *Americam National Standards Institute* (ANSI) e foi chamado SQL-86. Houve uma pequena revisão em 1989, chamada SQL-89, e uma revisão maior, em 1992, chamada SQL-92. A *International Standards Organizacion* (ISO) colaborou com a ANSI no desenvolvimento do padrão SQL-92. Atualmente, a maioria dos SGBDs comerciais suporta a versão SQL:1999 do padrão, uma extensão importante da SQL-92 adotada recentemente (RAMAKRISHNAN; RAGHU, 2008).

As operações de conjunto da SQL estão disponíveis em álgebra relacional. A principal diferença, naturalmente, é que elas são operações em multiconjuntos na SQL. Já que as tabelas são multiconjuntos de tuplas.

A linguagem SQL pode ser considerada um dos principais motivos para o sucesso dos bancos de dados relacionais comerciais. Como ele se tornou um padrão para esse tipo de banco de dados, os usuários ficarão menos preocupados com a migração de suas aplicações de outros tipos de sistemas de banco de dados – por exemplo, sistemas de rede e hierárquicos – para sistemas relacionais. Isso aconteceu porque, mesmo que os usuários estivessem insatisfeitos com o produto de SGBD relacional em particular que estava usando, a conversão para outro produto de SGBD relacional não seria tão cara ou demorada, pois os dois sistemas seguiam os mesmos padrões de linguagem. Na prática é obvio, existem muitas diferenças entre diversos pacotes de SGBDs relacionais comerciais. Porém se o usuário for cuidadoso em usar apenas dos recursos que fazem parte do padrão, e se os dois sistemas relacionais admitirem fielmente o padrão, então a conversão entre ambos deverá ser bastante simplificada. Outra vantagem de ter esse padrão é que os usuários podem escrever comandos em um programa de aplicação de

banco de dados que pode acessar dados armazenados em dois ou mais SGBDs relacionais sem ter de mudar a sublinguagem de banco de dados (SQL) se os sistemas admitirem o padrão SQL (ELMASRI; SHAMKANT, 2011).

SQL é uma linguagem de banco de dados abrangente: tem instruções para definição de dados, consultas e atualizações. Ela é uma DDL (*Definition Data Language*) e uma DML (*Data Manipulation Language*). Além disso, ela tem facilidades para definir visões sobre o banco de dados, para especificar segurança e autorização, para definir restrições de integridade e para especificar controles de transação. Ela também possui regras para embutir instruções SQL em uma linguagem de programação de uso geral, como *Java*, *COBOL* ou C/C++ (ELMASRI; SHAMKANT, 2011).

Historicamente, a álgebra e o cálculo relacional foram desenvolvidos antes da linguagem SQL. De fato, de algumas maneiras, a SQL é baseada nos conceitos tanto da álgebra quanto do cálculo. Um modelo de dados precisa incluir um conjunto de operações para manipular o banco de dados para definir a estrutura e as restrições do banco de dados. O conjunto básico de operações para o modelo relacional é a álgebra relacional. Essas operações permitem que um usuário especifique as solicitações de recuperação básicas com expressões da álgebra relacional. O resultado de uma recuperação é uma nova relação, que pode ter sido formada de uma ou mais relações. As operações da álgebra, assim, produzem novas relações, que podem ser manipuladas ainda mais usando operações da mesma álgebra. Uma sequência de operações da álgebra relacional forma uma expressão da álgebra relacional, cujo resultado também será uma relação que representa o resultado de uma consulta de banco de dados (ou consulta de recuperação) (ELMASRI; SHAMKANT, 2011).

Segundo Elmasri e Shamkant (2011), em um SGBD, uma consulta costuma ter muitas estratégias de execução possíveis, e o processo de escolha de uma estratégia adequada para processá-la é conhecido como otimização de consulta. O SGBD precisa então idealizar uma estratégia de execução ou plano de consulta para recuperar os resultados da consulta com base nos arquivos de banco de dados.

## 2.4 Compiladores

O compilador é um tipo de tradutor de linguagens de programação. Tradutores de linguagens de programação são sistemas que aceitam como entrada o código de um programa, escrito em uma determinada linguagem (código-fonte), e produzem como saída, um código equivalente para o mesmo programa (código-objeto), porém escrito em outra linguagem. No caso específico dos compiladores, trata-se de tradutores capazes de mapear códigos escritos em linguagens de alto nível para códigos equivalentes em linguagem simbólica ou linguagem de

máquina (AHO, 1995).

Monte e Lopes (2013), diz que o processo de compilação é dividido em duas fases: análise e síntese. Na análise, o código fonte é didivido nas suas partes constituintes com o objetivo de gerar uma estrutura intermediária capaz de representá-lo, enquanto a síntese consiste em construir o código-objeto a partir da representação intermediária gerada pela fase de análise.

Após o processo de compilação pode-se implementar a fase de otimização de código, segundo Monte e Lopes (2013), esta fase, como o nome diz, é responsável por otimizar o código em termos de velocidade de execução e utilização de memória, de forma que o código-objeto a ser gerado na próxima fase possa ser mais eficiente.

## 2.5 Linguagem de Programação *Java*

Segundo Deitel (2006), o *Java* foi anunciado formalmente pela *Sun*, em uma conferência que aconteceu em maio de 1995. Chamou a atenção da comunidade de negócios por causa do enorme interesse na *World Wide Web*. É uma linguagem que já se expandiu no mundo inteiro.

A linguagem *Java* é conhecida mundialmente. É uma linguagem orientada a objetos fazendo parte da quinta geração de linguagem de programação juntamente com C++, *Delphi*, *Visual Basic*, C♯, *VB.Net*. O *Java* fornece uma variedade de bibliotecas pré-definidas que auxiliam bastante no desenvolvimento de aplicações. Destaca-se das outras linguagens orientadas a objeto por dar suporte a vários tipos de aplicações, com propósitos diferentes. *Java* é a base para praticamente todos os tipos de aplicação em rede e é o padrão global para o desenvolvimento e entrega de aplicações móveis, jogos, conteúdo baseado na *web* e *softwares* empresariais. Com mais de 9 milhões de desenvolvedores em todo o mundo, *Java* permite desenvolver e implementar de forma eficiente aplicativos interessantes e serviços. Com ferramentas abrangentes, um ecossistema maduro, e desempenho robusto, *Java* oferece portabilidade aplicações em ambientes de computação. (ORACLE, 2013).

Para desenvolvimento de aplicações na linguagem *Java*, é necessário apenas o *kit* de desenvolvimento *Java development Kit* (JDK) que apresenta a versão 1.7 como a mais recente, e uma plataforma (IDE) de desenvolvimento. Sendo que para que as aplicações escritas na linguagem *Java* sejam executadas, deve-se ter o JDK instalado no computador.

## 2.6 Plataforma *NetBeans*

A plataforma *NetBeans* é uma estrutura genérica para aplicações *Swing* (biblioteca gráfica suportada pela linguagem *Java*. E fornece o caminho que antes cada desenvolvedor tinha

que realizar a conexão para itens de menu, barra de ferramentas e atalhos de teclado, gerenciamento de janelas, e assim por diante. A Plataforma *NetBeans* fornece uma arquitetura confiável e flexível. O aplicativo não enxerga qualquer coisa como uma IDE . Como a arquitetura da plataforma *NetBeans* é modular, é fácil criar aplicações que são robustas e extensíveis. Uma arquitetura que incentiva práticas de desenvolvimento sustentável (NETBEANS, 2014).

O *NetBeans* apresenta uma *interface* de fácil manipulação, é uma plataforma que funciona em vários sistemas operacionais, facilitando assim para os desenvolvedores, que podem optar qual sistema operacional usar.

A plataforma *NetBeans* fornece uma infraestrutura para registrar e recuperar implementações de serviço, o que permite minimizar as dependências diretas entre os módulos individuais e que permitem uma arquitetura flexível. Também fornece um sistema de arquivos virtual. Inclui uma *Application Programming Interface* (API) unificada, proporciona fluxo orientado acesso a estruturas planas e hierárquicas, como baseados em disco arquivo sem servidores locais ou remotos, de memória baseados em arquivos e documentos *eXtensible Markup Language* (XML) (NETBEANS, 2014).

## 2.7 O *Parser* ZQL

O ZQL é um analisador SQL escrito em *Java*. Ele analisa uma expressão SQL e preenche estruturas *Java* que representam instruções e expressões SQL. Um avaliador de expressão SQL vem com ZQL *Parser*, assim pode-se facilmente avaliar expressões SQL fora do analisador. Porém, o criador do ZQL alerta: “o ZQL é livre para o uso, mas não oferece nenhuma garantia". As APIs ZQL podem ser sujeitos a alterações, para enriquecer as funcionalidades ou corrigir *bugs*. O analisador em si é escrito com *JavaCC*, um gerador de *parser Java* (como o popular gerador *yacc* do *Unix*). Toma como entrada declarações SQL (*SELECT*, *INSERT*, *UPDATE*, *DELETE*, *COMMIT* , *ROLLBACK*, e transação), preenche as estruturas *Java* que representam as declarações analisadas.

O ZQL é uma ferramenta de código livre com licença da GNU *(General Public License)*, está na versão 3. É permitida a alteração do código, acrescentar funcionalidades e métodos de verificação às classes do programa, também é permitida a distribuição do programa (GIBELLO, 2013).

# 3 TRABALHOS RELACIONADOS

Nesta seção são apresentados alguns trabalhos relacionados que tratam da conversão entre linguagens ou entre Sistemas Gerenciadores de Banco de Dados (SGBD).

## 3.1 SQL *Convert*

SQL *Convert*: ferramenta para aprendizagem de álgebra relacional. Desenvolvido por Anderson de Oliveira Monte e Mariana Lisboa Lopes. Essa ferramenta tem o seu maior enfoque no ensino e a aprendizagem, e a análise das expressões escritas em álgebra relacional, assim como a conversão das expressões, para comandos equivalentes em SQL. O seu objetivo é oferecer uma ferramenta de aprendizagem, onde seja possível construir operações escritas em álgebra relacional, fazer as análises dessas expressões, a procura de erros léxicos, sintático e semânticos e posteriormente converter essas operações em comandos da linguagem SQL.

Segundo Monte e Lopes (2013), o desenvolvimento dessa ferramenta de aprendizagem possibilita a professores e alunos um auxílio ao ensino e aprendizagem de álgebra relacional e da linguagem SQL, pois tal ferramenta é capaz de fazer a análise de expressões escritas em álgebra relacional, indicando a existência de erros em suas análises léxica, sintática e semânticas e realizar a conversão dessas expressões em comandos equivalentes na linguagem SQL.

## 3.2 CSDC

CSDC – ferramenta de conversão de *scripts* SQL em diagrama de classes UML. desenvolvida por João Lucas Goergen. Tem o objetivo de realizar o processo de engenharia reversa de um *script* SQL para um diagrama de classes UML, além da utilidade que o uso desta oferece, pois pode contribuir em termos de planejamento para sistemas informatizados em geral.

O programa visa realizar a conversão de um *Script* SQL de um banco de dados relacional para um diagrama de classes, e ainda possibilitando a modularização das classes geradas, ou seja, uma organização por grupos/setores pré-definidos pelo usuário, e levando em conta a problemática acima descrita, a ferramenta desenvolvida torna-se essencial nesse contexto, afirma (GOERGE, 2012).

## 3.3 Conversor de Banco de Dados *Oracle* para *PostgreSQL*

A ferramenta é um conversor de dados do *Oracle* para o *PostgreSQL*. O programa foi desenvolvido baseado nas linguagens HTML, DHTML, PHP(*Hypertext Preprocessor*), *javascript* e uma base de dados *MySQL Server*, para a manipulação dos registros. Desenvolvido por Cristiano Seiti Takano, o programa tem o objetivo de ajudar programadores que precisam fazer uma conversão do *Oracle* para o *PostgreSQL* e, um tutorial explicativo de fácil navegação e obtenção da informação, exibindo exemplo dos comandos SQL.

## 3.4 Sistema para aprendizado de álgebra relacional e linguagem SQL

O sistema para aprendizado de álgebra relacional é uma ferramenta educacional voltada para o ensino de álgebra relacional, permitindo que sejam construídos planos de consulta na forma de árvores de expressão algébricas. Esses planos de consulta são transformados em linguagem SQL e podem ser executados nos bancos de dados *Oracle* XE e *MySQL* 5.5.

O sistema desenvolvido por André Martins dos Santos tem como objetivo disponibilizar um programa para aprendizado de álgebra relacional e linguagem SQL (Processo básico de consulta) e oferecer um editor gráfico para a construção de árvores de expressões algébricas, realizando a conversão das árvores de expressões algébricas em linguagem SQL, onde a consulta resultante pode ser executada nos bancos de dados relacional *Oracle* e *MySQL*.

# 4 DESENVOLVIMENTO DA FERRAMENTA

Este trabalho apresenta o desenvolvimento de uma ferramenta para realizar a conversão de consultas escritas na linguagem SQL para expressões na linguagem álgebra relacional, para isso foi estudada a ferramenta ZQL *parser*, que oferece os métodos necessários para realizar a análise de consultas SQL e retornar expressões resultantes em álgebra relacional, e o assunto compiladores para um bom entendimento de como construir um compilador.

Para realizar a análise dessas expressões escritas na linguagem SQL, assim como para convertê-las para outra linguagem que possibilite a análise dos operadores de forma mais objetiva é necessário utilizar uma ferramenta que consiga interpretar as consultas, analisá-las e realizar a conversão para expressões equivalentes em álgebra relacional. A ferramenta foi desenvolvida utilizando a linguagem de programação *Java* com a IDE de desenvolvimento *Netbeans* 7.3.1.

## 4.1 Infraestrutura para o desenvolvimento do trabalho

Para o desenvolvimento do programa foi necessário utilizar ferramentas específicas que auxiliaram bastante para a concretização deste trabalho. Os estudos realizados foram com base em pesquisas e projetos relacionados à área de conversão entre linguagens de programação, com ênfase na utilização da linguagem de programação *Java* e compiladores.

## 4.2 Classes Usadas do ZQL *parser*

A ferramenta ZQL *parser* foi construída para realizar a análise de expressões escritas na linguagem SQL, sejam elas consultas ou comandos da linguagem de definição de dados (DDL). O programa é utilizado para realizar a análise da consulta inserida e retorná-los ao programa cliente que os armazena em vetores correspondentes e finalmente pode imprimi-los organizados de acordo com as regras da álgebra relacional juntamento com os atributos definidos na consulta.

### 4.2.1 Classe *ZqlParser.java*

Uma das classes utilizadas do programa ZQL *parser* é a classe *ZqlParser.java*, nesta classe estão os métodos que analisam as instruções SQL digitadas pelo usuário ou a partir de um arquivo de texto. O código utilizado para ler a *string* de entrada é apresentado a seguir.

```
01 public static void main(String args[])throws ParseException{
02   ZqlParser p = null;
03     if ( args.length < 1  ) {
04 System.out.println("/* Reading from stdin (exit; to finish)
05   */");
06      p = new ZqlParser(System.in);
07     } else {
08   try {
09  p = new ZqlParser(new DataInputStream(new FileInputStream
10     (args[0])));
11      } catch (FileNotFoundException e) {
12        System.out.println("/* File " + args[0] +
13         " not found. Reading from stdin */");
14        p = new ZqlParser(System.in);
15      }
16    } // else ends here
17  if ( args.length > 0 ) {
18      System.out.println("/* Reading from " + args[0] + "
19    */");
20    }
21    ZStatement st = null;
22    while((st = p.readStatement()) != null) {
23      System.out.println(st.toString() + ";");
24    }
25    System.out.println("exit;");
26    System.out.println("/* Parse Successful */") ;
```

Na linha 3 verifica-se a existência de pelo menos um elemento no vetor de *strings*, caso o vetor esteja vazio, na linha 05 a classe requisita nova entrada. Quando há *strings* (informações) no vetor, o método **readStatement()** que implementa o método **SQLStatements()**, as verifica uma a uma comparando com os comandos declarados no método **SQLStatements()** declarado

na classe *ZqlJJParser.java*. O método **SQLStatements()** que verifica os comandos da consulta com os *tokens* declarados na ferramenta é apresentado a seguir.

```
01 final public Vector SQLStatements() throws ParseException {
02  Vector v = new Vector();
03  ZStatement s;
04    label_1:
05    while (true) {
06      s = SQLStatement();
07         if(s == null) {if (true) return v;} else v.addElemen
08        t(s);
09      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
10      case K_COMMIT:
11      case K_DELETE:
12      case K_EXIT:
13      case K_INSERT:
14      case K_LOCK:
15      case K_QUIT:
16      case K_ROLLBACK:
17      case K_SELECT:
18      case K_SET:
19      case K_UPDATE:
20        ;
21        break;
22      default:
23        jj_la1[4] = jj_gen;
24        break label_1;
25      }
26    }
27      {if (true) return v;}
28    throw new Error("Missing return statement in function");
29  }
```

Este código apresenta um dos métodos existentes na ferramenta ZQL *Parser* que realiza a comparação de uma *string* de entrada com os *tokens* declarados.

### 4.2.2 Classe *ZqlJJParser.java*

Esta classe contém os métodos que comparam os comandos presentes na consulta SQL que deseja-se converter com os *tokens* declarados na ferramenta. O método **BasicDataTypeDeclaration()** compara comandos de declaração de tipos, como por exemplo: *CHAR*, *INTEGER*, *VARCHAR* e etc. O código deste método é mostrado no Anexo A. O método **CommitStatement()** compara comandos como: *COMMIT*, *WORK* e *COMMENT*. No código apresentado a seguir é possível ver como são comparados os comandos apresentados anteriormente. A classe *ZqlJJParser.java* contém métodos para comparar cada um dos comandos pertencentes à linguagem SQL, isso mostra que a ferramenta ZQL *Parser* é robusta e oferece muitas possibilidades de implementação a quem deseja desenvolver uma ferramenta como a apresentada neste trabalho.

```
01 final public ZTransactStmt CommitStatement() throws Parse
02    Exception {
03  Token tk;
04  ZTransactStmt t = new ZTransactStmt("COMMIT");
05    jj_consume_token(K_COMMIT);
06    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
07    case K_WORK:
08      jj_consume_token(K_WORK);
09      break;
10    default:
11      jj_la1[7] = jj_gen;
12      ;
13    }
14    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
15    case K_COMMENT:
16      jj_consume_token(K_COMMENT);
17      tk = jj_consume_token(S_CHAR_LITERAL);
18      t.setComment(tk.toString());
19      break;
20    default:
21      jj_la1[8] = jj_gen;
22      ;
23    }
24    jj_consume_token(91);
25    {if (true) return t;}
```

```
26  throw new Error("Missing return statement in function");
27  }
```

## 4.3 Detalhes de implementação

A análise das consultas inseridas na ferramenta é realizada através da implementação das classes do programa ZQL *Parser*. Nestas classes há os métodos responsáveis por comparar a consulta de entrada e obter um vetor de comandos. Cada comando da consulta é inserido em um vetor correspondente para obter-se a melhor organização possível e retornar estes comandos com os simbólos da álgebra relacional e na ordem necessária para representar uma expressão algébrica. O código completo da classe principal da ferramenta é mostrado no Apêndice A. O método **Converter()**, que realiza a conversão da consulta na ferramenta é mostrado a seguir.

```
01 public void Converter() throws ParseException{
02        String SQL;
03               Vector selectArea = null;
04               Vector  fromArea = null;
05               ZExpression whereArea = null;
06               Vector GroupByArea = null;
07               Vector OrderByArea = null;
08               Vector BetweenArea = null;
09               Vector Andarea = null;
10
11    SQL = jTextArea1.getText();
12    ZqlParser p = new ZqlParser();
13    p.initParser(new ByteArrayInputStream(SQL.getBytes()));
14
15      ZQuery query = null;
16      query = (ZQuery)p.readStatement();
17      Vector select = query.getSelect();
18      Vector from = query.getFrom();
19      ZExpression where = (ZExpression) query.getWhere();
20      ZGroupBy groupby = query.getGroupBy();
21      Vector orderby = query.getOrderBy();
22      %jTextArea2.setText("$\sigma$"+select.toString()+"\n
23      @("+where.toString()+from.toString()+")");
```

```
24      }
```

Neste método, a consulta digitada na *JTextArea* da parte superior na *interface* da ferramenta, é verificada pelo método **initParser()** da classe *ZqlParser.java*, onde a consulta é analisada e os comandos são retornados à ferramenta. O código do método **initParser()** é mostrado a seguir.

```
01 public void initParser(InputStream in) {
02    if(_parser == null) {
03      _parser = new ZqlJJParser(in);
04    } else {
05      _parser.ReInit(in);
06    }
07  }
```

Considerando que a ferramenta tenha sido inicializada e o usuário esteja utilizando a tela principal, o processo de conversão da consulta SQL para a álgebra relacional é basicamente o seguinte:

1. Inserir uma consulta SQL na *JTextArea* da parte superior da ferramenta e clicar no botão "converter";

2. A ferramenta requisita ao programa ZQL *Parser* que analise a consulta e retorne os comandos da linguagem SQL presentes na consulta;

3. A ferramenta armazena esses comandos em vetores e os imprime juntamente com os símbolos da álgebra relacional organizados na ordem em que deve-se apresentar uma expressão algébrica.

# 5 A FERRAMENTA PARA CONVERSÃO DE CONSULTAS SQL PARA ÁLGEBRA RELACIONAL

A ferramenta para realizar a conversão de consultas SQL para a linguagem álgebra relacional (Figura 7), foi escrita na linguagem *Java* versão 7 utilizando a plataforma de desenvolvimento IDE *NetBeans* versão 7.3.1 juntamento com o JDK 1.7. A ferramenta possui uma *interface* gráfica intuitiva que permite ao usuário interagir durante o processo de conversão das consultas.

Existem projetos semelhantes que executam operações de conversão de expressões na linguagem álgebra relacional para a linguagem SQL que estimulam o aprendizado da álgebra relacional para gerar consultas SQL. Neste projeto pretende-se também estimular o aprendizado da álgebra, porém, o usuário precisa ter conhecimento da linguagem SQL para inserir os comandos na ferramenta e convertê-los para álgebra relacional. Assim, é possível o usuário aprender a álgebra relacional a partir da linguagem SQL.

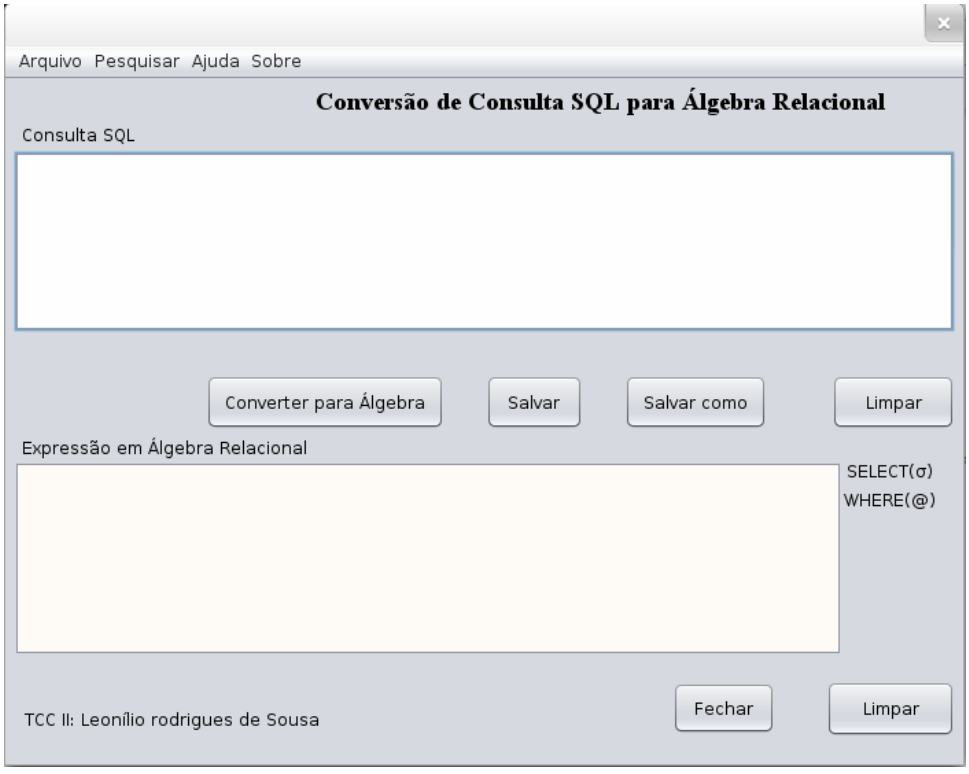



***Figura 7** – Ferramenta para conversão de consultas SQL para álgebra relacional.*

A *interface* da ferramenta Figura 7 oferece as opções para inserir a consulta na *JTextArea* da parte superior da tela e convertê-la. Além disso, oferece no menu as opções de **Arquivo**, **Pesquisar**, **Ajuda** e **Sobre**. Em ajuda pode-se obter informações de como utilizar a ferramenta e em sobre há informações sobre desenvolvedores da ferramenta e conceitos de banco de dados, linguagem SQL e álgebra relacional. O usuário deverá digitar a consulta SQL na *JTextArea* da parte superior, caso a consulta apresente algum erro, será mostrada uma mensagem (figura 8) informando ao usuário que a consulta está incorreta, pode-se clicar no botão *"ok"* e redigitar a consulta de forma correta.

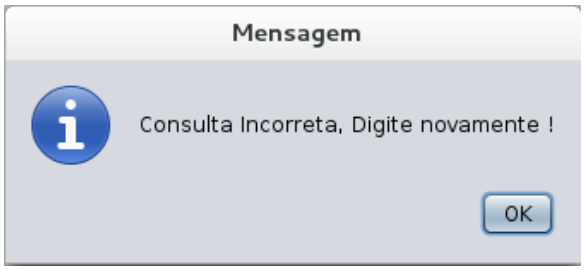



***Figura 8*** *– Mensagem consulta incorreta.*

Finalmente, após digitar a consulta corretamente ou corrigí-la, caso exista erro, na *JTextArea* superior presente na tela principal da ferramenta (Figura 9), o usuário obterá a expressão algébrica quivalente à consulta inserida na *JTextArea* inferior da tela principal da ferramenta de conversão.

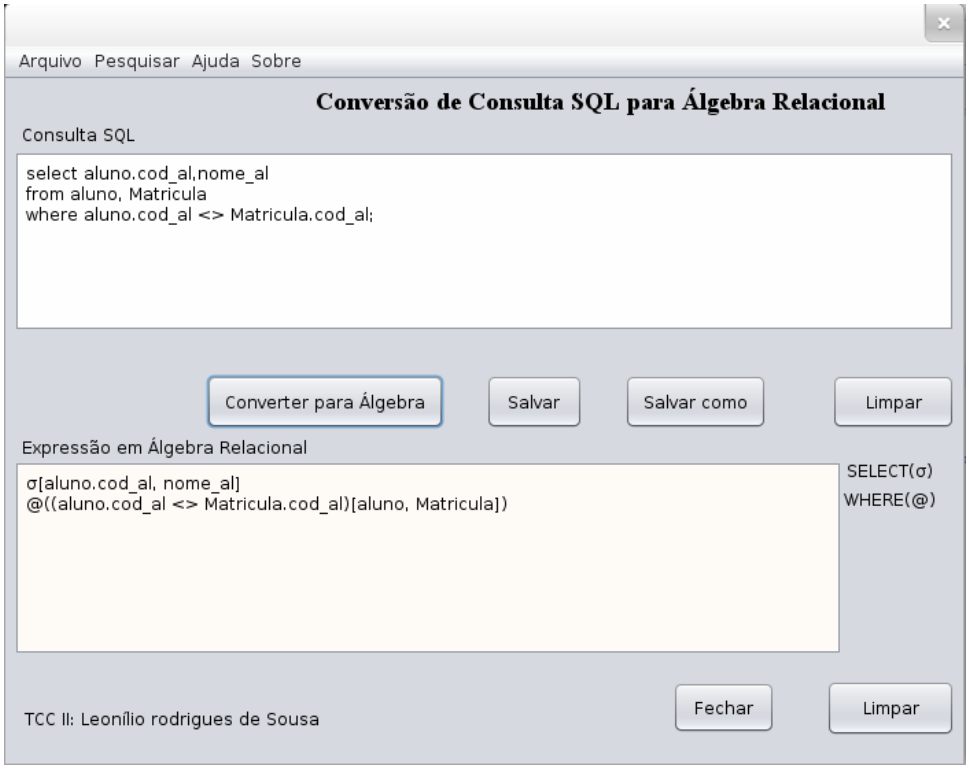



***Figura 9*** *– Consulta convertida.*

# 6 TESTES E RESULTADOS

Os testes foram realizados para verificar a eficiência da execução da ferramenta tanto com consultas simples quanto com consultas mais complexas que apresentam muitos operadores e atributos. Além disso os testes são importantes para avaliar a usabilidade da *interface* gráfica responsável por exibir os campos de entrada, os botões de operações e os campos de saída. Foram inseridas consultas escritas na linguagem SQL e retornados os resultados das conversões na linguagem álgebra relacional.

## 6.1 Ambiente de Execução

Para a execução dos testes foi utilizado um computador com a seguinte configuração: com processador *intel dual core* ™ 2 GHz com 2 GB de RAM e 160 de HD, e com sistema operacional *Debian* 7.5 (*wheezy*) 32-bit com ambiente gráfico *GNOME* 3.4.2.

## 6.2 Cenários de Teste

Os testes foram realizados com o objetivo de provar a eficácia da ferramenta de conversão de consultas SQL para álgebra relacional. Inicialmente, foram feitas várias conversões das quais algumas são simples e outras mais complexas. Em seguida, foram escolhidos dois testes, o primeiro convertendo uma consulta simples e o segundo convertendo uma consulta complexa com maior número de comandos e atributos.

Para os cenários, os seguintes testes foram realizados: I) converteu-se a consulta: *SELECT* * *FROM* aluno *WHERE* matricula = 201023345; II) a consulta: *SELECT* a.cod_al, a.nome_al,c.cod_curso, c.nome_curso, c.ch *FROM* aluno a, cursot c, matricula m *WHERE* a.cod_al=m.cod_al *AND* m.cod_curso=c.cod_curso.

### 6.2.1 Cenário I: Conversão de uma Consulta Simples

A consulta simples possui poucos comandos, assim foi convertida e ordenada facilmente pela ferramenta. O tempo de execução desta operação na ferramenta é muito pequeno, já que a consulta tratasse de uma *string* pequena, a execução não utiliza muitos recursos de processamento do computador. A Figura 10 mostra a tela com a consulta em SQL na *JTextArea* da parte superior e a consulta convertida para a álgebra relacional na *JTextArea* inferior.

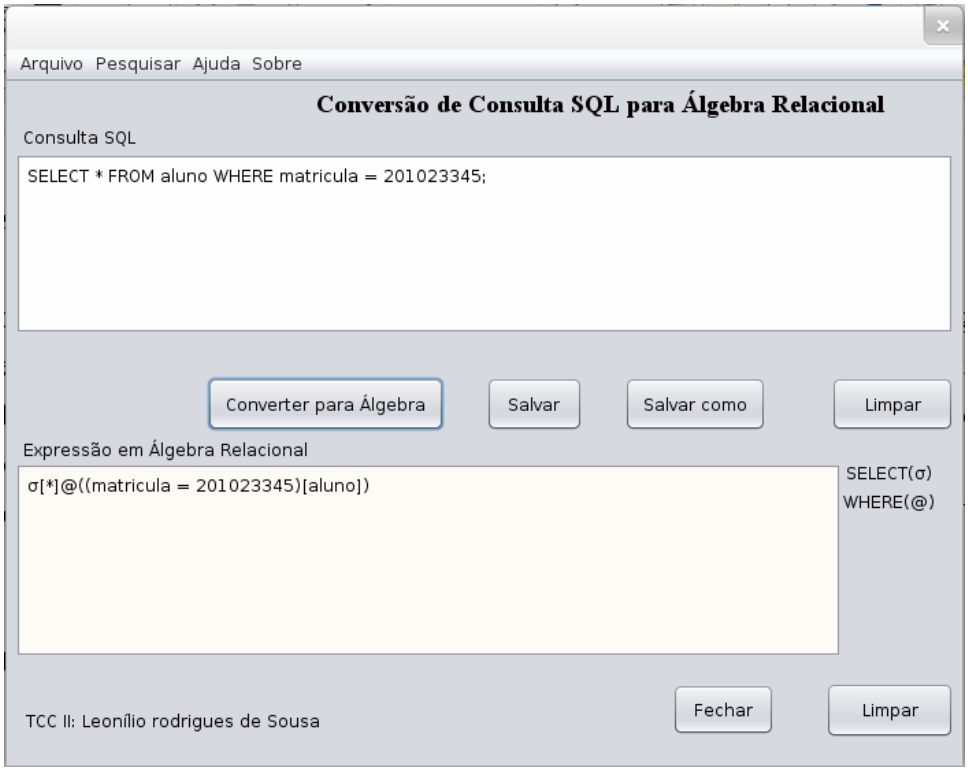



***Figura 10*** *– Tela com a conversão de consulta simples para a álgebra relacional.*

### 6.2.2 Cenário II: Conversão de uma Consulta Complexa

A quantidade de comandos que podem aparecer em uma consulta a qual deseja-se converter, pode ser muito grande. Foram inseridas consultas com uma quantidade considerável de comandos e a ferramenta as converteu e retornou o resultado em milésimos de segundo, mostrando que o desempenho da ferramenta é muito bom, mesmo inserindo consultas com grande quantidade de informações. A Figura 11 mostra a tela com a conversão de uma consulta com número considerável de comandos.

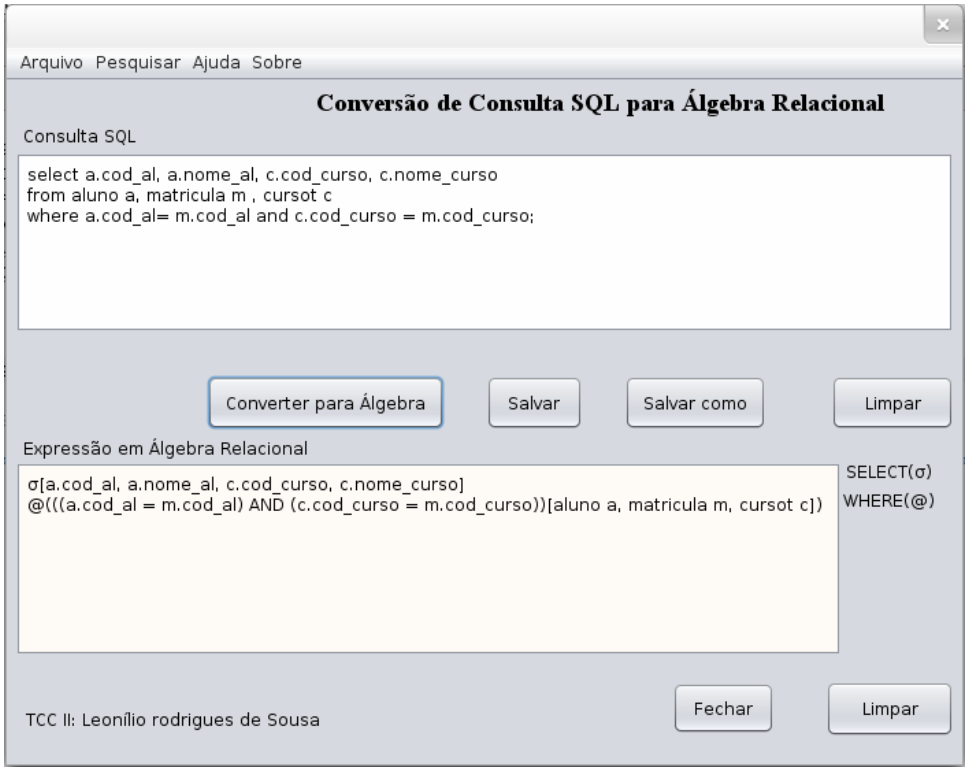



***Figura 11*** *– Tela com a conversão de consulta SQL complexa para a álgebra relacional.*

Os resultados das consultas convertidas podem ser utilizados para vários fins. Esses resultados podem ser utilizados para otimizar a execução de consultas em SGBDRs, para auxiliar no ensino da linguagem álgebra relacional através do uso da linguagem SQL e também para profissionais utilizarem como prática de aperfeiçoamento no desenvolvimento de consultas e reforçar o conhecimento tanto em banco de dados quanto em linguagens para banco de dados relacionais.

# 7 CONSIDERAÇÕES FINAIS

O presente trabalho apresenta uma ferramenta de grande importância para profissionais interessados em analisar consultas SQL. A ferramenta funciona como um auxilio para os usuários que desejam verificar a disposição dos símbolos em álgebra depois da conversão. A álgebra relacional tem uma característica procedimental e seus comandos executam de forma sequencial, ou seja, os símbolos serão executados na ordem em que se encontram.

A ferramenta apresentou boa usabilidade e retornou resultados equivalentes, ela retorna atributos de consultas complexas, organizando todos os atributos dessas consultas com o seu respectivo símbolo em álgebra relacional.

Essa ferramenta pode ajudar tanto aos profissionais quanto aos estudantes que querem obter conhecimento da linguagem álgebra relacional através da linguagem SQL, uma vez que já tenham bom conhecimento de SQL.

O sistema foi implementado com sucesso, pois a ferramenta é capaz de realizar as funções que lhe foram propostas desde o início da realização deste trabalho.

E por fim, ficam como sugestões para trabalhos futuros os seguintes itens:

a) Criar métodos para converter também *strings* que contenha operações de renomear, intersecção e atribuição;

b) Conectar a ferramenta com um SGBD para ao mesmo tempo retorna o resultado da consulta de um determinado banco de dados e oferecer suporte ao SGBD *MySQL Server* e *PostegreSQL*.

# Referências

AHO, A. V. *D. Compiladores - Princípios, Técnicas e Ferramentas*. [S.l.]: LTC, 1995.

CODD, E. F. Relational model of data for large shared data banks. *ACM Journaling*, v. 1, p. 11, 1970.

DATE, C. J. *Introdução a Sistemas de Banco de Dados*. 8. ed. Rio de Janeiro: Elsevier, 2003.

DEITEL, P. J. *Java: como programar*. 6. ed. São Paulo: Pearson Education do Brasil, 2006.

ELMASRI, Ramez; SHAMKANT, B. Navathe. *Sistemas de banco de dados*. 3. ed. São Paulo: Pearson Addison Wesley, 2005.

ELMASRI, Ramez; SHAMKANT, B. Navathe. *Sistemas de banco de dados*. 6. ed. São Paulo: Pearson Addison Wesley, 2011.

GARCIA, M. H. *Implementação de Sistemas de Banco de Dados*. Rio de Janeiro: Campus, 2001.

GIBELLO. *ZQL: Um Analisador Java SQL*. 2013. Disponível em: <http://zql.sourceforge.net>. Acesso em: Out. 2013.

GOERGE, João Lucas. Csdc – uma ferramenta de conversÃo de script sql em diagrama de classes uml. In: . Passo Fundo: [s.n.], 2012.

HEUSER, C. A. *Projeto de Banco de Dados*. 4. ed. Rio de Janeiro: S. Luzzatto, 2002.

MONTE, A. O.; LOPES, M. L. Sql convert: ferramenta para aprendizagem de álgebra relacional. In: . Blumenau: [s.n.], 2013.

NETBEANS. *The NetBeans Platform*. 2014. Disponível em: <https://netbeans.org/features/platform/index.html>. Acesso em: Dezembro. 2013.

ORACLE. *ORACLE Java*. 2013. Disponível em: <http://www.oracle.com/us/technologies/java/overview/index.html>. Acesso em: Mar. 2013.

RAMAKRISHNAN; RAGHU, J. G. *Sistemas de gerenciamento de banco de dados*. 3. ed. São Paulo: McGraw-Hill, 2008.

SANTOS, A. M. Sistema para aprendizado de Álgebra relacional e linguagem sql. In: . Santa Catarina: [s.n.], 2010.

SUMATHI, S. *Fundaments of relational database management systems*. New York: Spring Verlag, 2007.

# ANEXO A – Método *BasicDataTypeDeclaration()*

```
01 final public void BasicDataTypeDeclaration() throws Parse
02   Exception{
03    switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
04   case K_CHAR:
05    case K_FLOAT:
06    case K_INTEGER:
07    case K_NATURAL:
08    case K_NUMBER:
09    case K_REAL:
10    case K_VARCHAR2:
11    case K_VARCHAR:
12      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
13      case K_CHAR:
14        jj_consume_token(K_CHAR);
15        break;
16      case K_VARCHAR:
17        jj_consume_token(K_VARCHAR);
18        break;
19      case K_VARCHAR2:
20        jj_consume_token(K_VARCHAR2);
21        break;
22      case K_INTEGER:
23        jj_consume_token(K_INTEGER);
24        break;
25      case K_NUMBER:
26        jj_consume_token(K_NUMBER);
27        break;
28      case K_NATURAL:
29        jj_consume_token(K_NATURAL);
30        break;
31      case K_REAL:
```

```
32        jj_consume_token(K_REAL);
33        break;
34      case K_FLOAT:
35        jj_consume_token(K_FLOAT);
36        break;
37      default:
38        jj_la1[0] = jj_gen;
39        jj_consume_token(-1);
40        throw new ParseException();
41      }
42      switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
43      case 88:
44        jj_consume_token(88);
45        jj_consume_token(S_NUMBER);
46        switch ((jj_ntk==-1)?jj_ntk():jj_ntk) {
47        case 89:
48          jj_consume_token(89);
49          jj_consume_token(S_NUMBER);
50          break;
51        default:
52          jj_la1[1] = jj_gen;
53          ;
54        }
55        jj_consume_token(90);
56        break;
57      default:
58        jj_la1[2] = jj_gen;
59        ;
60      }
61      break;
62    case K_DATE:
63      jj_consume_token(K_DATE);
64      break;
65    case K_BINARY_INTEGER:
66      jj_consume_token(K_BINARY_INTEGER);
67      break;
```

```
68    case K_BOOLEAN:
69      jj_consume_token(K_BOOLEAN);
70      break;
71    default:
72      jj_la1[3] = jj_gen;
73      jj_consume_token(-1);
74      throw new ParseException();
75    }
76  }
```

# APÊNDICE A – Código da classe principal da ferramenta

```
01 package InterfaceGrafic;
02 import java.io.ByteArrayInputStream;
03 import java.util.Vector;
04 import java.util.logging.Level;
05 import java.util.logging.Logger;
06 import javax.swing.JOptionPane;
07 import org.gibello.zql.ParseException;
08 import org.gibello.zql.ZExpression;
09 import org.gibello.zql.ZGroupBy;
10 import org.gibello.zql.ZQuery;
11 import org.gibello.zql.ZqlParser;
12   public class TCCII_Leonilio extends javax.swing.JFrame {
13   public TCCII_Leonilio() {
14        initComponents();
15       this.setLocation(280,100);
16        jTextArea2.setEditable(false);
17        }
18 private void jMenuItem4ActionPerformed(java.awt.event.Action
19     Event evt) {
20     // TODO add your handling code here:
21     int input = JOptionPane.showConfirmDialog(this, "Deseja
22     Sair ?", "Confirmação", JOptionPane.YES_NO_OPTION);
23     if(input == JOptionPane.YES_NO_OPTION){
24     System.exit(0);
25     }
26    }
27   private void jMenuItem7ActionPerformed(java.awt.event.
28   ActionEvent evt) {
29        // TODO add your handling code here:
```

```
30        new Sobre().show();
31        }
32   private void jMenuItem6ActionPerformed(java.awt.event.
33   ActionEvent evt) {
34        // TODO add your handling code here:
35        new AlgebraRelac().show();
36    }
37   private void jBFecharActionPerformed(java.awt.event.
38   ActionEvent evt) {
39        // TODO add your handling code here:
40    int input = JOptionPane.showConfirmDialog(this, "Deseja
41  Sair ?", "Confirmação", JOptionPane.YES_NO_OPTION);
42        if(input == JOptionPane.YES_NO_OPTION){
43        System.exit(0);
44        }
45    }
46     private void jBLimparActionPerformed(java.awt.event.
47     ActionEventevt) {
48        // TODO add your handling code here:
49        jTextArea1.setText(null);
50        }
51  private void jBConverterActionPerformed(java.awt.event.
52  ActionEvent evt) {
53            try {
54                // TODO add your handling code here:
55                VerificaDados();
56                Converter();
57            SQL = jTextArea1.getText();
58            jTextArea2.setText(SQL);*/
59            } catch (ParseException ex) {
60  JOptionPane.showMessageDialog(null, "Consulta Incorreta,
61  Digitenovamente !");
62  Logger.getLogger(TCCII_Leonilio.class.getName()).log(Level.
63  SEVERE,null, ex);
64        }
65    }
```

```
66  private void jMenuItem1ActionPerformed(java.awt.event.
67 ActionEvent evt) {
68        // TODO add your handling code here:
69        jTextArea1.setText(null);
70        jTextArea2.setText(null);
71    }
72   private void jBLimpar2ActionPerformed(java.awt.event.
73  ActionEvent evt) {
74        // TODO add your handling code here:
75        jTextArea2.setText(null);
76    }
77    public void Converter() throws ParseException{
78        String SQL;
79               Vector selectArea = null;
80               Vector  fromArea = null;
81               ZExpression whereArea = null;
82               Vector GroupByArea = null;
83               Vector OrderByArea = null;
84               Vector BetweenArea = null;
85               Vector Andarea = null;
86        SQL = jTextArea1.getText();
87        ZqlParser p = new ZqlParser();
88        p.initParser(new ByteArrayInputStream(SQL.get
89       Bytes())); ZQuery query = null;
90        query = (ZQuery)p.readStatement();
91        Vector select = query.getSelect();
92        Vector from = query.getFrom();
93        ZExpression where = (ZExpression) query.getWhere();
94        ZGroupBy groupby = query.getGroupBy();
95        Vector orderby = query.getOrderBy();
96       jTextArea2.setText("$\sigma$"+select.toString()+"\n
97      @("+where. toString()+from.toString()+")");
98        }
99    public boolean VerificaDados(){
100        if (!jTextArea1.getText().equals("")){
101       return true;
```

```
102       }else{
103       JOptionPane.showMessageDialog(null, "Insira a
104      consulta SQL  na area de texto acima !");
105       return false;
106       }
107    }
```