UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

FERRAMENTA DE APOIO AO PROJETO FÍSICO DE BANCO DE DADOS UTILIZANDO SISTEMA ESPECIALISTA

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDO À UNIVERSIDADE REGIONAL DE BLUMENAU PARA A OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA COM NOME EQUIVALENTE NO CURSO DE CIÊNCIAS DA COMPUTAÇÃO — BACHARELADO

FERNANDO EUGÊNIO BAULER

BLUMENAU, JULHO/2002

2002/1-33

FERRAMENTA DE APOIO AO PROJETO FÍSICO DE BANCO DE DADOS UTILIZANDO SISTEMA ESPECIALISTA

FERNANDO EUGÊNIO BAULER

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIO PARA OBTENÇÃO DO TÍTULO DE:

BACHAREL EM CIÊNCIAS DA COMPUTAÇÃO

_____

Prof. Alexander Roberto Valdameri — Orientador na FURB

_____

Prof. José Roque Voltolini da Silva — Coordenador do TCC

BANCA EXAMINADORA

_____

Prof. Alexander Roberto Valdameri

_____

Prof. Roberto Heinzle

_____

Prof. Maurício Capobianco Lopes

AGRADECIMENTO

Aos meus pais, Sônia e Osni, que sempre me apoiaram no desenvolvimento da minha profissão e sempre estiveram presentes nos momentos difíceis.

Em especial ao meu irmão Bruno, que sempre esteve ao meu lado e sempre foi um grande companheiro.

A minha namorada Sueli pela atenção, ajuda e compreensão nesta importante etapa de minha vida.

Ao meu amigo, Paulo Henrique, que sempre me ajudou, desde o começo da minha vida profissional.

Ao meu orientador, Alexander Valdameri, que me guiou e me ajudou sempre que necessitei.

Ao professor Roberto Heinzle pelas dicas e orientações no momento inicial do meu projeto.

A todos meus amigos que sempre estiveram ao meu lado.

A Deus, a Jesus e ao meu grande Anjo da Guarda por estar sempre presente.

A todos, meus sinceros agradecimentos.

SUMÁRIO

AGRADECIMENTO III

RESUMO VIII

ABSTRACT IX

1 INTRODUÇÃO 1

1.1 OBJETIVOS 2

1.2 ORGANIZAÇÃO DO TRABALHO 2

2 BANCO DE DADOS 4

2.1 BANCO DE DADOS RELACIONAL 4

2.2 SISTEMA GERENCIADOR DE BANCO DE DADOS 5

2.2.1 COMPOSIÇÃO 6

2.2.2 CARACTERÍSTICAS 7

2.3 ARQUITETURA 7

2.4 NÍVEL INTERNO 8

2.5 ACESSO AO BANCO DE DADOS 9

2.6 ÍNDICES 11

2.6.1 ÍNDICES COMPOSTOS 12

2.7 LINGUAGEM ESTRUTURADA 12

2.7.1 LINGUAGEM DE DEFINIÇÃO DE DADOS 13

2.8 ORACLE 14

2.8.1 TABLESPACES E DATAFILES 15

2.8.2 BLOCOS, EXTENSÕES E SEGMENTOS 15

2.8.3 CRIAÇÃO DAS TABELAS 17

2.8.4 TIPOS DE DADOS 18

2.9 MICROSOFT SQL SERVER 19
2.9.1 PÁGINAS E EXTENSÕES 21
2.9.2 FILEGROUPS 23
2.9.3 CRIAÇÃO DAS TABELAS 24
2.9.4 TIPOS DE DADOS 24
3 INTELIGÊNCIA ARTIFICIAL 27
3.1 SISTEMAS ESPECIALISTAS 27
3.2 COMPONENTES DO SISTEMA ESPECIALISTA 28
3.2.1 BASE DO CONHECIMENTO 28
3.2.2 AQUISIÇÃO DO CONHECIMENTO 29
3.2.3 MOTOR DE INFERÊNCIA 29
3.2.4 SISTEMA DE JUSTIFICAÇÃO 29
3.2.5 SISTEMA DE CONSULTA 30
3.2.6 QUADRO NEGRO 30
4 TÉCNICAS E FERRAMENTAS UTILIZADAS 31
4.1 INTELIGENCIA ARTIFICIAL 31
4.1.1 AQUISIÇÃO DO CONHECIMENTO 31
4.2 COMPILADORES 32
4.2.1 ANALISADOR LÉXICO 32
4.2.2 ANALISADOR SINTÁTICO 33
4.3 DELPHI 33
4.4 FERRAMENTAS CASE 33
4.5 ESPECIFICAÇÃO 34
5 DESENVOLVIMENTO DO TRABALHO 35
5.1 ESPECIFICAÇÃO 35

5.1.1 DIAGRAMA DE CASO DE USO 35
5.1.2 DIAGRAMA DE CLASSES 37
5.1.3 DIAGRAMA DE SEQÜÊNCIA 39
5.2 IMPLEMENTAÇÃO 45
6 CONSIDERAÇÕES FINAIS 55
6.1 CONCLUSÃO 55
6.2 SUGESTÕES PARA TRABALHOS FUTUROS 55
6.3 LIMITAÇÕES 56
ANEXO I 57
ANEXO II 61
ANEXO III 70
ANEXO IV 71
REFERÊNCIAS 80

LISTA DE FIGURAS

Figura 1 – SGBD, INTERFACE COM O USUÁRIO. 5

Figura 2 – ARQUITETURA DE UM SGBD 8

Figura 3 – ACESSO AO BANCO DE DADOS 10

Figura 4 – *TABLESPACE* E *DATAFILES* 15

Figura 5 – BLOCO DE DADOS, EXTENSÕES E SEGMENTOS 16

Figura 6 – ESTRUTURA DO *MSSQL SERVER* 21

Figura 7: PÁGINA 22

Figura 8: EXTENSÕES 23

Figura 9 – DIAGRAMA DE CASO DE USO 36

Figura 10 – DIAGRAMA DE CLASSES 38

Figura 11 – DESENVOLVEDOR CADASTRA REGRAS 40

Figura 12 – USUÁRIO CADASTRA REGRAS 41

Figura 13 – ANALISAR *SCRIPT* 42

Figura 14 – ANALISAR TABELA 44

Figura 15 – TELA PRINCIPAL 46

Figura 16 – EDITAR BASE DE CONHECIMENTO 47

Figura 17 – APRESENTAÇÃO DAS TABELAS 48

Figura 18 – MAIORES INFORMAÇÕES SOBRE A TABELA 49

Figura 19 – NÍVEL DE UTILIZAÇÃO 50

Figura 20 – ÍNDICES 51

Figura 21 – *TABLESPACES* E *DATAFILES* 52

Figura 22 – ORDEM DOS CAMPOS 53

Figura 23 – RESULTADOS 54

RESUMO

Este trabalho descreve a implementação de uma ferramenta capaz de analisar *scripts* gerados a partir de ferramentas CASE. O conteúdo extraído é submetido à análise em um módulo baseado em sistemas especialistas, o qual busca agregar melhorias aos comandos de definição dos objetos nos bancos de dados. A ferramenta foi validada através da submissão aos *scripts* gerados a partir dos CASE *Power Designer* e *ER/Studio*.

ABSTRACT

This work describes the implementation of a tool capable to analyze scripts generated from CASE tools. The extracted content is submitted to analises in one modulates established in specialists systems, which search to add improvements to the commands of definition of objects in the databases. The tool was validated through the submission to scripts generated by the CASE Power Designer and ER/Studio.

1 INTRODUÇÃO

Com o advento das ferramentas CASE (*Computer Aided Software Engineering*), muitas das empresas de desenvolvimento de software adotaram uma nova postura (uso de ferramentas CASE) em relação à metodologia de desenvolvimento. Segundo Fisher (1990), as ferramentas para a engenharia de *software* reduzem substancialmente, ou eliminam, inúmeros problemas de projeto e desenvolvimento inerentes aos projetos de médio e grande porte.

Cabe ressaltar que tais ferramentas não auxiliam o desenvolvedor na forma de qualidade dos objetos físicos criados, já que estas ferramentas atendem a diversos bancos com diferentes configurações.

Sistemas Gerenciadores de Bancos de Dados (SGBD) armazenam dados em discos, para mais tarde, transformá-los em informações úteis para os usuários. Os sistemas fazem constantes operações de busca destes dados, fazendo transferência do disco para memória e vice-versa. Por esse motivo, precisa-se planejar muito bem as estruturas físicas, para não haver perda de performance, melhorando a velocidade e qualidade do software que está utilizando os dados armazenados.

A escolha de estruturas de armazenamento adequadas procura minimizar o número de acessos a disco, melhorando o desempenho, a concorrência e otimizando a utilização de espaço em disco. Além da escolha da estrutura de armazenamento para cada relação, o projeto físico de um Banco de Dados (BD) também inclui definição de índices, mecanismos de acesso adicionais e política de alocação de espaço em disco.

Segundo Rabuske (1995), um sistema especialista é um sistema computacional que resolve problemas de uma maneira bastante parecida com o especialista humano. São sistemas com um conhecimento específico e profundo sobre campos restritos do conhecimento. Sistemas especialistas procuram atingir soluções de determinados problemas do mesmo modo que especialistas humanos se estiverem sob as mesmas condições. Para o desenvolvimento deste trabalho será utilizada técnica de sistemas especialistas. Utiliza-se também de técnicas de compiladores para extração dos *tokens* dos *scripts* das ferramentas CASE.

1.1 OBJETIVOS

O objetivo principal deste trabalho é construir um protótipo que vem a auxiliar os desenvolvedores na criação dos objetos físicos, verificando as variáveis que cada projeto pode ter através de técnicas de inteligência artificial (IA), mais especificamente técnicas de sistemas especialistas (SE).

Este trabalho visa à construção de uma ferramenta de auxílio a desenvolvedores, onde pesquisar-se-à quais as características de cada banco de dados e quais as melhores formas de construção do modelo físico, para que os mesmos possam ter suas aplicações viabilizadas para os bancos de dados de forma eficaz.

Os objetivos específicos do trabalho são:

a) verificar as características dos bancos de dados *Oracle* e *MSSQL Server*;
b) obter *scripts* gerados pelas ferramentas CASE mais comuns no mercado (*Power Designer*, *ER/Studio*);
c) otimizar a forma de criação dos objetos de acordo com as especificações estudadas de cada banco de dados;
d) gerar o dicionário de dados a partir de scripts analisados.

1.2 ORGANIZAÇÃO DO TRABALHO

O primeiro capítulo apresenta a introdução, objetivos pretendidos e a organização do trabalho.

O segundo capítulo apresenta os conceitos de bancos de dados. Em seguida são apresentadas algumas características dos bancos de dados estudados: *Oracle* e *Microsoft SQL Server*.

O terceiro capítulo descreve a inteligência artificial e as técnicas dos sistemas especialistas.

O quarto capítulo demonstra as técnicas e ferramentas utilizadas neste trabalho. Descreve também um pouco sobre os compiladores, analisador léxico e sintático.

O quinto capítulo apresenta o desenvolvimento do trabalho, incluindo a descrição da especificação e da implementação do protótipo.

O sexto capítulo finaliza o trabalho, apresentando as conclusões, limitações e sugestões para novos trabalhos.

2 BANCO DE DADOS

Segundo Date (2000), o sistema de banco de dados (BD) é basicamente um sistema de manutenção de registros por computador, ou seja, um sistema cujo objetivo global é manter os dados e torná-los disponíveis quando solicitados.

Um sistema de banco de dados envolve quatro componentes principais:

a) dados: é a parte essencial dos BD, sendo que, mais tarde, estes dados serão transformados pelos usuários em informações úteis.
b) *hardware*: é composto pela memória e pelos discos que armazenam os dados.
c) *software*: é a parte que faz a interação do acesso físico com os usuários.
d) usuários: os usuários podem ser divididos em 3 classes: os programadores de aplicações ou desenvolvedores do sistema, os usuários finais propriamente ditos e o administrador de banco de dados (DBA).

Observam-se algumas vantagens utilizando um sistema de banco de dados em relação aos métodos tradicionais de armazenamento de dados, entre elas: é compacto, rápido, possibilita menos trabalho braçal, o fluxo é corrente e possui os dados centralizados, evitando a redundância e a inconsistência dos dados. Pode-se compartilhar os dados e aplicar restrições de segurança.

2.1 BANCO DE DADOS RELACIONAL

Segundo Date (2000), a maioria dos sistemas de bancos de dados dos últimos anos são relacionais. Este modelo representa a tendência dominante do mercado e é simplesmente um dos mais importantes de toda a história do ramo de banco de dados.

De maneira muito simples, Date (2000) diz que “banco de dados relacional é um banco de dados visto pelos usuários como um conjunto de tabelas”.

Segundo Fernandes (2000), o banco de dados relacional tem como objetivo implementar o modelo de dados relacional, com todas as características básicas, ou seja, entidades, atributos e relacionamentos.

Um *software* de banco de dados relacional usa tabelas para implementar entidades e colunas para atributos. As entidades (ou tabelas) são o espaço reservado para armazenar as informações e os atributos (ou colunas) seriam as formas de organização desta estrutura.

Os relacionamentos são mantidos por valor com o uso de *primary key* e de *foreing keys*. As *primary key* são as chaves primárias das tabelas. São os identificadores únicos de cada registro, para que, os mesmos possam ser referenciados por outras tabelas através das *foreing keys* (chaves estrangeiras).

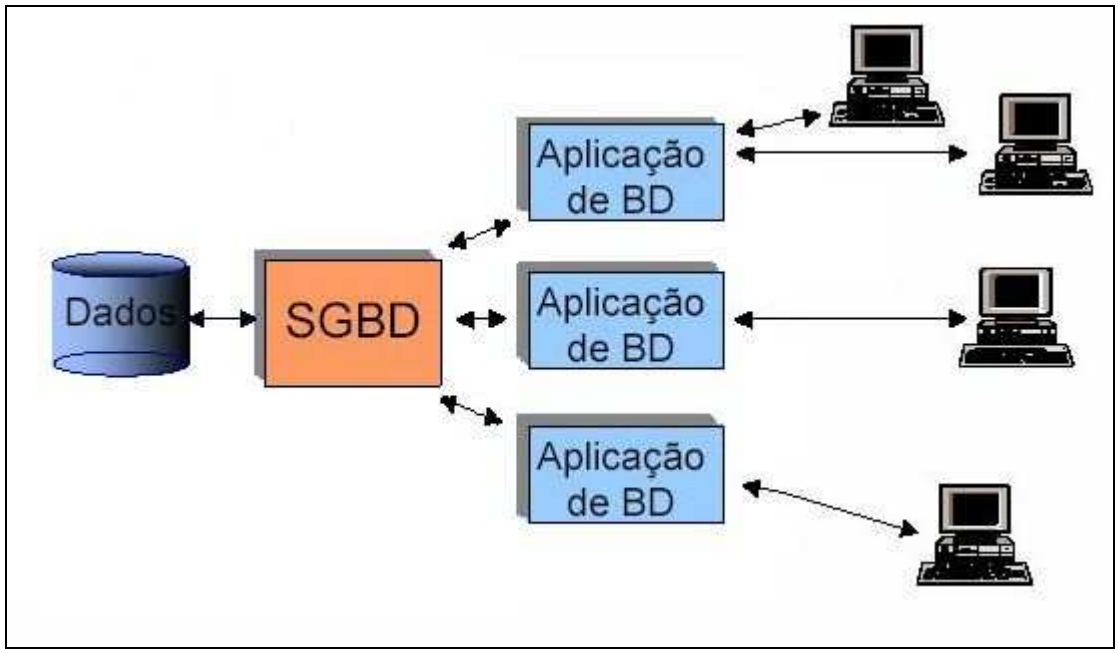
2.2 SISTEMA GERENCIADOR DE BANCO DE DADOS

Segundo Date (2000), "Sistema Gerenciador de Banco de Dados (SGBD) é o *software* que manipula todos os acessos ao banco de dados".

O SGBD é uma coleção de programas que permitem ao usuário definir, construir e manipular bases de dados para as mais diversas finalidades.

Para Date (2000), "Pode-se caracterizar também, que o SGDB proporciona a interface com o usuário ao sistema de banco de dados. A interface com o usuário pode ser definida como um limite no sistema abaixo do qual tudo é invisível ao usuário", conforme podemos visualizar na figura 1.

Figura 1 – SGBD, INTERFACE COM O USUÁRIO.



Fonte: Valdameri (2002)

Segundo Date (2000), o SGBD tem várias funções, que podem ser divididas em duas grandes categorias:

a) controle e proteção dos dados, que incluem controle de recuperação, concorrência, segurança e integridade;
b) geração de relatórios, gráficos comerciais, geração de aplicação entre outros.

O SGBD possui também algumas regras básicas, sendo elas:

a) auto-contenção: um SGBD não contém apenas os dados em si, mas armazena completamente toda a descrição dos dados, seus relacionamentos e formas de acesso;
b) independência dos dados: é quando a aplicação estiver realmente imune a mudança na estrutura de armazenamento ou na estratégia de acesso aos dados;
c) abstração dos dados: é fornecida ao usuário somente uma representação conceitual dos dados, o que não inclui maiores detalhes sobre sua forma de armazenamento real;
d) visões: deve-se permitir que cada usuário visualize os dados de forma diferente daquela existente previamente no banco de dados;
e) transações: exige-se que o banco de dados tenha ao menos uma instrução que permita a gravação de uma série de modificações simultâneas e uma instrução capaz de cancelá-las;
f) acesso automático: evitar a ocorrência de *dead-locks*, que são usuários esperando uma liberação de um outro usuário.

2.2.1 COMPOSIÇÃO

Segundo Date (2000), o SGBD deve ser capaz de aceitar as definições de dados, como um *script* de ferramenta CASE, e convertê-los para a forma objeto apropriado. Deve também poder lidar com as solicitações de pesquisa de dados feitas pelos usuários. Para isso, o SGBD possui os seguintes componentes:

a) gerenciador de acesso ao disco: SGBD utiliza o sistema operacional para acessar os dados armazenados em disco, controlando o acesso concorrente às tabelas;
b) compilador DDL (*Data Definition Language*): processa as definições do esquema do BD, acessando quando necessário o dicionário de dados;
c) dicionário de dados: contém as tabelas, índices, forma de acesso e relacionamentos existentes nos BD;

d) processador do BD: manipula requisições à própria base de dados em tempo de execução. É o responsável pelas atualizações e integridade;
e) processador de pesquisas: analisa as solicitações dos usuários e se estas forem consistentes, aciona o processador do BD para acesso efetivo aos dados.

2.2.2 CARACTERÍSTICAS

Os SGBD implementam algumas características operacionais básicas de controle e armazenamento de dados, segurança de acesso aos dados e de cópias de recuperação em caso de falhas no sistema.

Algumas características gerais de um SGBD são:

a) controle de redundâncias: as informações devem ser armazenadas em um único local, não existindo duplicação dos dados;
b) compartilhamento dos dados: o SGBD deve ter um controle de concorrência do acesso aos dados, garantindo a escrita/leitura de dados sem erros;
c) controle de acesso: recursos para controlar a autoridade de cada usuário;
d) interfaceamento: deve haver formas de acesso de forma "amigável", em linguagem natural, em SQL ou ainda via menus de acesso, não sendo somente acessível por aplicações;
e) esquematização: são mecanismos que possibilitem a compreensão e entendimento dos relacionamentos existentes entre as tabelas e de sua eventual manutenção;
f) controle de integridade: impedir que a integridade dos dados seja comprometida;
g) backups: recuperar os dados, por motivos de falhas de *hardware* ou *software*, através da existência de arquivos de backups.

Existem SGBD que não satisfazem completamente todas as características acima, porem isso não os inválida como SGBD.

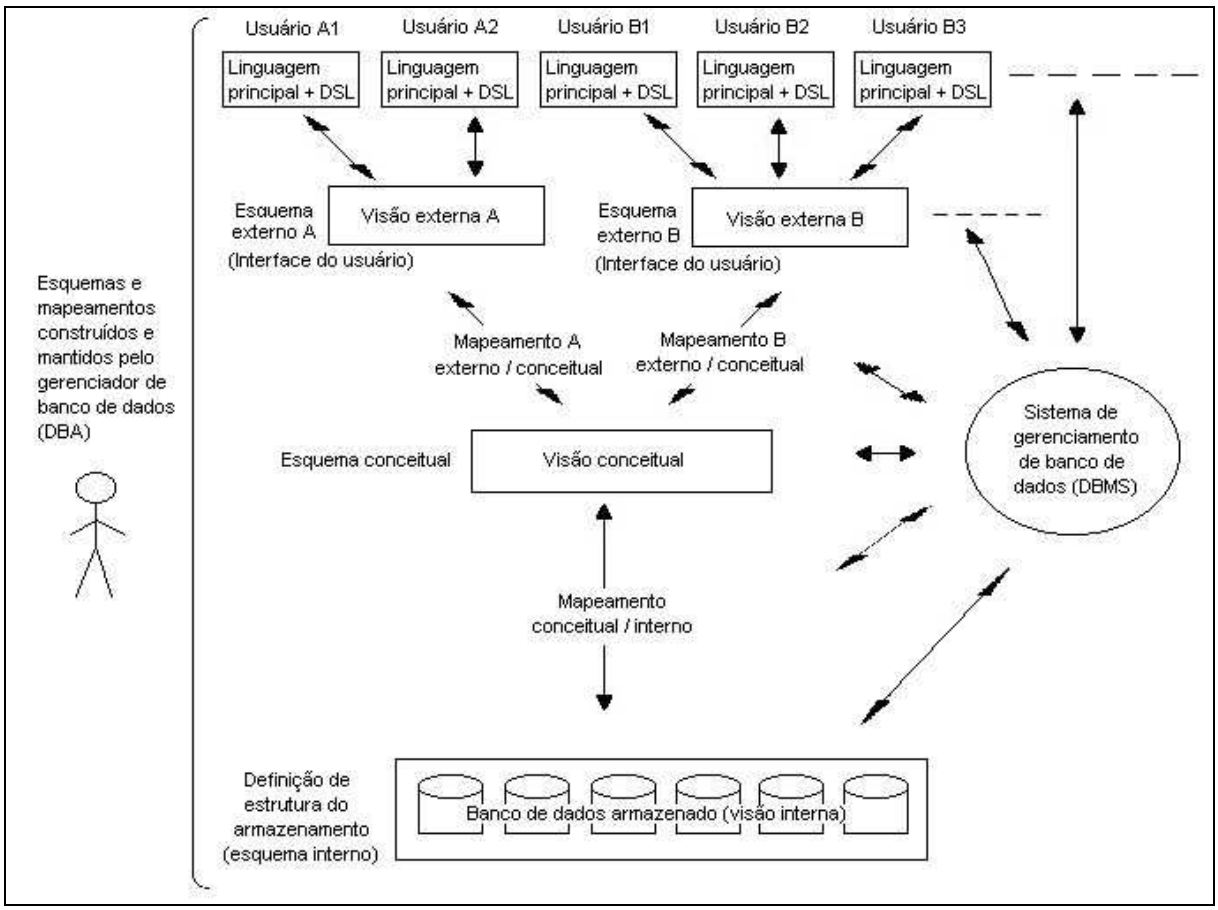
2.3 ARQUITETURA

Conforme Date (2000), a arquitetura de um SGBD divide-se em três níveis gerais: interno, conceitual e externo.

Nível interno é o mais próximo ao armazenamento físico. Relaciona-se à forma de como são realmente armazenados os dados. O nível externo é o mais próximo do usuário

final, o usuário que visualiza apenas o aplicativo. O nível conceitual é o "nível de simulação" entre os dois grupos. Também conhecido como nível lógico comunitário. Podem-se visualizar os níveis conforme a figura 2.

Figura 2 – ARQUITETURA DE UM SGBD



Fonte: adaptado de Date (2000)

A seguir, dar-se-á um enfoque maior ao nível interno, que é a forma pela qual os objetos físicos são criados, principal finalidade deste trabalho.

2.4 NÍVEL INTERNO

Segundo Date (2000), o nível interno é a maneira pela qual os dados são armazenados.

Não existe uma estrutura de armazenamento ótima que possa acolher todas as aplicações, sendo que pode-se ter várias estruturas de armazenamento diferentes, cada qual para suas determinadas aplicações.

A visão interna é uma representação de baixo nível dos bancos de dados. Consistem em ocorrências de registros armazenados (ou registros internos). A visão interna ainda está longe do nível físico, pois não lida com registros físicos, chamados de blocos ou páginas (Date, 2000).

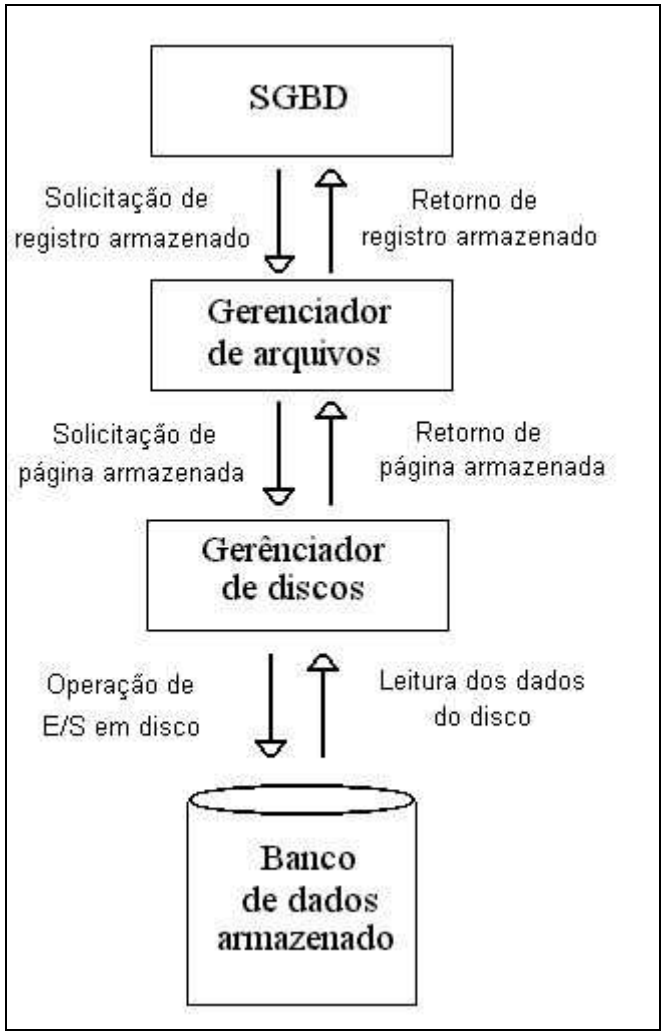
A visão interna é descrita por meio do esquema interno que especifica os tipos de registros armazenados, os índices existentes, como os campos estão representados e em que seqüências estão armazenados.

Segundo Date (2000), os programas aplicativos podem, em situações excepcionais, ter permissão para acessar diretamente no nível interno. Esta prática não é recomendada, pois afeta a segurança e pode vir a prejudicar a integridade dos dados. Porém, às vezes essa é a única maneira de conseguir a funcionalidade e desempenho exigido, da mesma maneira que um programador de uma linguagem de alto nível pode descer ao nível de escrever códigos em *assembler* (linguagem próxima da linguagem de máquina).

2.5 ACESSO AO BANCO DE DADOS

O acesso ao BD físico acontece como demonstrado na figura 3. Estes quatro componentes são essenciais para a leitura do registro armazenado.

Figura 3 – ACESSO AO BANCO DE DADOS



Fonte: adaptado de Date (2000)

A figura 3 apresenta de forma simplificada o processo de acesso aos bancos de dados em geral. Primeiramente o SGBD decide que o registro armazenado é necessário, pedindo a recuperação do dado ao gerenciador de arquivos. Este por sua vez verifica em que página o registro está armazenado e solicita ao gerenciador de discos que recupere a página. Então o gerenciador de discos determina a localização física da página e emite a operação de leitura (E/S).

A seguir, explicar-se-á sobre os tipos de índices, que são estruturas que melhoram a performance das consultas e acesso aos bancos de dados.

2.6 ÍNDICES

Os índices são estruturas de dados associados à tabela cuja principal finalidade é diminuir o tempo de acesso e recuperação dos dados. Uma tabela pode ter um ou mais índices associados a ela, sendo que o índice não modifica a forma de escrita do comando SQL, apenas aumenta a velocidade de recuperação e acesso aos dados (Ramalho, 1999).

Conforme Ramalho (1999), “Se por um lado, um índice agiliza o acesso às consultas, operações de *insert*, *delete* e *update* podem se tornar mais lentas se existirem muitos índices associados à tabela”.

Segundo Fernandes (2000), a criação de índices adicionais em uma tabela pode trazer tanto benefícios, quanto prejuízos, pois toda vez que é inserido conteúdo na tabela, o SGBD tem que atualizar a entrada do mesmo na sua tabela de índices.

Para Date (2000), a vantagem do índice é a aceleração na recuperação dos dados, porém há também uma redução na velocidade das atualizações. Toda vez que um novo registro é armazenado, uma nova entrada terá que ser colocada na indexação.

Segundo Date (2000), os índices podem ser seqüenciais ou diretos.

a) seqüencial: pode ser útil em consultas de limites. (Exemplo: letras no limite de L – R);
b) direto: útil nas consultas de listas (Exemplo: cidade de Blumenau).

Segundo Ramalho (1999), quanto menos dados a tabela conter, menos índices as tabelas devem ter, pois o tempo perdido para achar o índice a usar é mais custoso do que fazer um *full scan*, ou seja, uma consulta completa à tabela.

Segundo Date (2000), os bancos de dados relacionais suportam vários tipos de índices. O mais comum e mais utilizado por todos é o tipo árvore *B*Tree*. As árvores B são um conjunto de índices que apontam para um conjunto de índices. Os ramos superiores contêm dados que apontam para um nível mais baixo que contém índices até chegar ao nível que contenham os dados. Em teoria, estes níveis são ilimitados. Na prática, utiliza-se até 3 níveis.

Como saber que colunas indexar? Existem alguns itens que são relevantes na hora de escolher a criação de um índice. Alguns deles são:

a) campos muito usados em seleções de filtro são bons candidatos a serem indexados;

b) colunas que possam conter muitos valores nulos não são boas para serem indexadas;
c) colunas que contenham muitos valores iguais também não são boas para serem indexadas;
d) colunas das chaves primárias devem ser indexadas. Normalmente já é definido um índice de *primary key* na criação da tabela;
e) colunas que são chaves estrangeiras também são boas candidatas a serem indexadas;

Pode-se também construir índices por uma combinação de dois ou mais campos. São úteis quando as consultas são feitas por mais de um campo. Por exemplo, os professores da cidade de Blumenau com idade maior que 30.

2.6.1 ÍNDICES COMPOSTOS

Os índices podem ser compostos por mais de um campo. Isto é útil quando o filtro utilizado é composto não só por um único campo, de maneira que o SGBD possa localizar o conteúdo com uma única exploração de índice, e não com duas ou mais explorações de índices.

Isso melhora ainda mais a performance, pois se são feitos dois índices separados, é difícil decidir qual índice a ser explorado em primeiro lugar, uma vez que as duas seqüências possíveis podem ter características muito diferentes em desempenho (Date, 2000).

2.7 LINGUAGEM ESTRUTURADA

Em junho de 1970, o Dr. E. F. Codd publicou um artigo intitulado "*A Relational Model of Data for Large Shared Data Banks*". Este artigo determinou o inicio da mudança na filosofia de banco de dados, até então hierárquicos ou de rede. A IBM *Corporation, Inc.* , desenvolveu uma linguagem para usar o modelo imaginado por Codd. Esta linguagem foi batizada de SEQUEL (*Structured English Query Language*) e posteriormente apenas SQL. Hoje, a linguagem SQL é aceita como um padrão para os bancos de dados relacionais (Fernandes 2000).

A linguagem SQL divide-se em 3 grandes grupos:

a) DML: Linguagem de Manipulação de Dados (*Data Manipulation Language*). Os comandos DML, destinados a consultas, inserções, exclusões e alterações em um ou

mais registros de uma ou mais tabelas de maneira simultânea. Como exemplo de comandos DML pode-se citar os comandos *select*, *insert*, *update* e *delete*;

b) DCL: Linguagem de Controle de Dados (*Data Control Language*), dispõe de comandos de controle como *Grant* e *Revoke*;

c) DDL: Linguagem de Definição de Dados (*Data Definition Language*). É a parte de definição física das tabelas e suas características.

Algumas características da linguagem SQL:

a) capacidade de gerenciar índices, sem a necessidade de controle individualizado de índice corrente, algo muito comum nas linguagens de manipulação de dados do tipo registro a registro;

b) construção de visões, que são formas de visualizarmos os dados na forma de listagens independente das tabelas e organização lógica dos dados;

c) cancelamento de uma série de atualizações ou de gravações. Os comandos *commit* (confirmar) e *rollback* (voltar/cancelar) são responsáveis por estas facilidades.

A linguagem SQL consegue implementar estas soluções somente pelo fato de estar baseada em BD, que garantem por si mesmo a integridade das relações existentes entre as tabelas e seus índices.

A seguir Apresentar-se-á linguagem DDL, que é a forma cujos objetos são criados nos *scripts* gerados pelas ferramentas CASE.

2.7.1 LINGUAGEM DE DEFINIÇÃO DE DADOS

Segundo Fernandes (2000), esta é a parte do SQL que descreve como as tabelas e outros objetos podem ser definidos, alterados e removidos.

Alguns dos principais comandos DDL são:

a) *CREATE* / *DROP TABLE*: cria e apaga tabelas na base de dados;

b) *ALTER TABLE*: altera a definição de uma tabela, podendo-se adicionar ou remover campos, alterar seus tamanhos etc;

c) *CREATE* / *DROP INDEX*: cria e apaga índices nas tabelas;

d) *CREATE* / *DROP VIEW*: cria e apaga visões. Estas visões podem ser resultados de pesquisas em outras tabelas ou mesmo uma tabela específica.

Segundo Ramalho (1999), os campos que armazenam muitos valores nulos deveriam ser especificados no final da definição do comando de criação da tabela (*CREATE TABLE*), de maneira que ocupem menos espaço no banco de dados.

Na próxima etapa, serão mostrados os bancos de dados estudados, suas características e conceitos. Iniciando-se com o banco de dados *Oracle* e seguindo com o *Microsoft SQL Server*.

2.8 ORACLE

Segundo Ramalho (1999), o *Oracle* é um sistema de gerenciamento de banco de dados relacional que, além do banco de dados, possui uma instância de servidor *Oracle*. O *Oracle* possui duas estruturas, a física e a lógica. Elas são separadas, podendo-se gerenciar o armazenamento físico sem afetar o acesso lógico de armazenamento.

No *Oracle*, a maior parte dos acessos ao banco de dados é feita na linguagem SQL. Apesar de muitos dos produtos *Oracle* apresentarem uma interface que aparentemente não utiliza a linguagem, o que fazem, na verdade, é converter as solicitações dos usuários em comandos de SQL no relacionamento com o banco de dados (Fernandes 2000).

A estrutura física está armazenada em três tipos de arquivos: um ou mais *datafiles*, dois ou mais arquivos de registros de redo, que são os arquivos onde o *Oracle* guarda informações de *log*, e um ou mais arquivos de controle, que armazenam informações de parâmetros de inicialização, quais as estruturas que existem e como estas estão associadas.

A estrutura lógica é determinada por um ou mais *tablespaces*, pelos segmentos, extensões e pelos objetos de esquemas. Um esquema é uma coleção de objetos. Os objetos de esquemas podem ser tabelas, visões, seqüências, índices entre outros.

Os objetos de esquema e os relacionamentos entre eles formam o projeto relacional de um banco de dados.

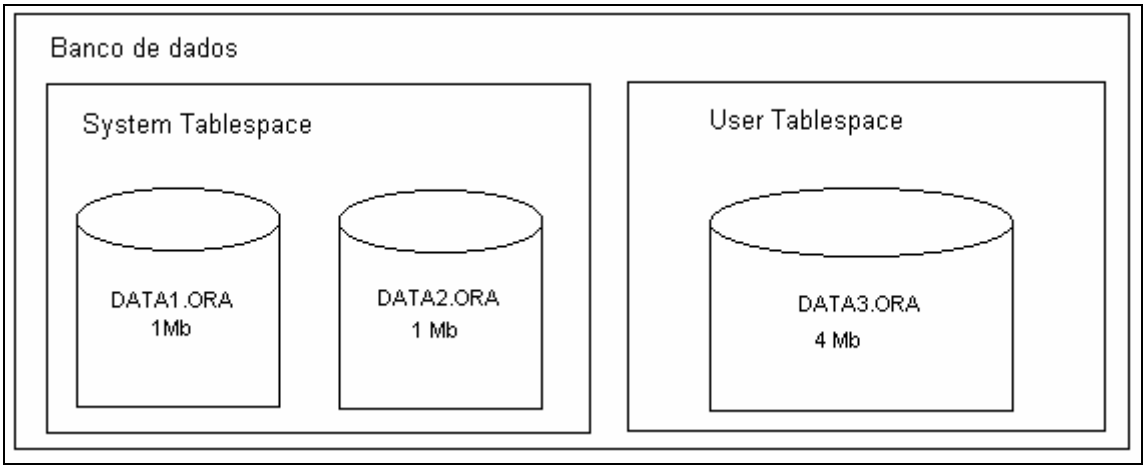
Segundo Ramalho (1999), toda vez que um banco de dados é iniciado, uma área global do sistema (SGA) é alocada e os processos de segundo plano do *Oracle* são inicializados. A combinação destes processos com os *buffers* de memória são chamados de instâncias.

A seguir serão apresentadas as formas de armazenamento.

2.8.1 TABLESPACES E DATAFILES

O *tablespace* corresponde à unidade lógica de armazenamento. É dentro dele que armazenamos as tabelas, índices, dicionários de dados etc. Um *tablespace* é composto por diversas partes do mesmo tamanho. Estas partes são chamadas de blocos.

Figura 4 – *TABLESPACE* E *DATAFILES*



Fonte: Ramalho (1999)

Os *datafiles* são os arquivos físicos onde são armazenados os dados, tais como índices e tabelas. As características dos *datafiles* são:

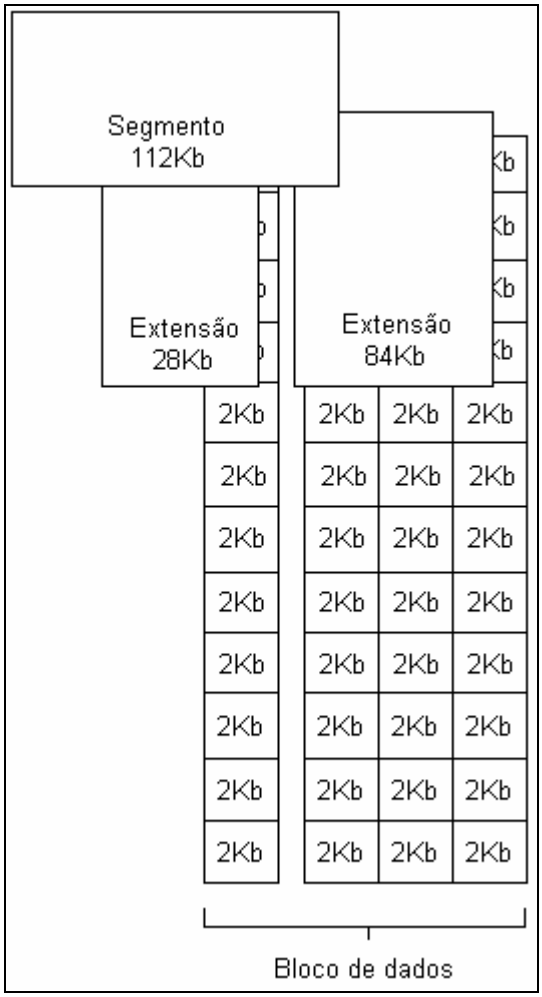
a) pode ser associado a apenas um banco de dados;
b) os arquivos de banco de dados podem ter características definidas para que eles cresçam automaticamente caso fiquem sem espaço;
c) um ou mais *datafiles* formam um *tablespace*.

De maneira simples, os *datafiles* são a parte mais interna e mais próxima do nível físico.

2.8.2 BLOCOS, EXTENSÕES E SEGMENTOS

Conforme Ramalho (1999), o bloco de dados corresponde aos números específicos de *bytes* armazenados no disco. É a menor forma de representação dos dados no *Oracle*.

Figura 5 – BLOCO DE DADOS, EXTENSÕES E SEGMENTOS



Fonte: Ramalho (1999)

O tamanho do bloco é definido pelo DBA na criação do banco de dados através de configurações iniciais. Este bloco será a unidade mínima de leitura e gravação para disco. Toda vez que se deseja obter informações de uma determinada linha de uma tabela, no mínimo far-se-á solicitação de leitura de um bloco (Fernandes 2000).

Uma extensão é uma área contínua para armazenamento de informações. É um conjunto de blocos de dados contínuos, usado para armazenar determinados tipos de informações (Ramalho, 1999).

A primeira extensão alocada tem o nome especial de *initial*. As demais são chamadas de *next*. Por questões de performance, considera-se ideal que todos os dados de uma determinada tabela estivessem armazenados na extensão inicial. Como isto nem sempre é

possível, deve-se, na especificação de um segmento qualquer, eliminar o tamanho das extensões *initial* e *next* e a quantidade que um segmento pode esticar, anexando novas extensões à sua área útil (Fernandes 2000).

Se os blocos de dados da extensão inicial de um segmento ficarem cheios, automaticamente será alocada uma nova extensão para aquele segmento, com o tamanho igual ou superior àquele anterior.

O nível de armazenamento lógico que está acima das extensões é o segmento. É um conjunto de extensões alocado para determinar uma estrutura lógica. Os tipos de segmentos podem incluir os segmentos de dados, de índices de *rollback* e o segmento temporário.

Segundo Ramalho (1999), os parâmetros *PCTFREE* e *PCTUSED* utilizados na criação de uma tabela, são importantes para o gerenciamento e controle de espaço. O *PCTFREE* indica a quantidade de espaço que deve ser deixada para a atualização de linhas do bloco. O parâmetro *PCTUSED* ajusta a porcentagem mínima que pode ser usada para os dados de uma linha antes que novas linhas sejam adicionadas ao bloco.

2.8.3 CRIAÇÃO DAS TABELAS

A criação da tabela deve ser precedida de uma análise e uma total definição da estrutura e da funcionalidade do banco de dados. Na criação da tabela é indicada a criação das colunas, definida a organização da tabela, *constraints*, *tablespace*, características de armazenamento entre outras informações.

A criação das tabelas é definida pelo comando *create table*. A sintaxe do comando pode ser visualizada no quadro 1.

Quadro 1 – CREATE TABLE ORACLE

CREATE TABLE **esquema.tabela** (coluna1 **tipo_de_dado**, colunaN **tipo_de_dado**, primary key campo) **PCTFREE** integer **PCTUSED** integer **STORAGE** (INITIAL 128K NEXT 128K MINEXTENTS 1 MAXEXTENTS 4096 PCTINCREASE 0) **TABLESPACE** tablespace **LOB** (Lob_1, LobN) STORE AS lob_segment **INDEX** tablespace

Os parâmetros da criação são:

a) esquema: opcional que indica em que base será criada a tabela;

b) tabela: nome da tabela;
c) tipo_de_dado: especifica o tipo da coluna. No *Oracle* são permitidas até 254 colunas;
d) *pctfree*: é opcional e tem valor padrão 10. Indica o percentual de cada *data block* que será reservado para futuras atualizações da tabela;
e) *pctused*: é opcional e tem valor padrão 40. Indica o percentual mínimo de espaço usado que o *Oracle* deve manter antes de um *data block* se tornar candidato a uma inserção de linha;
f) *storage:* indica as características de armazenamento da tabela. Esta cláusula deve ser especificada para minimizar a alocação de espaço complementar. O parâmetro *initial* é extraído a partir da informação da quntidade inicial de registros da tabela, sendo que o parâmetro *next* é obtido da quantidade de registros que esta tabela receberá. *Minextents* e *maxextents* são a quantidade mínima e máxima das extensões. *Pctincrease* é o percentual de incremento destes segmentos.
g) *tablespace*: é opcional e indica em que *tablespace* a tabela deverá ser armazenada;
h) *lob*: é opcional e indica onde os campos *LOB*, que são os campos que armazenam textos grandes e figuras (*blob*, *clob*, *long*), devem ser armazenados;
i) *index*: é opcional e indica onde os índices devem ser armazenados;

No comando de criação da tabela, pode-se especificar um *tablespace* diferente para índices e para os campos do tipo *lob*. Isto serve para otimizar o desempenho, pois os mesmos ficam em *tablespaces* com definições especificas. É recomendação da *Oracle* que se coloque os índices e *lobs* em *tablespaces* separados para um ganho de performance.

2.8.4 TIPOS DE DADOS

Segundo Ramalho (1999), o *Oracle* suporta diversos tipos de dados em suas tabelas. Os tipos são:

a) *bfile*: contém um localizador para um arquivo binário externo. O tamanho máximo é de 4 Gb;
b) *blob*: objeto binário de tamanho máximo 4 Gb;
c) *char*(tamanho): campo fixo com tamanho máximo de 2000 bytes;
d) *clob*: objeto binário contendo caracteres do tipo *single byte.* Tamanho máximo de 4

Gb;

e) *date*: data válida entre 1,4712 a.C. até 31,4712;

f) *long*: caractere variável com tamanho de até 2 Gb;

g) *long raw*: variável do tipo *raw binary* com tamanho de até 2 Gb;

h) *nchar*(tamanho): dado de tamanho fixo do tipo caractere com tamanho máximo de 2000 bytes;

i) *nclob*: objeto do tipo caractere contendo caracteres do tipo *multibyte*. Possui tamanho de até 4 Gb;

j) *nvarchar2*(tamanho): dado de caractere de tamanho variável de até 4000 bytes;

k) *number*(p,s): dado do tipo numero com precisão *p* e escala *s*. a precisão pode variar de 1 a 38 e a escala de –84 a +127;

l) *raw*: dado do tipo *raw binary* com tamanho máximo de 2000 bytes;

m) *rowid*: string hexadecimal que representa o endereço único de uma linha na tabela;

n) *mlslabel*: formato binário de um rotulo do sistema operacional. Atualmente em desuso;

o) *varchar2*(tamanho): dado do tipo caractere com tamanho variável e limitado em 4000 bytes;

p) *numeric*(p,s): é idêntico ao *number* (p,s);

q) *decimal*(p,s): é idêntico ao *number*(p,x);

r) *integer*: é idêntico ao *number*(38)

s) *int*: é idêntico ao *integer*;

t) *smallint*: é idêntico ao *integer*;

u) *float*(b), *double precision* e *real* : são idênticos ao *number*;

A seguir descreve-se as informações sobre o outro banco estudado, o banco de dados da *Microsoft.*

2.9 MICROSOFT SQL SERVER

O *Microsoft SQL Server* (MSSQL) é um banco de dados relacional, que tem em sua estrutura um conjunto de *databases* para administração do banco. São os chamados *system databases* que são:

a) *master* : guarda as informações de todo o sistema. Armazena os *logins* dos usuários

e todas as configurações do banco. Possui também as informações de todas as outras bases de dados dos usuários e a localização das mesmas;

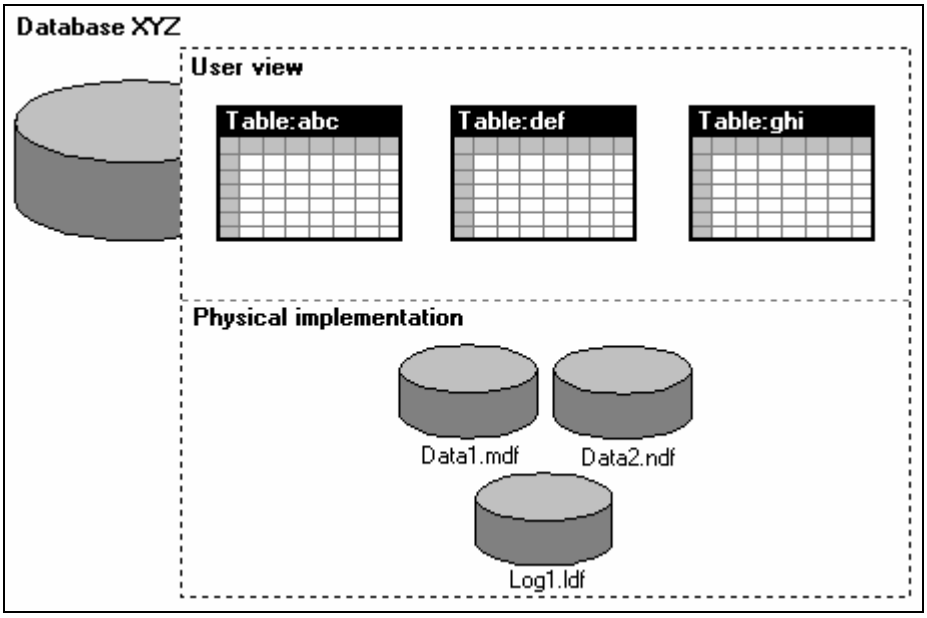
b) *tempdb*: armazenas as informações temporárias sobre as tabelas dos usuários e suas possíveis *stored procedures.* O *tempdb* é um recurso global, visto por todos os usuários de todas as bases de dados. É recriada toda vez que o MSSQL é iniciado;
c) *model*: serve de modelo para as bases de dados criadas;
d) *msdb*: guarda as informações de agendamento e serviços, como por exemplo, o agendamento de *backups* automáticos.

Segundo Microsoft (1998), as bases de dados são armazenadas fisicamente em dois ou três arquivos, dependendo da especificação na criação da base. Eles são:

a) arquivo de dados primário: de extensão mdf, contêm informações sobre o database e todas os dados das tabelas. Toda base de dados possui um arquivo primário;
b) arquivo de dados secundário: de extensão ndf, pode ser usado para alocar espaço em outros discos. Uma base de dados não precisa necessariamente de um arquivo de dados secundário;
c) arquivo de log: de extensão ldf, armazena as informações de log usados para recuperar o database caso necessário. Todas as bases precisam ter pelo menos um arquivo de log. O seu tamanho mínimo é de 512 Kb.

Pode-se definir então a estrutura do MSSQL conforme a figura 6.

Figura 6 – ESTRUTURA DO *MSSQL SERVER*



Fonte: Microsoft 1998

O MSSQL possui uma maneira de armazenar os dados fisicamente de forma transparente para os usuários. Fisicamente, elas são armazenadas em páginas e extensões. A seguir apresenta-se mais detalhes desta arquitetura.

2.9.1 PÁGINAS E EXTENSÕES

A unidade fundamental de armazenamento de dados é a página. O tamanho das páginas é de 8Kb. Os primeiros 96 *bytes* são do cabeçalho e servem para armazenar informações de sistema, como por exemplo, o tamanho de espaço livre na página. As linhas de dados são postas logo abaixo da linha de cabeçalho de forma serial. No final da página é armazenada a tabela de *row offset* que armazena a entrada de cada linha na página e qual o seu tamanho.

Os campos do tipo *text*, *ntext*, e *image* não são armazenados na mesma página dos dados. Para tanto, existem diversos tipos de páginas, sendo elas:

a) *data*: página de dados, exceto os tipo *text*, *ntext* e *image*;

b) *index*: possui as entradas dos índices;

c) *text*, *image*: armazena os dados dos tipos *text*, *ntext* e *image*;

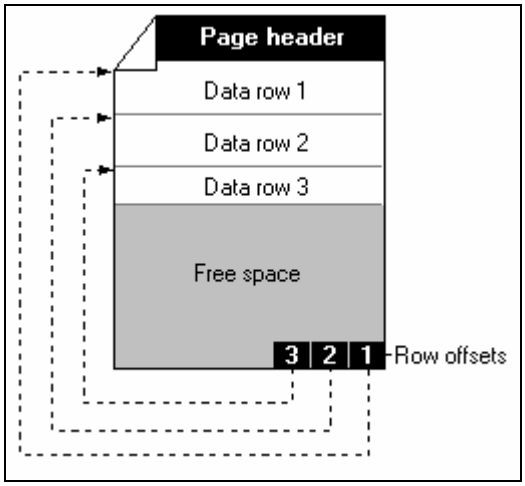
d) *global allocation map*: informações sobre as extensões alocadas;

e) *page free space*: dados sobre os espaços livre nas páginas;

f) *index allocation map*: informações sobre extensões utilizadas por tabelas ou índices.

A figura 7 demonstra a estrutura da página no MSSQL.

Figura 7: PÁGINA



Fonte: Microsoft 1998

Extensão é a unidade básica de armazenamento das tabelas e dos índices. A extensão é um conjunto de 8 páginas. Para ter uma alocação de dados eficiente, o MSSQL não aloca entradas nas extensões para tabelas pequenas.

As extensões podem ser organizadas de duas maneiras:

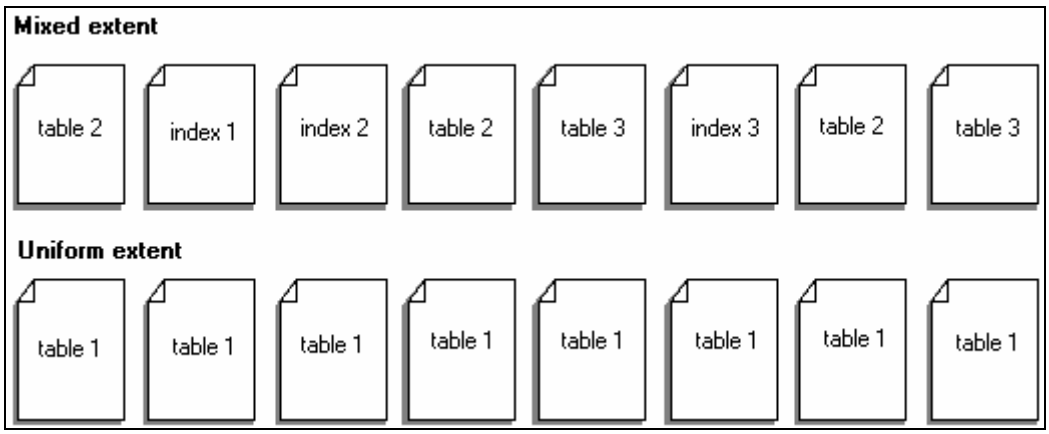
a) extensão uniforme: possui somente um criador e todas as 8 páginas podem ser usadas somente por ele;

b) extensão misturada: as suas oito páginas são compartilhadas para qualquer base.

As páginas e índices são alocados em extensões misturadas, mas quando a tabela ou o índice cresce demais a ponto de necessitar 8 páginas, elas são trocadas para uma extensão uniforme.

O MSSQL pode ter até 128 páginas e 16 extensões por *megabyte*. Na figura 8 é demonstrada a diferença entre os tipos de extensões.

Figura 8: EXTENSÕES



Fonte: Microsoft 1998

A seguir apresentar-se-á definição de *filegroups* de uma base de dados.

2.9.2 FILEGROUPS

Os arquivos de base de dados podem ser alocados em diferentes grupos de arquivos, para diversas finalidades como alocação de espaço e administração das bases.

Alguns sistemas podem melhorar a performance controlando a localização de datas e índices em discos específicos. Os *filegroups* ajudam neste processo. O administrador do sistema pode criar *filegroups* em diferentes discos e associar as tabelas e os campos do tipo *text*, *ntext*, or *image* nestes diferentes grupos.

Existem 3 diferentes grupos:

a) primário: contém o primeiro arquivo de dados da base e todos os outros arquivos que não tiveram um *filegroup* especificado. Todas as páginas para as tabelas do sistema são alocadas neste *filegroup*;
b) grupo definido pelo administrador: são os especificados na criação da base, e podem ser utilizados pelos objetos na especificação da criação do mesmo;
c) padrão: contém as páginas para todas as tabelas e índices que não tenham um *filegroup* especificado. Se nenhum *filegroup* padrão for especificado na hora de criação da base, o *filegroup* primário será assumido como padrão.

A seguir analisar-se-á a forma de criação das tabelas, que é a principal forma de armazenamento de dados nos bancos de dados.

2.9.3 CRIAÇÃO DAS TABELAS

As tabelas no MSSQL podem ser criadas através das ferramentas específicas do banco, como o *Enterprise Manager* ou através de comandos DDL. A sintaxe de criação das tabelas através de DDL é definida conforme o quadro 2.

Quadro 2 – CREATE TABLE MSSQL

CREATE TABLE [<nome da base>.proprietário.| proprietário.] <nome da tabela> ({<definição das colunas>} [,...n]|<definição da chave primária>) [ON {filegroup | DEFAULT}] [TEXTIMAGE_ON {filegroup | DEFAULT}]

Os parâmetros da criação são:

a) <nome da base>: especificar em que base de dados a tabela deve ser criada;
b) proprietário: usuário proprietário da base;
c) <nome da tabela>: nome da tabela criada;
d) <definição das colunas>: indicação das colunas que a base de dados terá. Inclui o nome da coluna, o tipo de campo e se permite ou não nulo;
e) <definição da chave primária>: definição da chave primária da tabela.
f) *on filegroup*: em que *filegroup* será especificada a tabela. Opcional. Se nenhum for informado, o padrão será assumido;
g) *textimage_on filegroup*: em que *filegroup* será especificado os campos do tipo *lob*. Opcional. Se nenhum for informado, o padrão será assumido.

2.9.4 TIPOS DE DADOS

Uma tabela pode ter diversos tipos de dados. Segundo Imaster (2002), os tipos de dados que podem existir em uma tabela no MSSQL versão 7.0 são:

a) *tinyint*: valores numéricos inteiros variando de 0 até 256;
b) *smallint*: valores numéricos inteiros variando de –32.768 até 32.767;
c) *int*: valores numéricos inteiros variando de -2.147.483.648 até 2.147.483.647;
d) *bit*: somente pode assumir os valores 0 ou 1. Utilizado para armazenar valores lógicos;

e) *decimal*(I,D) e *numeric*(I,D): armazenam valores numéricos inteiros com casas decimais utilizando precisão. I deve ser substituído pela quantidade de dígitos total do número e D deve ser substituído pela quantidade de dígitos da parte decimal (após a vírgula). *Decimal* e *numeric* possuem a mesma funcionalidade, porém *decimal* faz parte do padrão *ANSI* e o *numeric* é mantido por compatibilidade. Por exemplo, *decimal*(8,2) armazena valores numéricos decimais variando de 999999,99 até -999999,99;

f) *smallmoney*: valores numéricos decimais variando de -214.748,3648 até 214.748,3647;

g) *money*: valores numéricos decimais variando de -922.337.203.685.477,5808 até 922.337.203.685.477,5807;

h) *real*: valores numéricos aproximados com precisão de ponto flutuante, indo de -3.40E + 38 até 3.40E + 38;

i) *float*: valores numéricos aproximados com precisão de ponto flutuante, indo de -1.79E + 308 até 1.79E + 308;

j) *smalldatetime*: armazena hora e data variando de 1 de janeiro de 1900 até 6 de junho de 2079. A precisão de hora é armazenada até os segundos;

k) *datetime*: armazena hora e data variando de 1 de janeiro de 1753 até 31 de Dezembro de 9999. A precisão de hora é armazenada até os centésimos de segundos;

l) *char*(N): armazena N caracteres fixos (até 8.000) no formato não *unicode*. Se a quantidade de caracteres armazenada no campo for menor que o tamanho total especificado em N, o resto do campo é preenchido com espaços em branco;

m) *varchar*(N): armazena N caracteres (até 8.000) no formato não *unicode*. Se a quantidade de caracteres armazenada no campo for menor que o tamanho total especificado em N, o resto do campo não é preenchido;

n) *text*: armazena caracteres (até 2.147.483.647) no formato não *unicode*. Se a quantidade de caracteres armazenada no campo for menor que 2.147.483.647, o resto do campo não é preenchido. Procure não utilizar este tipo de dado diretamente, pois existem funções específicas para trabalhar com este tipo de dado;

o) *nchar*(N): armazena N caracteres fixos (até 4.000) no formato *unicode*. Se a

quantidade de caracteres armazenada no campo for menor que o tamanho total especificado em N, o resto do campo é preenchido com espaços em branco;

p) *nvarchar*(N): armazena N caracteres (até 4.000) no formato *unicode*. Se a quantidade de caracteres armazenada no campo for menor que o tamanho total especificado em N, o resto do campo não é preenchido;

q) *ntext*: armazena caracteres (até 1.073.741.823) no formato *unicode*. Se a quantidade de caracteres armazenada no campo for menor que 1.073.741.823, o resto do campo não é preenchido. Procure não utilizar este tipo de dado diretamente, pois existem funções específicas para trabalhar com este tipo de dado;

r) image: variável binária com tamanho de 2^31 - 1 (ate 2,147,483,647) bytes.

Com isto, concluí-se o estudo dos bancos de dados *Oracle* e *Microsoft SQL Server*. No próximo capítulo, enfocar-se-á inteligência artificial. A inteligência artificial foi a técnica escolhida para o desenvolvimento do protótipo. Dar-se-á uma atenção especial aos sistemas especialistas na forma de regras de produção.

3 INTELIGÊNCIA ARTIFICIAL

Segundo Rabusque (*apud* Rautenberg (1996)), “Inteligência é definida como a capacidade geral de um indivíduo consciente ajustar seu pensar a novas exigências, ou seja, é a adaptabilidade mental geral a novos problemas e condições. Também pode-se dizer que a inteligência é o processo contínuo de aquisição, de triagem, de ordenação e de interpretação da informação”.

Segundo Rautenberg (1996), na computação, o estudo do comportamento inteligente é chamado de Inteligência Artificial (IA). IA é a parte da informática que visa equipar os computadores com raciocínio e capacidade perceptivas.

A IA pode ser usada em diversas aplicações, sendo elas em processamento de linguagem natural, no reconhecimento de padrões, na robótica, em base de dados inteligentes e em sistemas especialistas.

3.1 SISTEMAS ESPECIALISTAS

Segundo Genaro (1986), sistemas especialistas são programas que possuem e manipulam o conhecimento da mesma forma que o especialista humano. Eles utilizam a lógica e regras encontradas na prática para encontrar as soluções para os problemas, sendo que tais sistemas também podem errar e aprender com os erros.

Para Kamke (2001), sistemas especialistas são programas que procuram achar soluções para os problemas apresentados da mesma forma que um humano resolveria estando sobre as mesmas condições.

No início da década de 1960, começaram os primeiros trabalhos em sistemas especialistas. Pretendia-se construir máquinas com poder de raciocinar e solucionar problemas de forma inteligente (Heinzle, 1995).

Conforme explicação de Genaro (1986), algumas das características mais comuns dos sistemas especialistas são:

a) resolvem problemas tão bem quanto os especialistas humanos;
b) raciocinam heuristicamente;
c) interagem com usuários humanos;
d) manipulam e raciocinam sobre decisões simbólicas;

e) funcionam com dados e regras incertas ou erradas;
f) contemplam hipóteses múltiplas;
g) explicam o por que das perguntas;
h) demonstram e explicam suas conclusões.

Conforme Ribeiro (1987), “um sistema especialista é aquele que é projetado e desenvolvido para atender a uma aplicação determinada e limitada do conhecimento humano. É capaz de emitir uma decisão com o apoio em conhecimento justificado a partir de uma base de informações, tal qual um especialista de determinada área do conhecimento humano”.

3.2 COMPONENTES DO SISTEMA ESPECIALISTA

Segundo Kamke (2001), a generalidade, os objetivos, a representação interna e as ferramentas utilizadas são fatores para a composição de um sistema especialista. De forma geral, o sistema é constituído de seis elementos básicos: base de conhecimento, mecanismo de aprendizagem e aquisição de conhecimento, máquina ou motor de inferência, sistema de justificação, sistema de consulta e quadro negro.

3.2.1 BASE DO CONHECIMENTO

Conforme Rautenberg (1996), a base de conhecimento armazena os dados e pode ser representado sob forma de regras de produção, redes semânticas ou frames.

Segundo Heinzle (1995), a base de conhecimento é o local onde se armazenam os fatos, heurísticas, etc. É um local onde se armazena o conhecimento de um determinado assunto, que permite ao sistema especialista fazer o processamento e a consulta das informações para o processamento. O sistema deve possuir uma base flexível para ser facilmente atualizada, pois a qualidade da informação é um fator determinante no potencial do sistema especialista.

O processo de aquisição do conhecimento é um dos fatores mais importantes da construção da base de conhecimento, visto que o conhecimento humano não se encontra formalizado (Heinzle, 1995).

3.2.2 AQUISIÇÃO DO CONHECIMENTO

Para Rabuske (1995), a aquisição do conhecimento é a parte mais complexa da construção do sistema especialista.

É a parte do sistema que permite ampliar e modificar a base de conhecimento. Em muitos sistemas é a única forma de aquisição de conhecimento.

Segundo Rautenberg (1996), na maioria dos casos, são editores simples que alteram a base de conhecimento. Ainda há a possibilidade de tornar este mecanismo mais inteligente, fazendo com que tenha uma ordenação de prioridades, e melhore ainda mais a qualidade das respostas, sendo este um recurso optativo.

Para Levine (1988), “A fase de aquisição de conhecimento é a que representa maior dificuldade na construção de um sistema especialista. Esta dificuldade advém do fato de não existir uma linguagem comum de entendimento entre as partes envolvidas no projeto”.

3.2.3 MOTOR DE INFERÊNCIA

Os dados existentes na base de conhecimento são estáticos e sem muito valor se não forem tratados para obter então informações que sirvam para guiar o usuário. O mecanismo que trata e busca estes dados é chamado de motor, máquina ou ainda engenho de inferência.

Segundo Heinzle (1995), o motor de inferência é o elemento do sistema que busca na base o conhecimento e avalia cada situação, direcionando o processo de raciocínio, gerenciando situações de incerteza e levando ao resultado final.

O motor de inferência contém um interpretador que decide como aplicar as regras que estão na base de conhecimento e um seqüenciador que decide a ordem em que se devem seguir as regras.

3.2.4 SISTEMA DE JUSTIFICAÇÃO

Segundo Rautenberg (1996), a justificação é um requisito obrigatório, tendo, geralmente, capacidade de responder as perguntas:

a) como chegou a esta conclusão?
b) por que chegou a esta conclusão?
c) por que não chegou a tal conclusão?

3.2.5 SISTEMA DE CONSULTA

Sistema de consulta é a forma de interação do usuário do sistema. Estes participam ativamente do processo de inferência na base de conhecimento. O sistema de consulta deve ser bem simples e bem explicativo, de forma que atenda até os usuários sem conhecimento computacional. A terminologia computacional deve ser evitada e detalhes técnicos relativos ao sistema devem ser transparentes aos usuários. A linguagem utilizada deve ser a mais próxima possível da linguagem natural (Kamke, 2001).

Deve-se observar que o usuário não participou da criação e elaboração do sistema, portanto é natural que ele não tenha o conhecimento das formas de representação do conhecimento adotadas. (Rautenberg, 1996).

3.2.6 QUADRO NEGRO

Segundo Rabuske (1995), o quadro negro é a área de trabalho do sistema especialista. Embora todos os sistemas especialistas usem o quadro negro, nem todos o têm como um componente explícito. O quadro negro é o rascunho, uma área de memória aonde o sistema vai gravando e apagando dados que vai usando no processo de inferência até chegar a uma conclusão.

A seguir, serão apresentadas as técnicas e ferramentas que foram utilizadas na elaboração deste trabalho.

4 TÉCNICAS E FERRAMENTAS UTILIZADAS

O protótipo foi implementado no ambiente de desenvolvimento *Delphi* 5.0, onde foram empregados os conceitos de orientação a objetos para o desenvolvimento das classes. Para a especificação foi utilizado a *Unified Modelling Language* (UML) através da ferramenta *Rational Rose*. As técnicas de regras de produção da inteligência artificial foram desenvolvidas em *Delphi*, inclusive com o desenvolvimento do próprio motor de inferência. Não foi utilizada nenhuma ferramenta CASE para o desenvolvimento da base de conhecimento nem do motor de inferência.

A analise dos *scripts* foi feita através de técnicas de compiladores. Segundo Price (2001) as gramáticas livres de contexto formam a base para a análise sintática das linguagens de programação, pois permitem especificar a maioria das linguagens de programação usadas atualmente. Sebesta (2000) complementa dizendo que a notação mais comumente utilizada para se escrever uma gramática livre de contexto é a Forma de Bakus-Naur (BNF).

Não foi feita uma especificação na BNF, pois a análise a ser feita não é demasiadamente complexa, sendo analisado apenas comandos de *CREATE TABLE*, *CREATE INDEX* e *ALTER TABLE.*

Os *scripts* são oriundos das ferramentas CASE *Power Designer* e *ER/Studio.*

4.1 INTELIGENCIA ARTIFICIAL

Para o desenvolvimento do protótipo foram utilizadas técnicas de IA, utilizando sistemas especialistas na forma de regras de produção. A base de conhecimento foi especificada em arquivos texto e o motor de inferência foi feito na ferramenta *Delphi*.

4.1.1 AQUISIÇÃO DO CONHECIMENTO

A aquisição do conhecimento ocorreu através de estudos do acadêmico sobre os bancos de dados *Oracle* e *Microsoft SQL Server*. O conhecimento também pode ser adicionado na base pelo próprio usuário do sistema, no módulo de edição da base de conhecimento no menu principal.

4.2 COMPILADORES

Para Aho (1995), um compilador é um programa que lê um programa escrito em uma linguagem e o traduz numa outra linguagem equivalente, denominada de linguagem-alvo.

A compilação é dividida em duas partes: a análise e a síntese. Na primeira o código é analisado e em seguida é criada uma representação intermediária do mesmo. Na síntese o compilador constrói o programa objeto desejado a partir da representação intermediária (Aho, 1995).

A análise consiste em três fases intermediárias:

a) análise léxica: um fluxo de caracteres constituindo um programa é lido da esquerda para a direita sendo agrupado em *tokens*;
b) análise sintática: os *tokens* são agrupados hierarquicamente em coleções aninhadas;
c) análise semântica: são feitas verificações que asseguram que os componentes de um programa se combinam de forma significativa.

A seguir, apresentar-se-á as fases de análise léxica e sintática, que serão as únicas utilizadas no trabalho, visto que o mesmo não transforma o *script* em outra linguagem.

4.2.1 ANALISADOR LÉXICO

Para José (1987), a análise léxica é aquela que faz a interface entre o texto-fonte e os programas encarregados de sua análise e tradução. Sua missão fundamental é a de, a partir do texto-fonte de entrada, fragmentá-lo em seus componentes básicos (chamados de partículas, átomos ou *tokens*), identificando trechos elementares completos e com identidade própria, porém individuais para efeito de análise por parte dos demais programas do compilador.

Identificadas as partículas do texto-fonte, estas devem ser classificadas segundo o tipo a que pertencem, uma vez que para o módulo da análise sintática, que deverá utilizá-las em seguida, a informação mais importante acerca destas partículas é a classe à qual pertencem, e não propriamente o seu valor (José, 1987).

4.2.2 ANALISADOR SINTÁTICO

Segundo Price (2001), a analise sintática é a segunda fase de um compilador. Sua função é verificar se as construções usadas no programa, ou no caso o *script*, estão gramaticalmente corretas.

4.3 DELPHI

O protótipo foi desenvolvido orientado a objetos, utilizando a ferramenta Delphi 5.

O *Delphi* não é uma ferramenta 100% visual, sendo que muito do código teve que ser implementado. Forma utilizados os componentes padrões do ambiente, sem necessidade de componentes produzido por terceiros.

4.4 FERRAMENTAS CASE

Segundo Gane (1990), a sigla CASE (*Computer Aided Software Engeneering*) significa Engenharia de Software Auxiliada por Computador. O termo foi criado no inicio da década de oitenta, quando percebeu-se que ferramentas gráficas de modelagem poderiam ser úteis em analise e projetos de sistema. Sabendo que engenheiros de outras áreas já utilizavam ferramentas gráficas para ajudar nos cálculos matemáticos de desenvolvimentos de desenhos (CAD - *Computer Aided Design*), acreditou-se que as ferramentas gráficas poderiam ser úteis também para os profissionais de desenvolvimentos de sistemas de informação.

O objetivo principal das ferramentas CASE´s é separar o projeto da implementação do projeto. Quanto mais afastadas estas duas etapas estiverem, melhor (Fisher, 1990).

Segundo Martin (1991), a tecnologia CASE é o futuro da computação. Nela os analistas projetam suas aplicações em uma tela e já é gerado o código executável. A máquina pode automatizar muitas coisas como validações e checagem de dados, ganhando muito tempo dos programadores e analistas. Os usuários finais precisam ter a capacidade de resolver seus próprios problemas utilizando computadores. Estes usuários incluem desde engenheiros que executam cálculos intrincados até empresários que tomam decisões complexas com o apoio de informações computadorizadas.

As ferramentas CASE foram utilizadas para gerar os *scripts* que foram submetidos ao protótipo para agregar comandos que possibilitem melhorar a qualidade de alocação de

espaço e a performance de busca das informações. Os *scripts* foram gerados pelos CASE *Power Designer* da empresa *Sybase* e *ER/Studio* da empresa *Embarcadeiro.*

4.5 ESPECIFICAÇÃO

Para a especificação do protótipo foi utilizado a UML, que é apresentado através do diagrama de caso de uso, do diagrama de classes e do diagrama de seqüência. Estes diagramas foram construídos na ferramenta CASE *Rational Rose*.

A UML pode ser usada para mostrar os limites e as principais funções do sistema, representar a estrutura estática, modelar o comportamento dos objetos, apresentar a implementação física e a arquitetura do sistema.

Seus principais módulos são:

a) diagrama de *use case:* é a especificação das ações que um sistema, subsistema, ou classe pode realizar;
b) diagrama de seqüência: é a ordenação das mensagens que acontecem em determinadas funções dentro do sistema.
c) diagrama de classes: mostram classes, interfaces e os relacionamentos entre esses elementos. As classes especificam a estrutura e o comportamento dos objetos no sistema.

A seguir serão apresentadas a especificação e implementação do protótipo, para a exemplificação e exposição dos itens estudados.

5 DESENVOLVIMENTO DO TRABALHO

O presente trabalho resultou na criação de um protótipo de software que possibilita analisar *scripts* de ferramentas CASE. Nele, utilizar-se-á base de conhecimento na forma de regras de produção, uma forma bastante utilizada em muitos dos sistemas especialistas existentes. Sua estrutura constitui-se basicamente de uma premissa ou um conjunto de premissas e de uma conclusão ou de um grupo de conclusões.

5.1 REQUISITOS

O objetivo principal deste protótipo é auxiliar os desenvolvedores na criação dos objetos, em forma de perguntas ao usuário e uma análise das tabelas que estão declaradas nos *scripts*. Com base nestas informações, será feito o acesso à base do conhecimento para verificar parâmetros de criação da tabela e criação de índices adicionais.

Caracteriza-se por perguntas simples e explicação em cada etapa destas, para não deixar o usuário confuso ou com dúvidas. Ao final da análise de uma tabela, é informado o que se alterou somente daquela tabela, tendo a possibilidade de analisar outras tabelas ou gerar um novo *script* para poder salvar em arquivo. Existe também a facilidade de cadastro ou alterações na base do conhecimento, através de um formulário com explicações para cada parâmetro da base. Atende a dois bancos amplamente utilizados no mercado: *Oracle* e *Microsoft SQL Server*.

5.2 ESPECIFICAÇÃO

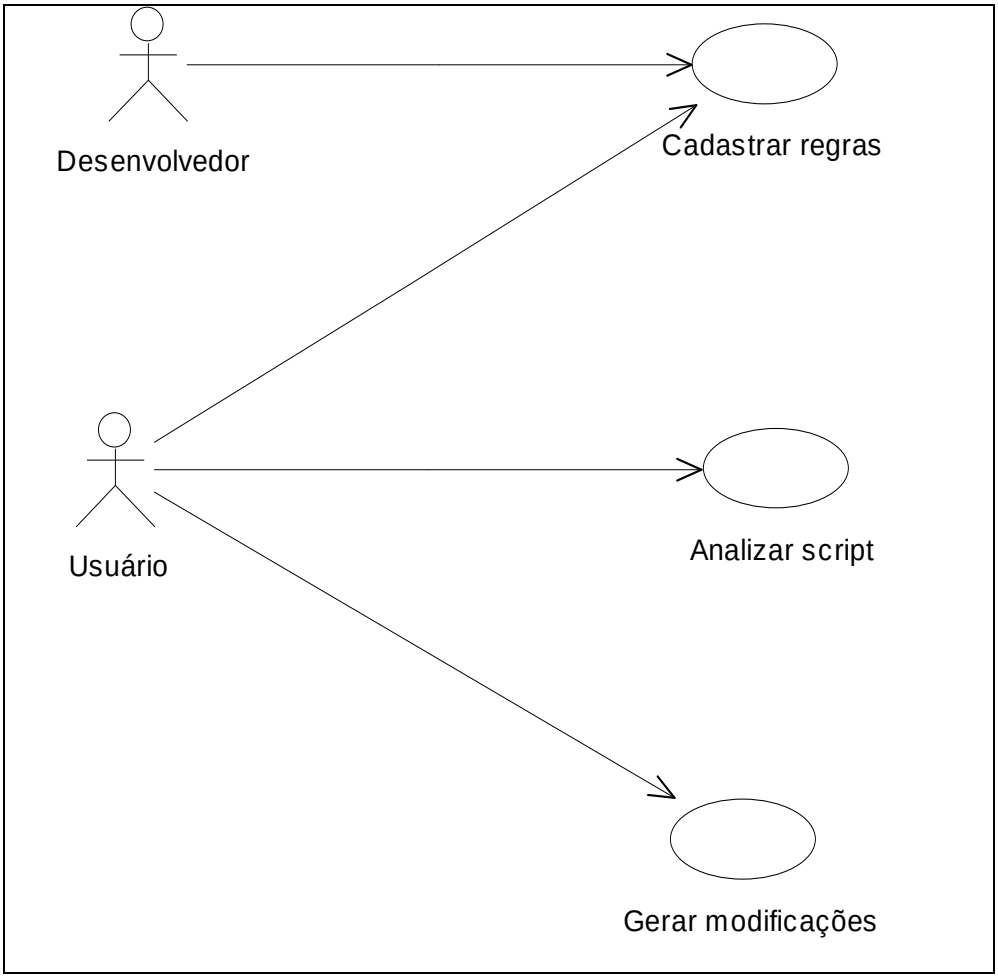
A especificação é apresentada através do diagrama de caso de uso, do diagrama de classes e do diagrama de seqüência.

5.2.1 DIAGRAMA DE CASO DE USO

Para o software foi elaborado o diagrama de casos de uso, sendo que o sistema possui dois usuários principais: o desenvolvedor e o usuário do programa. A função do desenvolvedor é cadastrar as regras iniciais da base de conhecimento, sendo que o usuário também pode cadastrar regras na base, basta ter o conhecimento para tanto.

O usuário solicita ao programa a análise do *script* informado e fornece as respostas para as perguntas que lhe são feitas para fazer as devidas modificações na criação dos objetos. Alguns exemplos de *scripts* gerados pelos CASE encontram-se no anexo I.

Figura 9 – DIAGRAMA DE CASO DE USO



A figura 9 demonstra os casos de uso identificados para o problema que são:

a) cadastrar regras: o desenvolvedor cadastra as regras principais do sistema, sendo que estas mesmas regras podem ser alteradas pelo próprio usuário do sistema;
b) analisar script: o usuário vai respondendo o questionário que será de utilidade para buscar a regra na base do conhecimento;
c) gerar modificações: conforme as respostas do usuário, são feitas modificações necessárias no *script*, que pode ser salvo em disco.

A seguir será demonstrado o diagrama de classes e uma explicação do que é cada classe.

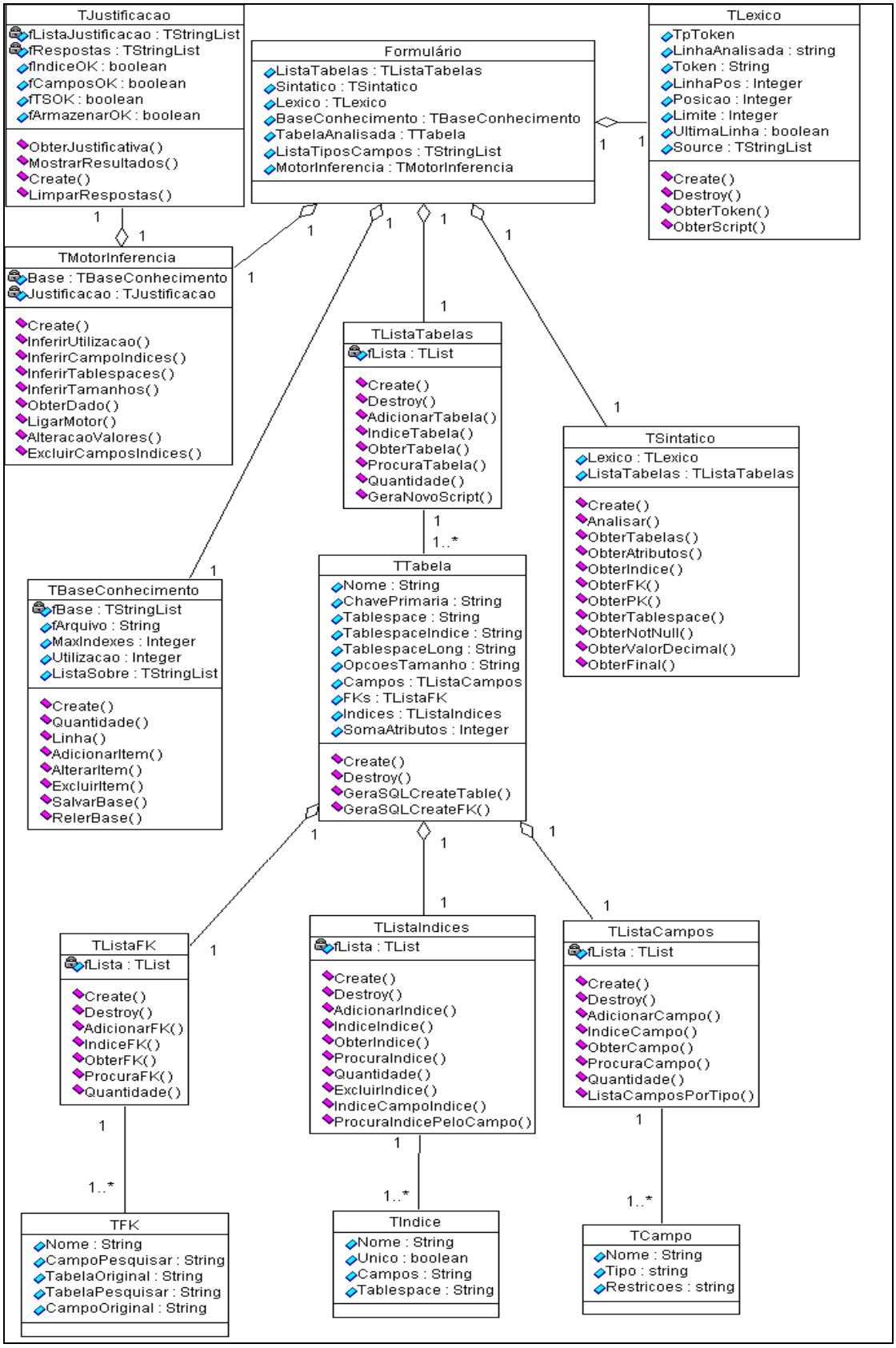
5.2.2 DIAGRAMA DE CLASSES

O diagrama de classes serve para especificar todas as classes que serão utilizadas dentro do sistema. No desenvolvimento do protótipo foram identificadas as classes:

a) TFK - Classe que contém as informações sobre as *foreign key*;
b) TListaFK - Contém a lista de todas as *foreign key* a serem analisadas;
c) TIndice - Classe que contém as informações sobre os índices;
d) TListaIndices - Contém a lista de todos os índices a serem analisadas;
e) TCampo - Classe que contém as informações sobre os campos;
f) TListaCampos - Contém a lista de todos os campos a serem analisadas;
g) TTabela - Classe que contém as informações sobre a tabela;
h) TListaTabelas - Contém a lista de todas as tabelas a serem analisadas;
i) TBaseConhecimento - Classe que contém a base de conhecimento, que são regras definidas pelo desenvolvedor e que podem ser alteradas pelo usuário. Encontrar-se-á em arquivo texto. Nesta classe também está contido o componente “Aquisição de conhecimento” do sistema especialista, visto que não é necessária a criação de uma nova classe para a inserção de dados na base de conhecimento por serem funções de pouca complexidade e acessarem somente esta;
j) TJustificacao - Extrai as justificativas dos itens selecionados na base de conhecimento e apresenta os resultados obtidos ao usuário;
k) TMotorInferencia - O motor de inferência faz o acesso à base de dados e colhe as informações necessárias, direcionando a resposta que será apresentada ao usuário;
l) TLexico - Sua função principal é obter o próximo *token* do *script*;
m) TSintatico - Analisa se os *tokens* provenientes do analisador léxico estão corretos e na ordem certa;
n) TFormulario – Contém as variáveis que pertencem ao sistema, sendo acessíveis a todos os componentes do sistema especialista.

As classes com seus atributos foram especificadas conforme a figura 10.

Figura 10 – DIAGRAMA DE CLASSES



No anexo II encontra-se o cabeçalho das classes criadas.

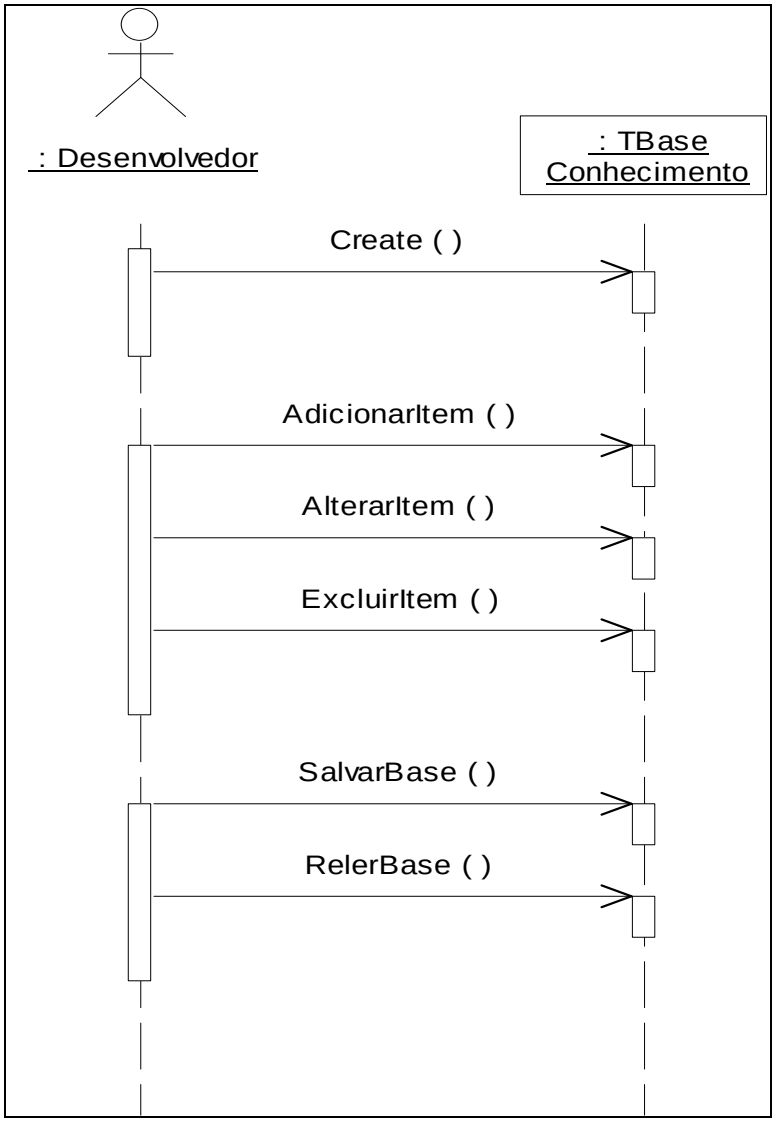
A seguir serão demonstradas as principais funções do sistema através do diagrama de seqüência.

5.2.3 DIAGRAMA DE SEQÜÊNCIA

Na elaboração do diagrama de seqüência foram abordados os três principais casos que ocorrem no sistema, que é a alteração na base de conhecimento através do desenvolvedor e usuário, a análise e captura dos dados e a geração das novas modificações.

Na figura 11 visualiza-se o cadastro das regras na base de conhecimento pelo desenvolvedor. Ao cadastrar um novo item, a base de conhecimento que é mantida em memória é atualizada, não sendo necessária a saída do programa para atualizar os dados. Os dados da base de conhecimento são mantidos em uma lista de *strings (StringList)*, sendo acessados através do motor de inferência quando se altera a resposta do usuário.

Figura 11 – DESENVOLVEDOR CADASTRA REGRAS

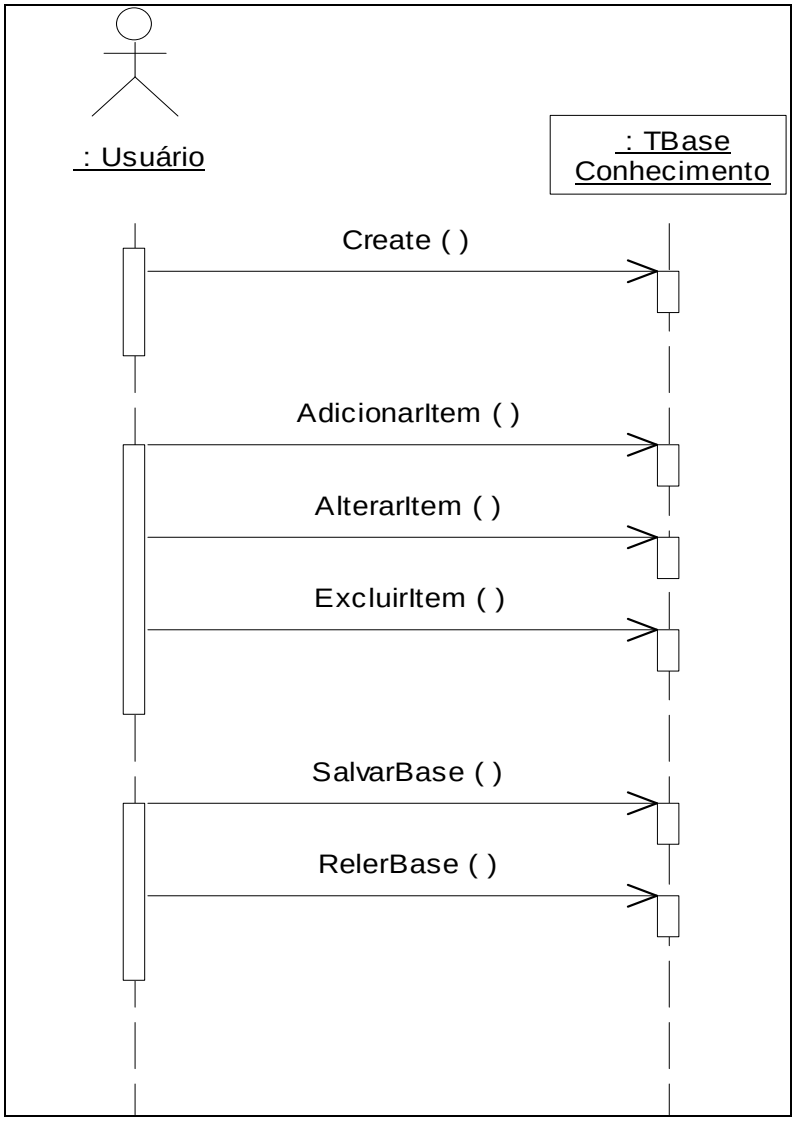


Ao adicionar, incluir ou alterar um item na lista, o programa o faz na memória, sendo que se o usuário ou o desenvolvedor resolver não gravar realmente os itens, o programa irá reler o arquivo da base de conhecimento, caso contrário, salvará a nova base que está em memória.

A base de conhecimento é criada na hora em que o formulário é criado, sendo passado como parâmetro para o motor de inferência, facilitando a implementação do mesmo. A base do conhecimento possui um tamanho reduzido sendo que as regras não são de grande complexidade. As regras podem ser vistas conforme o anexo III.

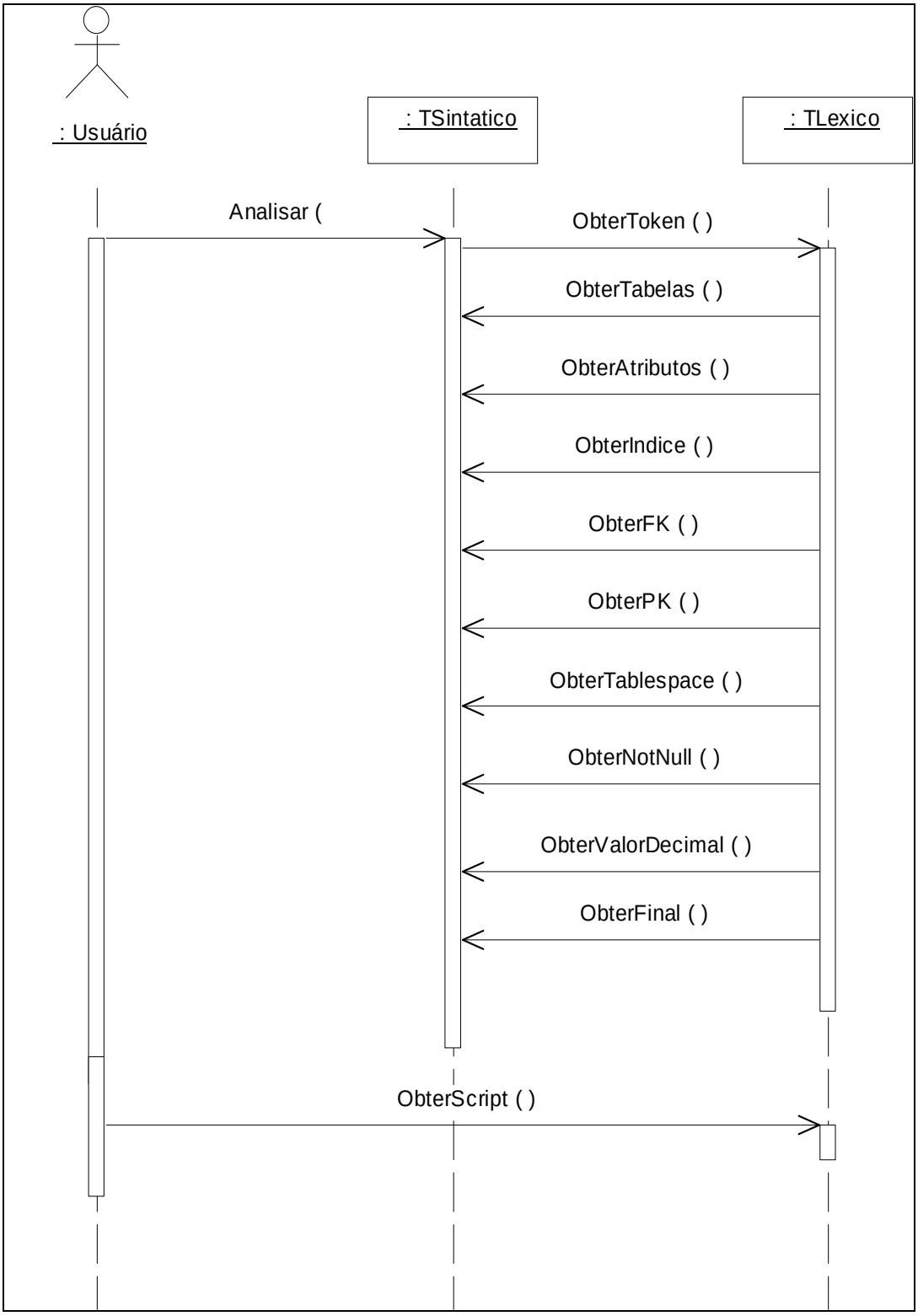
A figura 12 demonstra o mesmo cadastro de regras descrito anteriormente pelo desenvolvedor, com a diferença que o personagem que o faz agora é o usuário do sistema. Porém as condições de gravação e leituras são as mesmas.

Figura 12 – USUÁRIO CADASTRA REGRAS



Na figura 13 o usuário solicita a análise do *script*, que fará uma análise léxica e sintática do *script* informado.

Figura 13 – ANALISAR *SCRIPT*

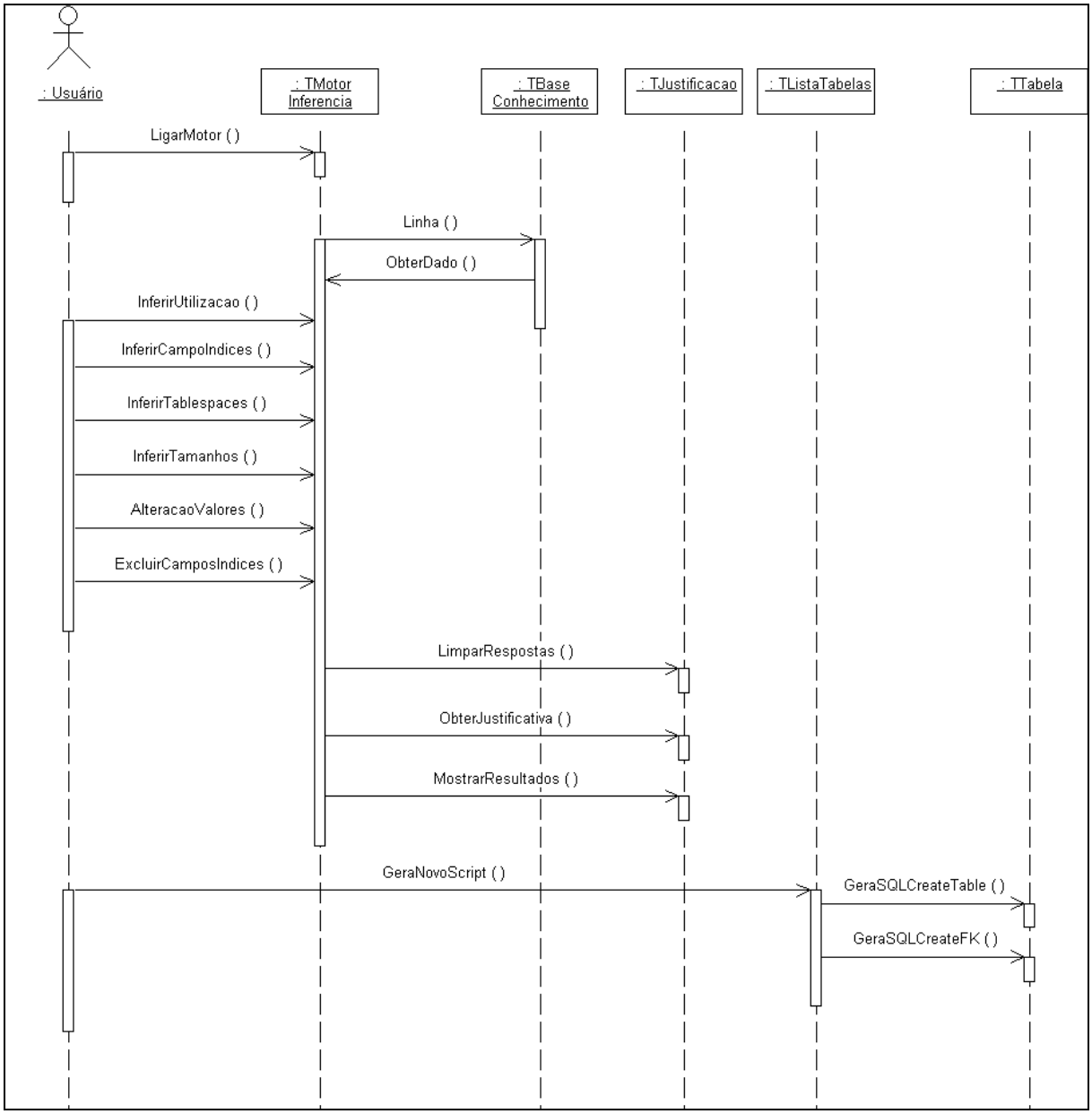


O usuário solicita a análise do *script*, e a medida que o analisador léxico vai repassando os *tokens* ao analisador sintático, o mesmo armazena em memória as tabelas, campos, índices e *foreing keys* analisadas.

A lista de tabelas é armazenada em uma variável derivada do tipo *TList*, com ponteiros para as demais estruturas relacionadas a campos, índices e chaves estrangeiras.

Após esta análise do que se encontra no *script*, uma lista com as tabelas são apresentadas para o usuário. O mesmo escolhe a tabela a analisar, fazendo com que inicie o sistema especialista, conforme a figura 14.

Figura 14 – ANALISAR TABELA



O motor de inferência tem como finalidade principal acessar a base de conhecimento e trazer o item relacionado às informações respondidas pelo usuário.

Quando o usuário escolhe uma tabela, o sistema busca todas as informações da tabela na lista e tabela e armazena os dados em um objeto do tipo TTabela, começando a interagir com o mesmo, fazendo as perguntas necessárias para pegar os itens na base de conhecimento.

Nas trocas de respostas dos usuários, as informações são alteradas pela rotina AlteracaoValores(), que verifica o que deve-se buscar na base, chamando a rotina que corresponde às informações fornecidas pelo usuário.

A função ObterDado() é a rotina que busca a informação fornecida pelo usuário. A informação propriamente dita pode não constar na base, sendo que a mais próxima será adquirida pela rotina. A rotina é uma busca em profundidade, sendo que foi implementada a busca em até quatro níveis de profundidade através da rotina InferirUtilizacao(), que busca pela soma dos campos, o nível de utilização da tabela, a quantidade de registros iniciais da tabela e a quantidade de registros inseridos semanalmente.

A rotina LigarMotor() faz a inicialização das variáveis e limpeza dos campos da tela.

A função InferirCampoIndice() insere os campos que são informados pelo usuário na lista de índices da tabela. Uma quantidade ideal de índices consta na base de conhecimento de acordo com o nível de utilização da tabela.

A justificativa para o dado escolhido é buscado dentro da rotina ObterDado(), sendo que a justificativa se encontra na mesma linha da informação, sendo separados pelo identificador pipe (|).

Os dados são alterados na própria variável TabelaAnalisada, sendo que as alterações são informadas ao final da analise da mesma.

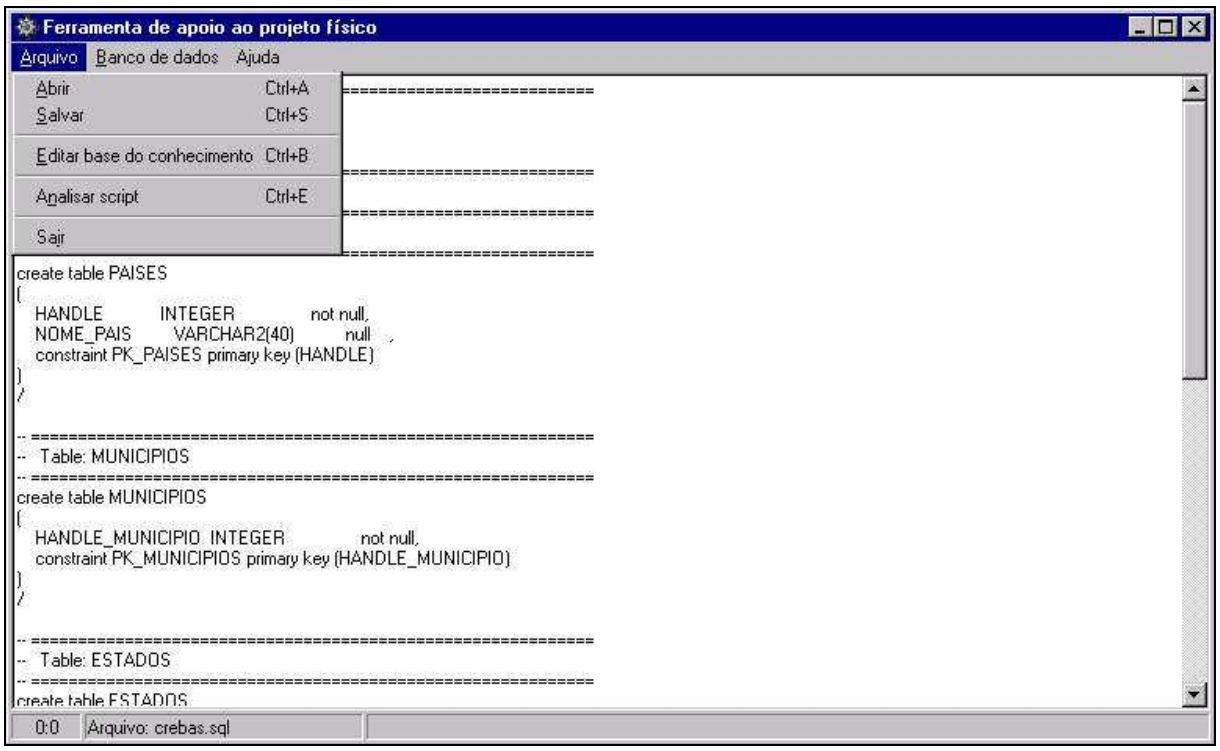
A seguir serão apresentadas as funções e telas do protótipo.

5.3 IMPLEMENTAÇÃO

Foram escolhidos os bancos de dados *Oracle* e *Microsoft SQL Server* por serem bancos com boa aceitação no mercado e por possuírem características de banco de dados relacionais e com bibliografia disponíveis. Os CASE *ER\Studio* e *Power Designer* foram escolhidos pela mesma forma dos bancos. Eles são comuns no mercado e o *Power Designer* é ensinado na universidade onde foi apresentado este trabalho. Os comandos que podem ser analisados são os comandos: *create table* e *alter table*.

Considerações sobre a forma de operação e de implementação serão apresentadas nesta seção. A implementação foi feita em *Delphi* 5 e apresenta a tela principal conforme a figura 15.

Figura 15 – TELA PRINCIPAL



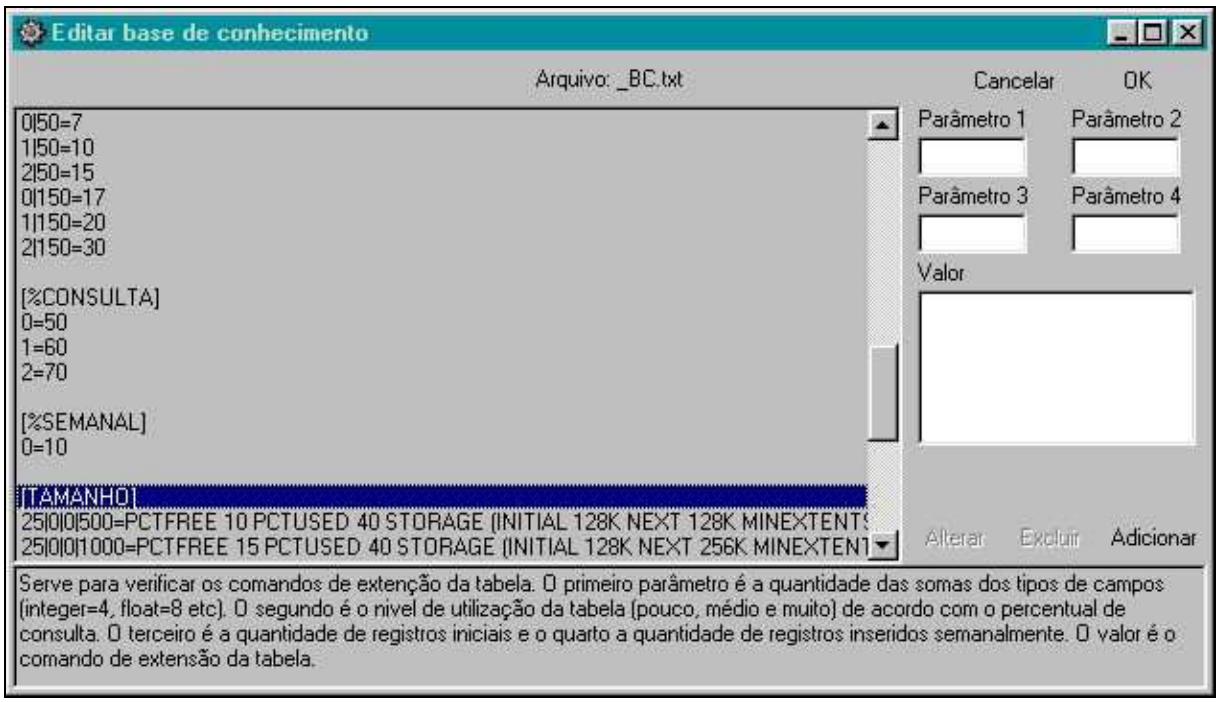
No menu "Arquivo" pode-se acessar as funções para abrir os *scripts*, salvar as modificações, editar a base de conhecimento e analisar o *script*.

Na tela principal é informado o *script* para ser analisado. No menu "Banco de dados" define-se que banco que está sendo trabalhado, devido a algumas diferenças nos comandos de criação dos objetos.

O item “Editar base do conhecimento” serve para cadastrar, alterar e incluir novos itens na base de conhecimento.

A base de conhecimento foi feita em arquivo texto simples, com suas *tags* e valores. As *tags* são explicativas na própria tela de edição da base, conforme demonstra a figura 16.

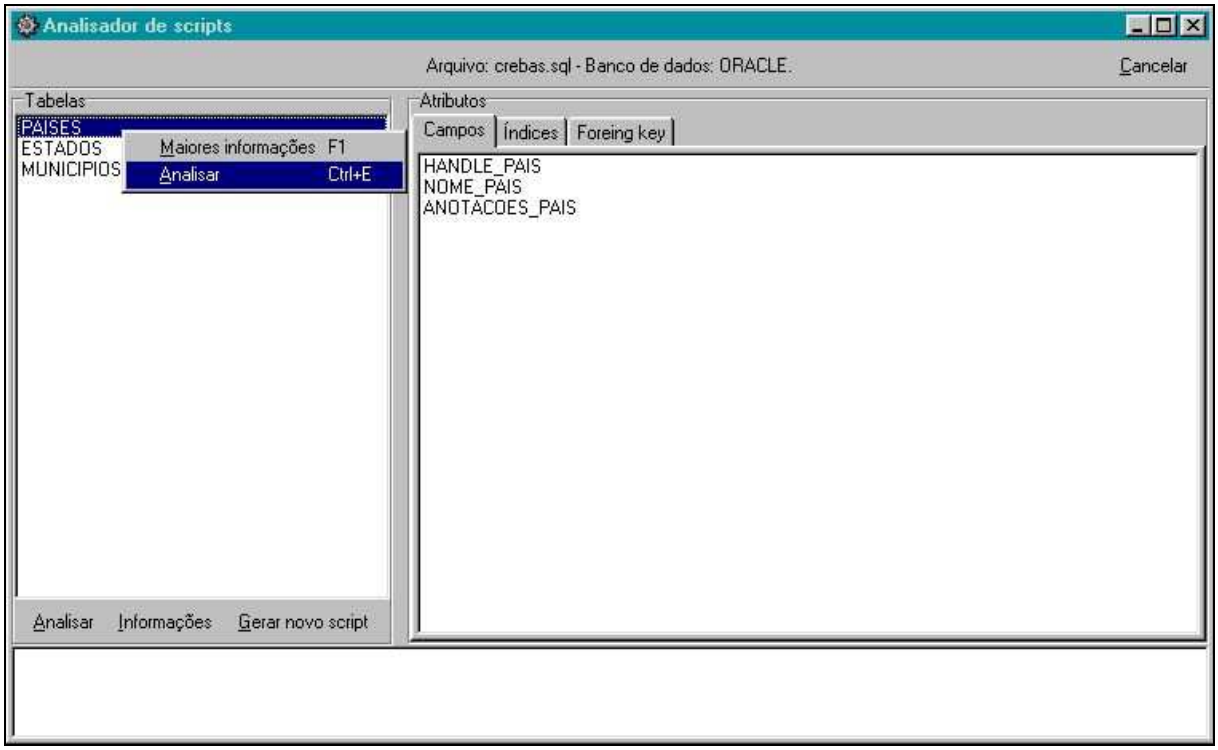
Figura 16 – EDITAR BASE DE CONHECIMENTO



Apenas novos itens nas sessões podem ser incluídos. Sessões novas não serão reconhecidas pelo sistema especialista.

O formulário de análise das bases de dados utiliza um analisador léxico e sintático, sendo que estes podem ser vistos no anexo IV, para obter as definições vindas do *script* e transporta para variáveis em memória, que é o componente “Quadro negro” do sistema especialista. O léxico obtém o *token* e o analisador sintático verifica se o *token* que está sendo enviado está na seqüência correta. Assim, a lista de tabelas e seus atributos são montados e apresentado ao usuário para análise conforme a figura 17.

Figura 17 – APRESENTAÇÃO DAS TABELAS

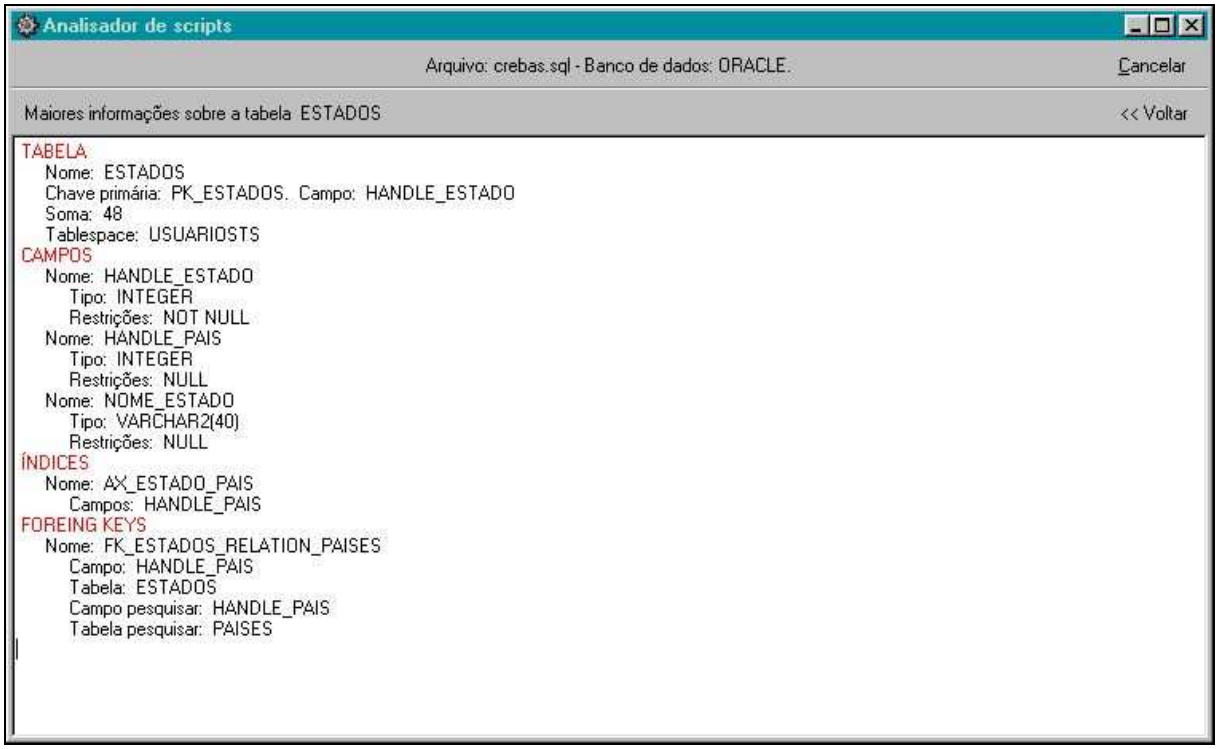


Com a lista de tabelas apresentada, o usuário pode escolher que tabelas analisar, clicando com o botão direito em cima da tabela ou utilizando-se das teclas de atalho. Nesta tela é apresentada uma breve introdução do que foi analisado, sendo que se for necessário que se tenha mais informações da tabela bastam pressionar F1 ou selecionar o item na lista.

É utilizada a tabela, pois elas são os objetos principais, sendo que os campos, índices e chaves estrangeiras estão ligados abaixo dela.

As informações são apresentadas conforme a figura 18.

Figura 18 – MAIORES INFORMAÇÕES SOBRE A TABELA



Os itens são apresentados de uma forma mais legível ao usuário para que ele possa verificar todas as informações que o analisador léxico e sintático fizeram.

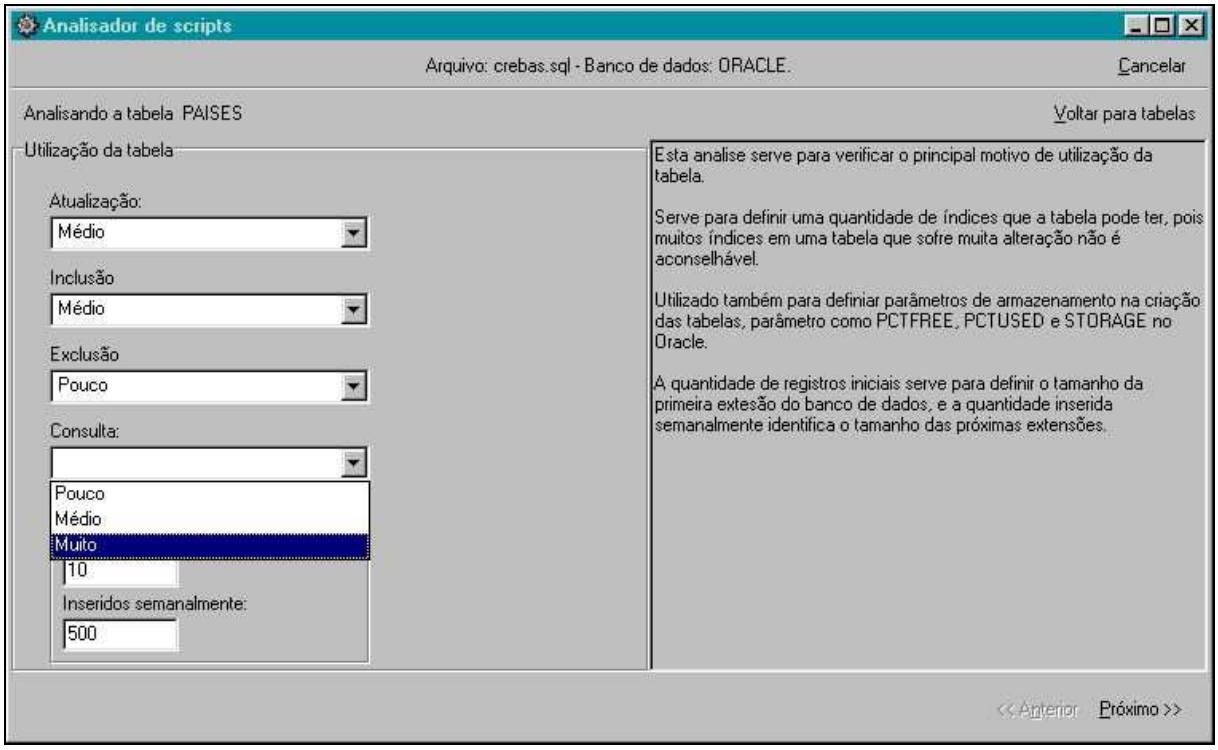
Fazendo a análise da tabela, apresentar-se-á o formulário de analise de *script*, que atua como o componente "Sistema de Consulta" da inteligência artificial. Nesta parte é que são feitas as perguntas ao usuário.

Nesta etapa também é utilizado o "Quando Negro" da inteligência artificial, que são as tabelas, campos, índices e chaves estrangeiras armazenadas em memória.

O motor de inferência é utilizado fazendo acesso à base de conhecimento de acordo com as definições das respostas do usuário neste mesmo formulário.

A primeira tela apresenta as perguntas quanto ao nível de utilização da tabela. As informações sobre o nível de utilização da tabela servem para indicar uma quantidade ideal de índices para a tabela e também serve para escolher as formas de extensões ao final do comando de criação das tabelas.

Figura 19 – NÍVEL DE UTILIZAÇÃO



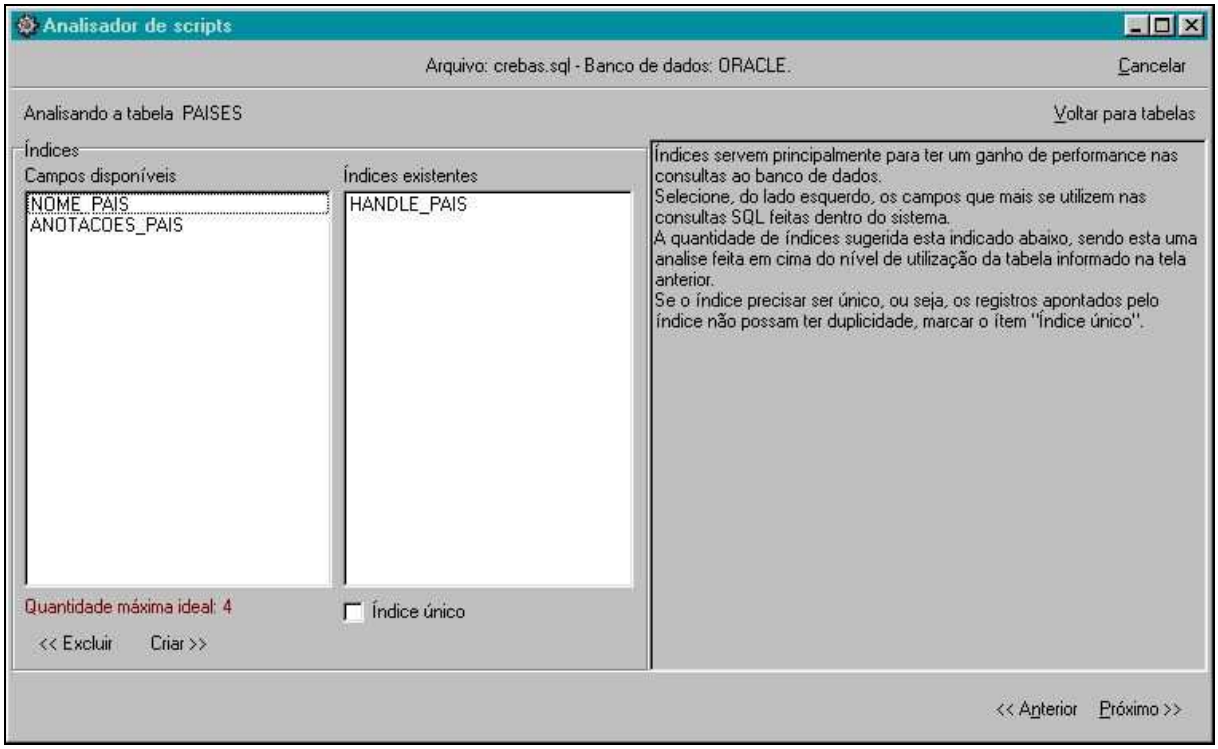
Os níveis são descritos como pouco, médio e muito, e são compreendidos em dois grupos principais. O primeiro engloba as atualizações, inclusões e exclusões e o segundo engloba apenas as consultas que são feitas.

Nesta tela também é informado ao programa a quantidade de registros que a tabela irá ter inicialmente e a quantidade de registros incluídos semanalmente.

Na tela seguinte serão apresentados os índices que a tabela possui e quais os outros campos que estão disponíveis para a indexação. A quantidade máxima de índices é extraída através do nível de utilização da tabela conforme a figura 19.

Na definição de índices, pode-se incluir, excluir e alterar a unicidade dos índices conforme a figura 20.

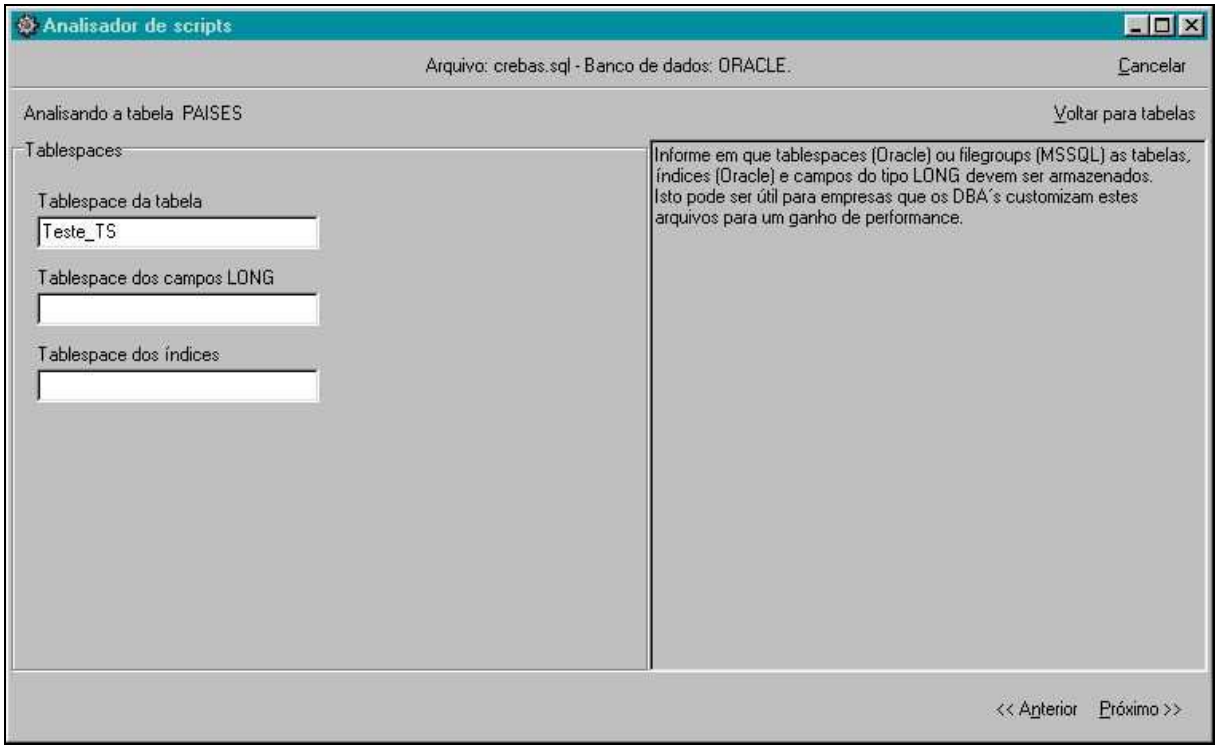
Figura 20 – ÍNDICES



A quantidade ideal de índices é apresentada no canto da tela, não sendo necessária a inclusão nem a exclusão de índices para chegar a tal quantidade.

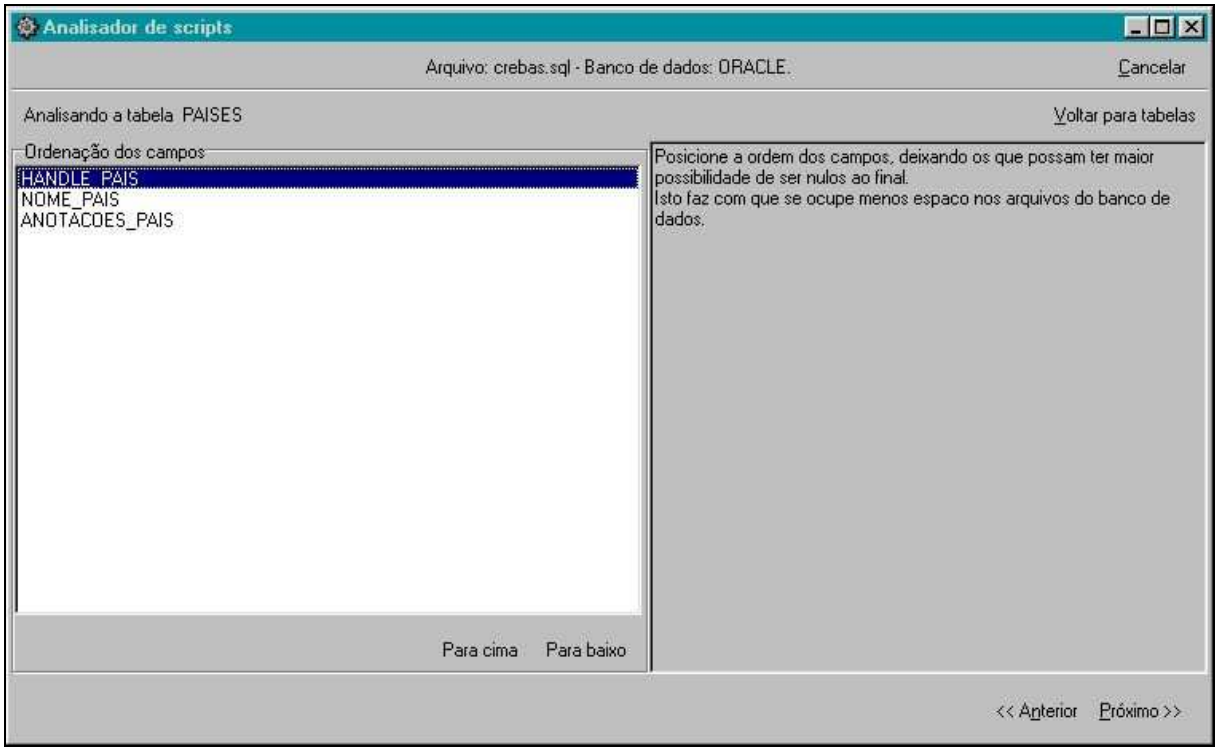
Na figura 21, é informado os *tablespaces* e *datafiles* para armazenar as tabelas, índices e campos do tipo *long*.

Figura 21 – *TABLESPACES* E *DATAFILES*



Após estas alterações as informações sobre a ordem dos campos são demonstradas, conforme a figura 22. Isto serve para que se ocupe menos espaço nos arquivos de dados dos bancos. Recomenda-se que os campos que possam vir a ter mais valores nulos fiquem no final do comando de criação da tabela.

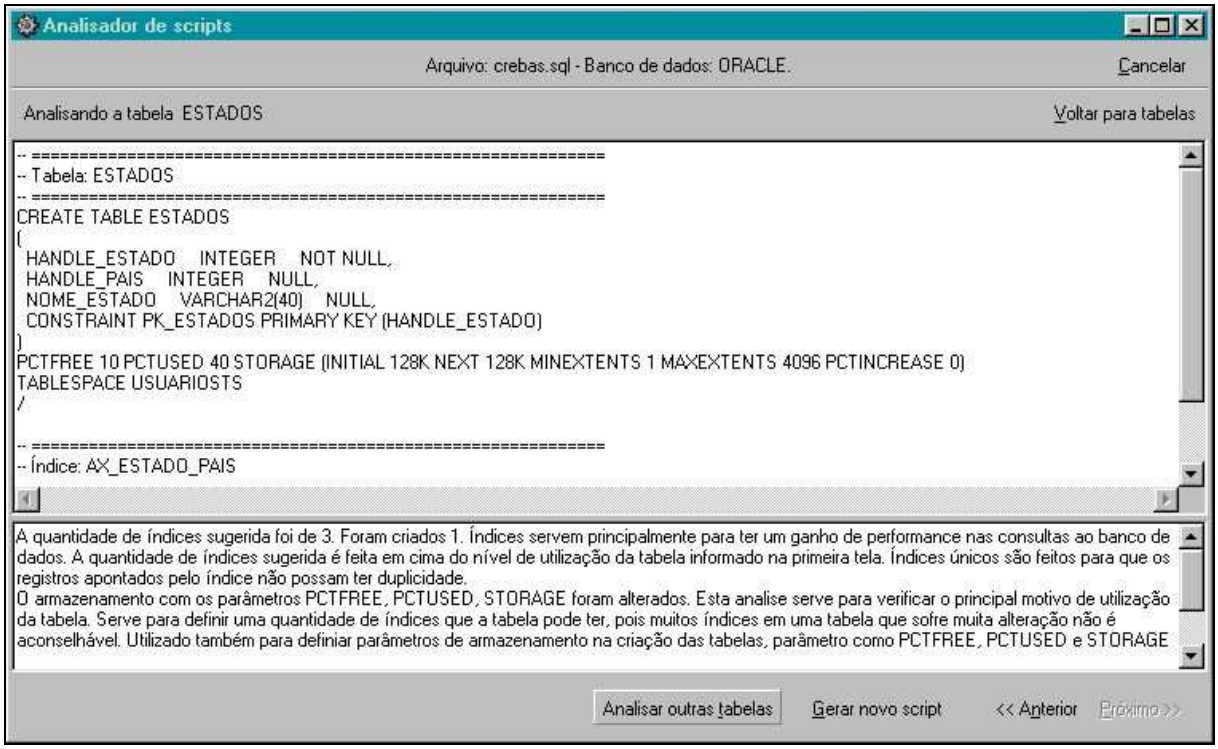
Figura 22 – ORDEM DOS CAMPOS



As alterações das ordens dos campos foram estudadas no capítulo 2.7.1.

Após isso, são apresentadas ao usuário todas as modificações sugeridas de acordo com as respostas fornecidas ao programa, conforme demonstra a figura 23. Em todas as telas existe também uma explicação do porque está se fazendo cada pergunta.

Figura 23 – RESULTADOS



Os resultados e as modificações serão demonstrados conforme a figura 23, podendo então modificar o arquivo principal através do botão “Gerar novo script” sendo que subseqüentemente poderá ser salvo em arquivo ou então analisar outras tabelas. As tabelas vão sendo marcadas como analisadas, mas podem sofrer a análise novamente.

A seguir apresentar-se-á as conclusões e sugestões para trabalhos futuros.

6 CONSIDERAÇÕES FINAIS

6.1 CONCLUSÃO

Ao desenvolver este estudo, buscaram-se informações acerca dos conceitos básicos de banco de dados e dos sistemas gerenciadores de banco de dados. Possibilitou-se a compreensão do funcionamento do modo de armazenamento dos bancos *Oracle* e *Microsoft SQL Server* e suas sintaxes de criação da principal estrutura de armazenamento: as tabelas e seus atributos.

Considera-se de grande importância o estudo feito sobre os sistemas especialistas oriundos da inteligência artificial, pois auxilia não só o desenvolvimento do protótipo como também foi uma peça importante no desenvolvimento de todo o trabalho.

As ferramentas utilizadas foram adequadas para o desenvolvimento do trabalho, sendo que os *scripts* gerados pelos CASE foram de fácil entendimento. O ambiente *Delphi* foi muito útil devido à facilidade de apresentação das telas para o desenvolvimento das perguntas aos usuários.

Adquiriram-se também conhecimento acerca de compiladores, especificamente analisadores léxico e sintático, que foi útil para a extração dos *tokens* para a identificação das tabelas e seus atributos.

Os bancos de dados *Oracle* e *Microsoft SQL Server* foram escolhidos por serem bancos com boa aceitação no mercado e por possuírem características de banco de dados relacionais e com bibliografia disponíveis. Os CASE *ER\Studio* e *Power Designer* foram escolhidos pela mesma forma dos bancos.

Com o decorrer deste estudo tem-se o intento de consolidar a utilização dos conhecimentos agregados no decorrer da formação acadêmica.

O programa atendeu os requisitos propostos, apresentando todos os componentes do sistema especialista e fazendo com que se gerassem as modificações necessárias para a melhoria na qualidade dos objetos físicos criados.

6.2 SUGESTÕES PARA TRABALHOS FUTUROS

Para extensões deste trabalho sugere-se:

a) estender os conceitos para outros bancos de dados;
b) tornar possível a geração de uma saída para validação dos objetos criados;
c) possibilitar a criação de índices compostos e outros tipos de índices;
d) criar uma especificação na BNF para estender a forma de análise do *tokens* gerados pelos CASE.

6.3 LIMITAÇÕES

Por se tratar de um protótipo, algumas limitações foram encontradas:

a) não foi implementada segurança através de identificação de usuários;
b) não foi gerada uma saída para a criação dos objetos alterados;
c) versões recentes de bancos *Oracle* e *Microsoft SQL Server* foram lançadas, com mais comandos, mas estes não foram estudados, sendo ainda versões recentes;
d) vários tipos de índices existem para os bancos, sendo que foi utilizado apenas o tipo de índice simples de cada banco.

Contudo, os objetivos principais deste projeto foram alcançados e validados conforme proposto.

ANEXO I

```
-- ============================================================
--   Database name:  TCC_MODEL_ORACLE_PD
--   DBMS name:      ORACLE Version 8
--   Created on:     23/4/2002  15:10
-- ============================================================


-- ============================================================
--   Table: PAISES
-- ============================================================
create table PAISES
(
    HANDLE_PAIS       INTEGER               not null,
    NOME_PAIS         VARCHAR2(40)          null    ,
    constraint PK_PAISES primary key (HANDLE_PAIS)
)
/


-- ============================================================
--   Table: ESTADOS
-- ============================================================
create table ESTADOS
(
    HANDLE_ESTADO     INTEGER               not null,
    HANDLE_PAIS       INTEGER               null    ,
    NOME_ESTADO       VARCHAR2(40)          null    ,
    constraint PK_ESTADOS primary key (HANDLE_ESTADO)
)
tablespace usuariosts
/
```

```
-- ============================================================
--   Table: MUNICIPIOS
-- ============================================================
create table MUNICIPIOS
(
    HANDLE_MUNICIPIO  INTEGER               not null,
    HANDLE_ESTADO     INTEGER               null    ,
    constraint PK_MUNICIPIOS primary key (HANDLE_MUNICIPIO)
)
/


-- ============================================================
--   Index: AX_ESTADO_PAIS
-- ============================================================
create index AX_ESTADO_PAIS on ESTADOS (HANDLE_PAIS asc)
/


-- ============================================================
--   Index: AX_MUNICIPIO_ESTADO
-- ============================================================
create index AX_MUNICIPIO_ESTADO on MUNICIPIOS (HANDLE_ESTADOS asc)
/


alter table ESTADOS
    add constraint FK_ESTADOS_RELATION_PAISES foreign key  (HANDLE_PAIS)
       references PAISES (HANDLE_PAIS)
/


alter table MUNICIPIOS
    add constraint FK_MUNICIPIOS_TEM_ESTADOS foreign key  (HANDLE_ESTADO)
       references ESTADOS (HANDLE_ESTADO)
/
```

```
------------------------------------------------------------

/* ============================================================ */
/*   Database name:  TCC_MODEL_MSSQL_PD                         */
/*   DBMS name:      Microsoft SQL Server 6.x                   */
/*   Created on:     29/03/02  13:50                            */
/* ============================================================ */

/* ============================================================ */
/*   Table: PAISES                                              */
/* ============================================================ */
create table PAISES
(
    HANDLE_PAIS    INTEGER               not null,
    NOME_PAIS      VARCHAR2(40)          null    ,
    constraint PK_PAISES primary key (HANDLE_PAIS)
)
go

/* ============================================================ */
/*   Table: ESTADOS                                             */
/* ============================================================ */
create table ESTADOS
(
    HANDLE_ESTADO  INTEGER               not null,
    HANDLE_PAIS    INTEGER               null    ,
    NOME_ESTADO    VARCHAR2(40)          null    ,
    constraint PK_ESTADOS primary key (HANDLE_ESTADO)
)
on TESTE
go

/* ============================================================ */
```

```
/*   Index: AX_ESTADO_PAIS                                      */
/* ========================================================== */
create index AX_ESTADO_PAIS on ESTADOS (HANDLE_PAIS)
go


alter table ESTADOS
    add constraint FK_ESTADOS_RELATION_PAISES foreign key  (HANDLE_PAIS)
       references PAISES (HANDLE_PAIS)
go
```

ANEXO II

```
//-------------------Foreing Keys-----------------------
//TFK - Classe que contém as informações sobre as FK
  TFK = class
  public

    Nome,                               { Nome da FK }
    CampoPesquisar,                     { Campo sendo pesquisado }
    TabelaOriginal,                     { Tabela da FK}
    TabelaPesquisar,                    { Tabela sendo pesquisada }
    CampoOriginal: string;              { Campo da FK }
  end;

//TListaFK - Contém a lista de todas as FK a serem analisadas
  TListaFK = class
  private
    fLista : TList;
  public
    constructor Create;                       { Cria a lista de FKs }
    destructor  Destroy; override;            { Destroi a lista }
    function AdicionarFK(FK: TFK): Integer;   { Adiciona uma FK na lista }
    function IndiceFK(Nome: string): Integer; { Procura o indice de uma FK na lista }
    function ObterFK(Indice: Integer): TFK;   { Retorna os atributos da FK }
    function ProcuraFK(Nome:string): TFK;     { Retorna os atributos da FK pelo nome }
    function Quantidade: integer;             { Retorna quantidade de FK }
  end;
//-------------------Fim Foreing Keys-----------------------

//-------------------Indices-----------------------
//TIndice - Classe que contém as informações sobre os índices
  TIndice = class
```

```
  public
    Nome:       string;                          { Nome do indice }
    Unico:      boolean;                         { Indice unico }
    Campos:     string;                          { Campos separados por virgula }
    TableSpace: string;                          { Tablespace para o indice }
  end;

//TListaIndices - Contém a lista de todos os índices a serem analisadas
  TListaIndices = class
  private
    fLista : TList;
  public
    constructor Create;                                          { Cria a lista de
indices }
    destructor  Destroy; override;                              { Destroi a lista }
    function AdicionarIndice(Indice: TIndice): Integer;          { Adiciona um indice
na lista }
    procedure ExcluirIndice(Campo: string);                     { Exclui um indice na
lista }
    function IndiceIndice(Nome: string): Integer;               { Procura o indice de
uma tabela na lista pelo nome}
    function IndiceCampoIndice(Nome: string): Integer;          { Procura o indice de
uma tabela na lista pelo campo}
    function ObterIndice(Indice: Integer): TIndice;                  { Retorna os
atributos dos indices }
    function ProcuraIndice(Nome:string): TIndice;                    { Retorna os
atributos dos indices pelo nome }
    function ProcuraIndicePeloCampo(Campo:string): TIndice;          { Retorna os
atributos dos indices pelo nome }
    function Quantidade: integer;                                   { Quantidade de
indice }
  end;
//-------------------Fim Indices-----------------------

//-------------------Campos-----------------------
//TCampo - Classe que contém as informações sobre os campos
```

```
  TCampo = class
  public
    Nome,               { Nome do campo }
    Tipo: string;       { Tipo do campo [string, inteiro etc] }
    Restricoes: string; { Indica se o campo permite null ou tem checks}
  end;


//TListaCampos - Contém a lista de todos os campos a serem analisadas
  TListaCampos = class
  private
    fLista : TList;
  public
    function AdicionarCampo(Campo: TCampo): Integer;    { Adiciona um campo na lista }
    constructor Create;                                 { Cria a lista de campos }
    destructor  Destroy; override;                      { Destroi a lista }
    function IndiceCampo(Nome: string): Integer;         { Procura o indice de um campo
na lista }
    function ListaCamposPorTipo(Tipo: string): String;   { Procura um tipo de campo e
retonra uma lista com os nomes dos campos }
    function ObterCampo(Indice: Integer): TCampo;       { Retorna os atributos do campo
pelo indice }
    function ProcuraCampo(Nome:string): TCampo;         { Retorna os atributos do campo
pelo nome }
    function Quantidade: integer;                       { Quantidade de campos }
    procedure InverterPosicao(OldPos, NewPos: integer); { Altera a posicao dos campos
na tabela }
  end;
//-------------------Fim campos-----------------------


//-------------------Tabelas-----------------------
//TTabela - Classe que contém as informações sobre a tabela
  TTabela = class
  public
    Nome               : string;                      { Nome da tabela }
```

```
          ChavePrimaria        : string;                          { Campo e nome da chave
primaria }
          Tablespace           : string;                          { Tablespace da tabela -
somente para oracle }
          TablespaceIndice     : string;                          { Tablespace da indice -
somente para oracle }
          TablespaceLong       : string;                        { Tablespace dos campos tipo
long - somente para oracle }
          OpcoesTamanho        : string;                          { Definicao do comando de
extenxão no Oracle }
          Campos              : TListaCampos;                { Campos da tabela }
          FKs                 : TListaFK;                    { FKs da tabela }
          Indices             : TListaIndices;               { Indices da tabela }
          SomaCampos          : integer;                      { Soma dos tamanhos dos Campos
}
          JaAnalisada          : boolean;                      { Indica que a tabela ja foi
analisada }
          constructor Create;                                { Cria a lista de atributos }
          destructor  Destroy; override;                      { Destroi a lista de atributos
}
          procedure GeraSQLCreateTable(Comando: TStringList); { Gera um comando de create
table da tabela }
          procedure GeraSQLCreateFK(Comando: TStringList);     { Gera um comando de alter
table para as FK da tabela }
        end;


      //TListaTabelas - Contém a lista de todas as tabelas a serem analisadas
        TListaTabelas = class
        private
          fLista: TList;                                     { Aqui serao adicionadas todas
as tabelas encontradas no script}
        public
          constructor Create;                                { Cria a lista de tabelas }
          destructor  Destroy; override;                          { Destroi a lista e suas
tabelas filhas }
          function AdicionarTabela(Tabela: TTabela): Integer; { Adiciona uma tabela na lista
}
```

```
           function IndiceTabela(Nome: string): Integer;          { Procura o indice de uma
tabela na lista }

           function ObterTabela(Indice: Integer): TTabela;        { Retorna a tabela e seus
atributos pelo indice }

           function ProcuraTabela(Nome:string): TTabela;          { Retorna a tabela e seus
atributos pelo nome }

           function Quantidade: integer;                             { Quantidade de tabelas
existentes }

           function GeraNovoScript: TStringList;               { Gera um novo script com todas
as tabelas encontradas e analisadas com suas modificacoes }

         end;

       //-------------------Fim Tabelas-----------------------


       type

       {

       TBaseConhecimento - Classe que contém a base de conhecimento,

       regras definidas pelo desenvolvedor que podem ser alteradas pelo usuário.

       Se encontrará em arquivo texto.

       Nesta classe também se encontrará a componente "Aquisição de conhecimento"

       do sistema especialista, pois seria supérfluo criar uma nova classe para inserir

       dados na base de conhecimento

       }

         TBaseConhecimento = class

         private

           fBase :   TStringList;

         public

           fArquivo: string;                       { Armazena o nome do arquivo da base de
conhecimento }

           MaxIndexes: integer;            { Quantidade maxima ideal de indices }

           Utilizacao: integer;              { Resultado da pesquisa do nivel de utilizacao da
tabela }

           ListaSobre: TStringList;            { Contem as informacoes do que significa cada
sessao da base de conhecimento }


       //---------------------------------------

           constructor Create(aArquivo: string);
```

```
            function    Quantidade: integer;
            function    Linha(Indice: integer):string; { Retorna a linha da base pelo indice}
        //--------------------------------------
            { Funcoes do componente "Aquisicao do conhecimento do sistema especialista" }
            procedure AdicionarItem(Indice: integer; Item: string);              { Adiciona um
item a lista }
            function  AlterarItem(Indice: integer;NovoItem: string): boolean;  { Altera o valor
de um item da lista }
            function  ExcluirItem(Indice: integer): boolean;                   { Exclui um item
da lista }
            procedure SalvarBase;                                                 { Salva as
modificacoes do usuario na base }
            procedure RelerBase;                                               { Rele a base do
arquivo }
            { Fim funcoes do componente "Aquisicao do conhecimento do sistema especialista" }
          end;


        {
        TJustificacao - Extrai as justificativas dos itens selecionados na base de conhecimento
e apresenta
        os resultados obtidos ao usuário
        {}
          TJustificacao = class
          private
            fListaJustificacao: TStringList;                                    { Contem as
justificativas no arquivo _BCJustificativa.txt }
            fRespostas : TStringList;                                    { Monta as respostas
para ser informado no final ao usuario }
          public
            fIndiceOK,                                                  { Indica se os indices
ja foram justificados }
            fCamposOK,                                                   { Indica se os campos
ja foram justificados }
            fTSOK,                                                             { Indica se as
tablespaces ja foram justificados }
            fArmazenarOK: boolean;                                       { Indica se os formas
de extensoes ja foram justificados }
```

```
          constructor Create;
          procedure MostrarResultados;                                { Mostra o resultado ao
usuario apos a analise da tabela }
          procedure LimparRespostas;                                    { Limpa as respostas
para informar as outras respostas de outras tabelas }
          function  ObterJustificativa(Item: string): string;         { Monta a justificativa
da sessao informada }
        end;


      {
      TMotorInferencia - O motor de inferência faz o acesso à base de dados e colhe as
informações
      necessárias, direcionando a resposta que será apresentada ao usuário
      }
        TMotorInferencia = class
        private
        public
          Base           : TBaseConhecimento;
          Justificacao   : TJustificacao;
      //---------------------------------------
          procedure  InferirUtilizacao(Atualizacao,  Inclusao,  Exclusao,  Consulta:  string);
{ Faz a busca do nivel de utilizacao }
          procedure            InferirCamposIndices(NomeCampo           :           string);
{ Cria os indices na tabela analisada }
          procedure            ExcluirCamposIndices(NomeCampo           :           string);
{ Exclui os campos da tabela analisada }
          procedure         InferirTablespaces(var         Tabela         :         TTabela);
{ Altara os tablespaces }
          procedure    InferirTamanhos(var    Tabela:    TTabela;Utilizacao,Valor,ValorSemanal:
string);   { Pega o valor dos registros iniciais, semamanais, nivel de utilizacao e qtdade
campos para buscar comandos de extensao }
      //---------------------------------------
          function      ObterDado(Sessao: string; Valores: string; Maior: boolean): string;
{ Faz a busca do valor na sessao informada na base de dados }
          function                        AlteracaoValores(Valor:      integer):      string;
{ Reve os valores de acordo com a pagina sendo respondida pelo usuario }
```

```
          procedure                                                    LigarMotor;
{ Inicializa as variaveis }
          constructor            Create(fBase            :            TBaseConhecimento);
{ Cria a base }
        end;


        {
        O componente "Quadro Negro" são as variáveis que estão na unit uClasses, que são as
        Tabelas com seus campos, indices e foreing keys. E também algum deles estão base
        base de conhecimento.
                    ------------------------------------------------
        O componente "Sistema de Consulta" é apresentado no formulário FormAnalise
        que possue as perguntas que são feitas aos usuários e o mesmo atua em cima
        do motor de inferência
        }


      //TLexico - Sua função principal é obter o próximo token do script
        TLexico = class
        public
          TpToken     : TTipoToken;                          { Tipo Token }
          LinhaAnalisada,                                    { Linha sendo analisada }
          Token       : string;                              { Contudo analizado }
          LinhaPos,                                          { Linha sendo analisada }
          Posicao,                                           { Posicao em LinhaAnalisada }
          Limite      : integer;                             { Tamanho da linha analisada }
          UltimaLinha : boolean;                             { Ultima linha da analise }
          Source      : TStringList;                         { Script a ser analisado }
          procedure   ObterToken;                              { Obtem o token e atualiza o
tipo do mesmo }
          procedure   ObterScript(Texto: string);              { Copia o script do texto da
tela principal para a variavel source }
          constructor Create;                                 { Cria o script que esta sendo
analisado em memoria e seta variaveis iniciais }
          destructor  Destroy; override;                     { Destroi o script }
        end;
```

```
       //TSintatico - Analisa se os tokens provenientes do analisador léxico estão corretos e
na ordem certa
         TSintatico = class
         public
           Lexico       : TLexico;                              { Possue as variaveis do lexico
que esta sendo analisado }
           ListaTabelas : TListaTabelas;                        { Possue a lista de tabelas }
           function Analisar : string;
           function ObterTabelas:string;
           function ObterFinal(var TabelaAux: TTabela): string;
           function ObterAtributos(var Campo : TCampo; var Soma: integer): string;
           function ObterIndice: string;
           function ObterFK: string;
           function ObterPK(var PK: string):string;
           function ObterTablespace(var Tabela: TTabela):string;
           function ObterRestricoes(var Campo: TCampo):string;
           function ObterValorDecimal(var Decimal: string): string;
           constructor  Create(fLexico : TLexico; fListaTabelas : TListaTabelas);
         end;
       //-------------------Fim Analisadores-----------------------
```

ANEXO III

```
[ATUALIZACAO]

0|10=10

1|10=5

2|10=2

0|50=15

1|50=10

2|50=7


[%SEMANAL]

0=10


[TAMANHO]

25|0|0|500= 25|0|0|500 PCTFREE 10 PCTUSED 40 STORAGE (INITIAL 128K NEXT 128K MINEXTENTS 1 MAXEXTENTS 4096 PCTINCREASE 0)

25|0|0|1000= 25|0|0|1000 PCTFREE 15 PCTUSED 40 STORAGE (INITIAL 128K NEXT 256K MINEXTENTS 1 MAXEXTENTS 4096 PCTINCREASE 0)
```

ANEXO IV

```
Function TSintatico.Analisar : string;
var
  Erro: string;
begin
  Result := '';
  Lexico.ObterToken;
  while Lexico.TpToken <> Final do begin
    if Lexico.Token = 'CREATE' then begin
      Lexico.ObterToken;
      if Lexico.Token = 'TABLE' then begin
        Erro := ObterTabelas;
        end
      else if Lexico.Token = 'INDEX' then begin
        Erro := ObterIndice;
        end
      else if Lexico.Token = 'UNIQUE' then begin
        Lexico.ObterToken;
        if Lexico.Token = 'INDEX' then begin
          Erro := ObterIndice;
          end
        else begin
          Erro := 'Esperando INDEX';
          end
        end
      else begin
        Result := 'Esperando TABLE ou INDEX';
        end;
      end
    else if Lexico.Token = 'ALTER' then begin
      Erro := ObterFK;
```

```
      end
    else begin
      Result := 'Esperando CREATE ou ALTER';
      Exit;
      end;
    if Erro <> '' then begin
      result := Erro;
      Exit;
      end
    else
      Lexico.ObterToken;
    end;
  if Erro = '' then
    FormAnalisar.PreencherTabelas;
end;
```

```
procedure TLexico.ObterToken;
var
  Aux  : string;
  State : byte;

  procedure PosicionaProximaLinha;
  begin
    if Source.Count = LinhaPos + 1 then begin
      UltimaLinha := True;
      State := 10;
      Exit;
      end;

    Posicao := 0;
    inc(LinhaPos);
    LinhaAnalisada:= Source.Strings[LinhaPos];
    Limite := length(LinhaAnalisada);
    State := 1;
```

```
  end;


begin
  Token := '';
  State := 1;
  if Posicao = Limite then
    State :=  10;
  while True do begin
    while (Length(LinhaAnalisada) = 0) and (not UltimaLinha) do
      PosicionaProximaLinha;
    case State of
      1:  begin
          inc(Posicao);
          if Posicao > Limite then
            State :=  10;
         case LinhaAnalisada[Posicao] of
              '0'..'9' : State := 2;
              'a'..'z' ,
              'A'..'Z' : State := 4;
              '.',';'  ,
              ',',':'  : State := 6;
              '(',')'  ,
              '+','|',
              '?','[',
              ']'      : State := 7;
              ''''     : State := 8;

              ' '      : State := 11;
              '"'      : State := 12;

              '=','>'  ,
              '<'      : State := 17;
```

```
        '/'      : State := 18;
        '-'      : State := 19;
   end;


end;
2: begin
   Aux := Aux + upcase(LinhaAnalisada[Posicao]);
   inc(Posicao);
   case LinhaAnalisada[Posicao] of
        '0'..'9' : State := 2;
        '/'      : State := 13;
        else       State := 3;
   end;
end;
3: begin
   TpToken := Numero;
   Token := Aux;
   Aux := Aux + upcase(LinhaAnalisada[Posicao]);
   Posicao := Posicao-1;
   Aux := '';
   Exit;
end;
4: begin
   Aux := Aux + upcase(LinhaAnalisada[Posicao]);
   inc(Posicao);
   case LinhaAnalisada[Posicao] of
        'a'..'z' ,
        'A'..'Z' : State := 4;
        '0'..'9' : State := 4;
        '_'      : State := 4;
        else       State := 5;
   end;
end;
```

```
5: begin
   TpToken := id;
   token := Aux;
   Posicao:=Posicao-1;
   Aux := '';
   exit;
end;
6: begin
   TpToken := Ponto;
   token := upcase(LinhaAnalisada[Posicao]);
   Aux := '';
   exit;
end;
7: begin
   TpToken := Simbolo;
   token := LinhaAnalisada[Posicao];
   Aux := '';
   exit;
end;
8: begin
   Aux := Aux + LinhaAnalisada[Posicao];
   inc(Posicao);
   case LinhaAnalisada[Posicao] of
        'a'..'z' ,
        'A'..'Z' : State := 9;
        else       State := 1;
   end;
end;
9: begin
   Aux := Aux + upcase(LinhaAnalisada[Posicao]);
   TpToken := letra;
   token := upcase(LinhaAnalisada[Posicao]);
   inc(Posicao);
```

```
    Aux := '';
    exit;
 end;
10: begin
  if UltimaLinha then begin
    TpToken := Final;
    exit;
    end;
    inc(LinhaPos);
    if Source.Count <= LinhaPos then // verifica se chegou no final do script
      begin
        TpToken := Final;
        UltimaLinha := True;
        exit;
      end
    else
      LinhaAnalisada := Source.Strings[LinhaPos];
    Limite := length(LinhaAnalisada);
    Posicao := 0;
    Aux := '';
    State := 1;
end;
11: begin
  State := 1;
end;
12: begin
    Aux:=Aux + upcase(LinhaAnalisada[Posicao]);
    inc(Posicao);
    case LinhaAnalisada[Posicao] of
         '"'   : State := 15;
         else    State := 12;
    end;
end;
```

```
13: begin
    Aux := Aux + upcase(LinhaAnalisada[Posicao]);
    inc(Posicao);
    case LinhaAnalisada[Posicao] of
         '0'..'9' : State := 13;
         '/'      : State := 13
         else       State := 14;
    end;
end;
14: begin
    TpToken:= Data;
    Posicao:=Posicao-1;
    token := Aux;
    Aux := '';
    exit;
end;
15: begin
    TpToken:= Literal;
    token:=Aux+upcase(LinhaAnalisada[Posicao]);
    Aux:='';
    exit;
end;
16: begin
    TpToken := id;
    token := Aux;
    Posicao:=Posicao-1;
    Aux := '';
    exit;
end;
17: begin
    Aux := Aux + upcase(LinhaAnalisada[Posicao]);
    inc(Posicao);
    case LinhaAnalisada[Posicao] of
```

```
          '>','<',
          '='      : State := 17;
          else       State := 16;
      end;
    end;

18: begin
      if Banco = sDrvMSSQL then begin
        inc(Posicao);
        while (Posicao > Limite) and (not UltimaLinha) do begin
          PosicionaProximaLinha;
          State :=  18;
          Posicao := 1;
          end;
        if not UltimaLinha then begin
          case LinhaAnalisada[Posicao-1] of
                 '/'       :begin
                              State := 18;
                             end;
                 '*'       :begin
                              if LinhaAnalisada[Posicao] = '/' then
                                State := 1
                              else
                                State := 18;
                             end;
                 else      begin
                              State := 18;
                            end;
            end;
          end
        else
          State := 10;
        end
```

```
          else begin
            Aux := Aux + upcase(LinhaAnalisada[Posicao]);
            Token:=Aux;
            exit;
            end;
         end;
      19: begin
          if Banco = sDrvORACLE then begin
            inc(Posicao);
            case LinhaAnalisada[Posicao] of
                   '-'       :begin
                              PosicionaProximaLinha
                             end;
                   else begin
                     Aux := Aux + upcase(LinhaAnalisada[Posicao]);
                     exit;
                     end;
              end;
            end
          else begin
            Aux := Aux + upcase(LinhaAnalisada[Posicao]);
            exit;
            end;
         end;
    end;
  end;
end;
```

REFERÊNCIAS

AHO, Alfred V, SETHI, Ravi, ULMAN, Jeffrey D**. Compiladores, princípios, técnicas e ferramentas**. Tradução de Daniel de Ariosto Pinto. Rio de Janeiro: Livros Técnicos e Científicos, 1995.

DATE, C. J. **Introdução a sistemas de bancos de dados**. Rio de Janeiro : Campus, 2000.

FERNANDES, Lucia. **Oracle para desenvolvedores**: Oracle developer, Oracle 8-8i, curso completo. Rio de Janeiro: Axcel Books, 2000. xciii, 1758p.

FISHER, Alan S. **Case :** utilização de ferramentas para desenvolvimento de software. Rio de Janeiro: Campus, 1990.

GANE, Chris. **CASE**: o relatório Gane. Rio de Janeiro: LCT – Livros Técnicos e Científicos, 1990.

GENARO, Sergio.**Sistemas Especialistas:** O conhecimento Arficial. São Paulo: LTC – Livros Técnicos e Científicos Editora AS, 1986.

HEINZLE, Roberto. **Protótipo de uma ferramenta para criação de sistema especialista baseado em regras de produção.** 1995. 145 f. Dissertação (mestrado em engenharia de produção e sistemas) – Setor de Engenharia da Produção e Sistemas, Universidade Federal de Santa Catarina, Florianópolis.

IMASTER. **Fórum de discussão**, [2002]. Disponível em: <http://www.imasters.com.br/web/colunistas/sql/23.asp>. Acesso em: 20 nov. 2002.

JOSÉ NETO, João. **Introdução à compilação**. Rio de Janeiro: Livros Técnicos e Científicos, 1987.

KAMKE, Claudiomar, CARLI, Wilson Pedro. **Protótipo de sistema especialista para padronização de modelo de dados baseado no Oracle Designer 2000**. , 2001. viii, 69p.

LEVINE, Robert I; DRANG, Diane E; EDELSON, Barry. **Inteligência artificial e sistemas especialistas**. São Paulo: McGraw-Hill, 1988.

MARTIN, James. **Técnicas estruturadas e case**. São Paulo: Makron, 1991.

MICROSOFT. **SQL Server books online**. Redmond: Microsoft, 1998.

PRICE, Ana Maria de Alencar; TOSCANI, Simão Sirineo Toscani. **Implementação de linguagens de programação:** compiladores. Porto Alegre: Sagra Luzzato, 2001.

RABUSKE, Renato Antônio. **Inteligência artificial.** Florianópolis: Editora da UFSC, 1995.

RAMALHO, José Antônio. **Oracle 8i.** São Paulo: Berkeley Brasil,1999.

RAUTENBERG, Sandro, HUGO, Marcel. **Um protótipo de sistema especialista difuso para definição de salários por habilidades**. 1996. x, 71p.

SEBESTA, Robert W. **Conceitos de linguagens de programação**. 4. ed. Tradução de José Carlos Barbosa dos Santos. Porto Alegre: Bookman, 2000.

VALDAMERI, Alexander R. **Apostila de banco de dados**. Apostila disponibilizada pelo professor da disciplina de Banco de Dados 2 do Curso de Ciências da Computação da Universidade Regional de Blumenau, Blumenau, abr. 2002.