UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIAS DA COMPUTAÇÃO

(Bacharelado)

PROTÓTIPO DE UM SISTEMA DE HOMOLOGAÇÃO DE PROJETOS FÍSICOS DE BANCO DE DADOS DESENVOLVIDOS NA FERRAMENTA ORACLE “DESIGNER 6I”

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

JAIR PAULO SATIG

BLUMENAU, JUNHO/2003

2003/1-35

PROTÓTIPO DE UM SISTEMA DE HOMOLOGAÇÃO DE PROJETOS FISICOS DE BANCO DE DADOS DESENVOLVIDOS NA FERRAMENTA ORACLE “DESIGNER 6I”

JAIR PAULO SATIG

ESTE TRABALHO DE CONCLUSÃO DE CURSO, FOI JULGADO ADEQUADO PARA OBTENÇÃO DOS CRÉDITOS NA DISCIPLINA DE TRABALHO DE CONCLUSÃO DE CURSO OBRIGATÓRIA 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. Everaldo Artur Grahl

_____

Prof. Maurício Capobianco Lopes

AGRADECIMENTOS

Em primeiro lugar agradeço ao grande criador Deus, pois sem ele, jamais estaria concluindo esta etapa.

Um agradecimento especial a minha esposa Solange e ao meu filhinho Gustavo que chegou quase na reta final para me dar alegrias e força , e também aos meus pais Arlindo e Lori, que em muitas oportunidades me deram forças e incentivos para que eu jamais desistisse dessa etapa que hoje está sendo concluída.

Agradeço também os meus amigos Adriano Moretti, Jair de Souza, Sandro Ferrari e Débora Nazário, que estiveram comigo algumas horas durante toda a caminhada da faculdade sempre dando incentivos para que jamais desistisse do caminho por min escolhido.

A todos os professores que contribuíram para minha formação. Ao meu orientador Alexander R. Valdameri pelo apoio e incentivo dado neste semestre para a realização deste trabalho.

Por último agradeço a todos que direta ou indiretamente cruzaram o meu caminho, tornando a minha vida digna de ser vivida, e dizer Valeu a Pena!!!

Obrigado a todos.

LISTA DE FIGURAS

LISTA DE QUADROS

RESUMO

Este trabalho de conclusão de curso descreve a construção de um protótipo para homologação de bases de dados desenvolvidas na ferramenta *Oracle Designer 6.i*. Para tanto foi desenvolvido um software em ambiente *Visual Basic.Net* para comparar o modelo de dados físicos com os padrões de desenvolvimento pesquisados, identificando os erros cometidos e gerando relatórios com as respectivas informações.

ABSTRACT

This work of course conclusion describes the construction of an prototype to homologate databases developed using the tool Oracle Designer 6.i. For that purpose was developed a software using visual basic.Net to compare the model of physical data with the searched methodology seeking identify the committed errors and to extract a report that will show the found divergences.

SUMÁRIO

1 INTRODUÇÃO

Durante os últimos anos tem se verificado um crescimento substancial da quantidade de dados armazenados em meios magnéticos. Estes dados, produzidos e armazenados em larga escala, são inviáveis de serem lidos ou analisados por especialistas através de métodos manuais tradicionais, tais como planilhas de cálculos e relatórios informativos operacionais. Por outro lado, sabe-se que grandes quantidades de dados equivalem a um maior potencial de informação.

Entretanto, as informações contidas nos dados operacionais, não interessam quando estudados individualmente. Diante deste cenário, surge a necessidade de se explorar estes dados para extrair informação, conhecimento implícito e utilizá-lo no âmbito do problema. Argumenta-se que a necessidade de sistemas para dar suporte a decisão têm crescido ao longo dos anos cada vez mais dentro da granularidade de informações mais refinadas, da seguinte maneira: nos anos 70, em nível de nichos, grupos de interesse; nos anos 80, em nível de segmentos de mercado; e nos anos 90, em nível de clientes.

Este último nível, naturalmente, requer o uso de uma quantidade maior de dados para se extrair o conhecimento. A exploração do valor destes dados, ou seja, a informação neles contida implicitamente, depende de técnicas como regras de associação, classificação, entre outras capazes de gerenciar tarefas complexas. Essas tarefas podem ser realizadas com um banco de dados.

Para armazenar dados de forma consistente e gerenciar tarefas complexas, e de fácil recuperação pode se utilizar um banco de dados relacional. Segundo Harrington (2002), pode-se definir banco de dados relacional como uma coleção de dados relacionados, onde dados significam fatos conhecidos que podem ser armazenados e que possuem significado implícito. Neste contexto está inserido o banco de dados Oracle.

Um banco de dados Oracle tem uma estrutura física e lógica. Como essas estruturas no servidor são separadas, o armazenamento físico dos dados pode ser gerenciado sem afetar o acesso às estruturas lógicas de armazenamento. A estrutura física do banco de dados Oracle é determinada pelos arquivos do sistema operacional que o constituem. Cada banco de dados Oracle é formado por três tipos de arquivos: um ou mais *datafiles*, dois ou mais arquivos de registro *redo* e um ou mais arquivos de controle.

A estrutura lógica do Oracle é determinada por um ou mais *tablespaces* (espaços lógicos de armazenamento) e pelos objetos de esquema de banco de dados. Um esquema é uma coleção de objetos que por sua vez são as estruturas lógicas que se referem diretamente aos dados do banco de dados. Os objetos de esquema incluem estruturas tais como: tabelas, visões, seqüências, procedimentos armazenados, sinônimos, índices, agrupamentos e *links* de banco de dados.

As estruturas de armazenamento lógico incluindo *tablespaces*, os segmentos e as extensões, determinam como é usado o espaço físico de um banco de dados. Os objetos de esquema e os relacionamentos entre eles formam o projeto relacional de um banco de dados.

O Oracle fornece a capacidade de armazenar e acessar os dados de forma consistente com um modelo definido conhecido como modelo relacional. Em conseqüência disso, o Oracle é caracterizado como sistema de gerenciamento de banco de dados relacional (SGBDR) (Oracle 2000).

Os dados de um banco de dados relacional são armazenados em tabelas. As tabelas relacionais são definidas pelas suas colunas e recebem um nome. Em seguida, os dados são armazenados como linhas na tabela. Elas podem estar relacionadas entre si e o banco de dados pode ser usado para garantir esses relacionamentos.

Em função dos altos custos e da grande quantidade de tempo exigida pelas atividades dos administradores de dados na revisão dos modelos de dados físicos e lógicos, estas atividades muitas vezes são negligenciadas ou reduzidas e, conseqüentemente, é comum a entrega do software para seu usuário com a presença de defeitos não revelados prejudicando assim a qualidade do desenvolvimento.

A preocupação com a qualidade de software está se tornando cada vez maior em função do grande volume de software produzido atualmente e a exigência cada vez maior de seus usuários para que produza os resultados esperados sem erros ou falhas.

Para minimizar esses efeitos o trabalho propõe-se pesquisar e definir um padrão de modelagem de dados, visando assim desenvolver uma ferramenta que irá realizar uma varredura no modelo de dados físico do banco de dados Oracle, mais conhecido como *metadados,* comparando as estruturas desenvolvidas com o padrão definido, e após o processamento, o sistema emitirá, relatórios que indiquem divergências no modelo criado, permitindo assim ao programador consertar os erros no próprio modelo de dados, antes que a

interface do sistema/programa seja desenvolvida, objetivando uma melhora significativa de qualidade no desenvolvimento do restante do sistema.

1.1 OBJETIVOS

O objetivo principal desse trabalho é desenvolver um protótipo de um sistema de homologação de modelos de dados físicos desenvolvidos com a ferramenta *Oracle Designer 6i*.

Os objetivos específicos do trabalho são:

a) investigar o metadados do SGBD ORACLE em sua estrutura de armazenamento no aspecto de integridade de dados;
b) pesquisar e definir e padrões para o desenvolvimento de banco de dados Oracle;
c) definir um glossário de termos de acordo com as divergências encontradas em relação a padrão de desenvolvimento;
d) gerar relatórios das divergências ocorridas durante o processo de homologação.

1.2 ORGANIZAÇÃO DO TRABALHO

O capítulo 1 introduz o contexto geral do trabalho dividido nos segmentos de introdução, objetivos e área do TCC proposto.

O capítulo 2 enfoca projeto de banco de dados e os modelos existentes.

O capitulo 3 fala sobre o sistema gerenciador de banco de dados Oracle e seu metadados, demostrando também o modelo de entidades que serão utilizadas para o processo de homologação

O capítulo 4 descreve o estudo sobre os padrões de desenvolvimento de bases de dados usando o banco de dados Oracle.

No capítulo 5 encontra-se a definição das ferramentas, que foram utilizadas para este trabalho, a especificação e implementação do protótipo, bem como os resultados obtidos, item no qual é aplicado um modelo de dados físico para ser validado pelo protótipo.

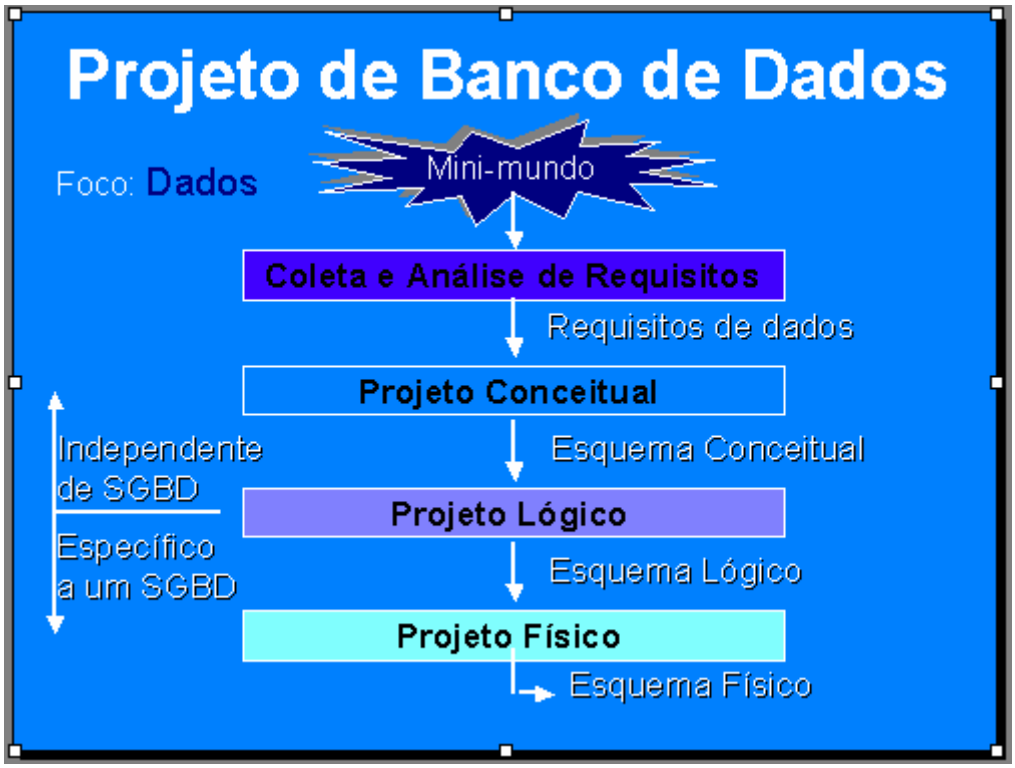
O capítulo 6 relata as considerações finais, limitações, e sugestões para continuidade de trabalhos futuros.

2 PROJETO DE BANCO DE DADOS

Segundo Cericola (1995), um banco de dados é uma coleção de dados organizados e integrados, armazenados em formas de tabelas interligadas através de chaves primárias e estrangeiras, podendo ser utilizada por todas as aplicações relevantes sem duplicação de dados, e sem a necessidade de serem definidos em programas, pois utiliza as definições existentes nas bases de dados, através do dicionário de dados ativo e dinâmico.

Para se criar um banco de dados robusto e confiável, existem roteiros que auxiliam, indicando passo a passo todas as atividades a serem desenvolvidas. Uma demonstração pode ser vista na fig. 01.

Figura 01 - Projeto de banco de dados



Fonte: Valdameri (2003)

Conforme mostra a fig. 01, o *Mini-Mundo* seria o problema ao qual estaria sendo formulada a base de dados. A partir deste problema devem ser levantados os requisitos que compõem o sistema. Esses requisitos podem ser definidos como as necessidades do negócio ao qual estará se modelando o banco de dados. Após a coleta e análise de requisitos, vem o projeto conceitual, em seguida o projeto lógico e finalmente o projeto físico. Esses projetos

são normalmente representados por modelos de banco de dados, sendo assim para representar o projeto conceitual poderá se usar o modelo conceitual, para o *projeto lógico* pode-se usar o *modelo lógico* e para o projeto físico o *modelo físico.*

2.1 MODELOS DE BANCO DE DADOS

Segundo Heuser (2000), um modelo de banco de dados é uma descrição dos tipos de informações que estão armazenadas em um banco de dados. Para construir um modelo de dados, usa-se uma linguagem de modelagem de dados. Linguagens de modelagem de dados podem ser classificadas de acordo com a forma de apresentar modelos, podem ser tanto em linguagens textuais ou gráficas. Um mesmo modelo de dados pode ser apresentado de várias formas. Cada representação do modelo recebe a denominação de esquema de banco de dados.

De acordo com a intenção do modelador, um banco de dados pode ser modelado em vários níveis de abstração. Um modelo de dados que servirá para explicar a um usuário leigo em informática qual é a organização de um banco de dados provavelmente não conterá detalhes sobre a representação em meio físico das informações. Já um modelo de dados usado por um técnico para otimizar a performance de acesso ao banco de dados conterá mais detalhes de como as informações estão organizadas internamente e portanto será menos abstrato.

Normalmente em projetos de banco de dados são considerados dois níveis de abstração de modelos de dados: modelo conceitual e modelo lógico. Um terceiro nível também pode ser adotado, o modelo físico, que é objeto de pesquisa do presente trabalho.

2.1.1 MODELO CONCEITUAL

Um modelo conceitual é uma descrição do banco de dados de forma independente de implementação em um SGBD. O modelo conceitual registra que dados podem aparecer no banco de dados, mas não registra como estes dados estão armazenados no SGBD.

Conforme Heuser (2000), um modelo conceitual é igual a um modelo de dados abstrato, que descreve a estrutura de um banco de dados de forma independente de um SGBD particular.

A técnica de modelagem conceitual mais difundida é a abordagem entidade-relacionamento (ER). Nesta técnica, um modelo conceitual é usualmente representado através de um diagrama chamado: diagrama entidade-relacionamento (DER).

2.1.2 MODELO LÓGICO

Um modelo lógico é uma descrição de um banco de dados no nível de abstração visto pelo usuário do SGBD. Assim, o modelo lógico é dependente do tipo particular do SGBD que está sendo usado.

O modelo lógico de um banco de dados relacional deve definir quais as tabelas que o banco contém, e para cada tabela, os nomes das colunas. O modelo lógico descreve a estrutura do banco de dados conforme vista pelo usuário do SGBD.

2.1.3 MODELO FÍSICO

O modelo físico é uma descrição das estruturas físicas de armazenamento do banco de dados, e deve conter tabelas visões, índices, chaves primárias, chaves únicas, chaves estrangeiras, sinônimos, tipo de dados e o tamanho dos campos.

3 ORACLE

Conforme Ault (1995), o Oracle é um sistema de gerenciamento de banco de dados relacional (SGBDR), com funções completas baseadas no padrão ANSI SQL3. Agrega os seguintes componentes: gerador de aplicações, dicionário de dados, gerador de relatórios, planilhas eletrônicas, pré-compiladores para linguagens hospedeiras, utilitários gráficos, cálculos estatísticos, armazenagem de dados de vários tipos, como: data, números inteiros, valores numéricos com ou sem decimais, ponto flutuante, caracteres variáveis ou fixos, binários, textos, som, imagem etc.

Um SGBDR armazena dados em tabelas chamadas de relações, as quais são representação bidimensional de dados, onde as linhas, chamadas de tuplas no jargão relacional, representam registros, e as colunas, chamadas de atributos, são as partes da informação contidas no registro.

Os três principais aspectos do modelo relacional, no qual o Oracle se baseia, são as estruturas, as operações e as regras de integridade (Fernandes 2000).

As estruturas são os objetos que guardam os dados de um banco de dados. As estruturas e os dados podem ser manipulados através de operações.

As operações são as ações que permitem aos usuários manipularem os dados e as estruturas de um banco de dados. Essas operações devem aderir a um conjunto de regras de integridade pré-definidas.

As regras de integridade são leis que governam quais operações são permitidas nos dados e nas estruturas de um banco de dados, com o propósito de protegê-los de operações indevidas (Sherer, 2000).

Além dessas características, está disponível numa grande variedade de computadores de grande porte, minis e micros, por possuir portabilidade com vários fabricantes. Tem uma arquitetura distribuída com execução bifásica automática, que permite às aplicações residirem em vários computadores e se comunicarem entre si, de forma transparente ao usuário. Hoje tem portabilidade para mais de 80 plataformas, incluindo 40+ Unix, VMS, MVS, VM, HP,

MPE/SL, Siemens, MS-DOS, MS-WINDOWS, ICL, OS/2, Macintosh, Rede Novell, Lan Manager, etc.

Segundo Cericola (1995), o SGBD Oracle foi desenvolvido nos Estados Unidos da América em 1977 pela Oracle Corporation. Esse produto começou a ser desenvolvido como um projeto da Nasa, e sua primeira instalação comercial ocorreu em junho de 1979.

Uma das grandes características da Oracle é estar acompanhando o mercado da informática com todas as novidades exeqüíveis que estão gradualmente sendo introduzidas, tais como produtos multimídia, CAD/CAM (Computer Aided Design/Computer Aided Manufacturing), gráficos, textos, desenhos, fotografias, som e imagem.

3.1 METADADOS ORACLE

Conforme Borbinha (2002), metadados significa informação estruturada sobre recursos de informação, artefatos ou serviços. Nesta perspectiva, pode-se considerar que os metadados são informações que resumem, enriquecem e complementam os objetos ou serviços referenciados, produzindo assim um referencial incremento da informação.

O trabalho propõe-se a investigar o metadados do SGBD ORACLE em sua estrutura de armazenamento no aspecto de integridade de dados. O metadados a qual o trabalho faz referência está contido no repositório da ferramenta *Case Designer 6.i.* A arquitetura do repositório Oracle está baseada na tecnologia do *Oracle8i*. O *Oracle Repository* é seguro, robusto e escalável.

Uma arquitetura segura e escalável é essencial para protejer e distribuir o conteúdo do repositório, à medida que o metadados (a informação descritiva sobre os dados) será mais provavelmente usado por muitas pessoas em toda a empresa. O fraco desempenho do sistema, ou falta de disponibilidade do sistema, reduziria ou mesmo eliminaria o valor que pode ser ganho pelo uso do repositório (Oracle, 2000).

O repositório em si mesmo consiste de um grande número de objetos da base de dados Oracle; tabelas, índices, visões, packages, constraints, sequences, sinônimos e triggers. Os componentes do repositório podem ser agrupados dentro de três tipos principais:

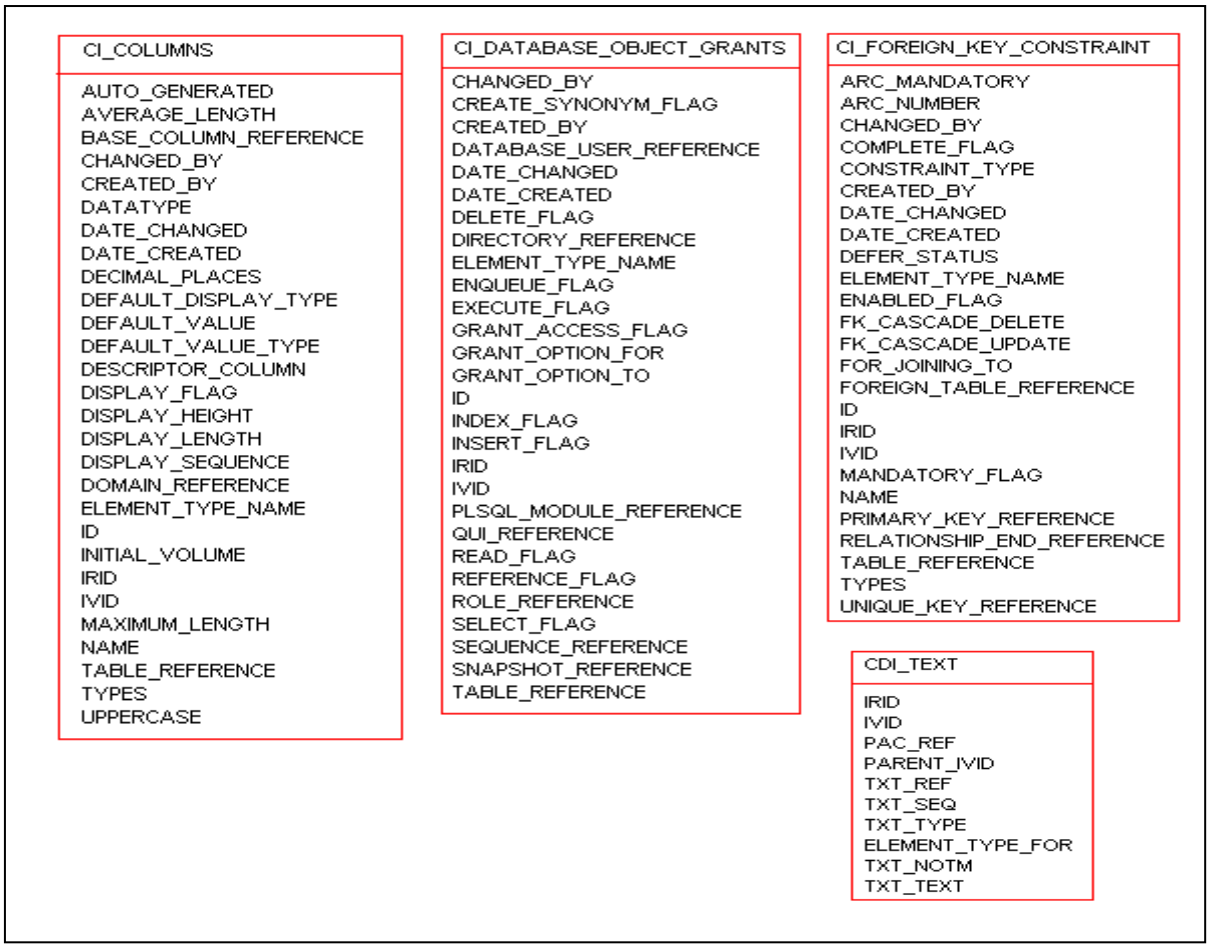
a) metadados: definição da tabela de dados que armazena a informação descritiva sobre os dados da empresa;

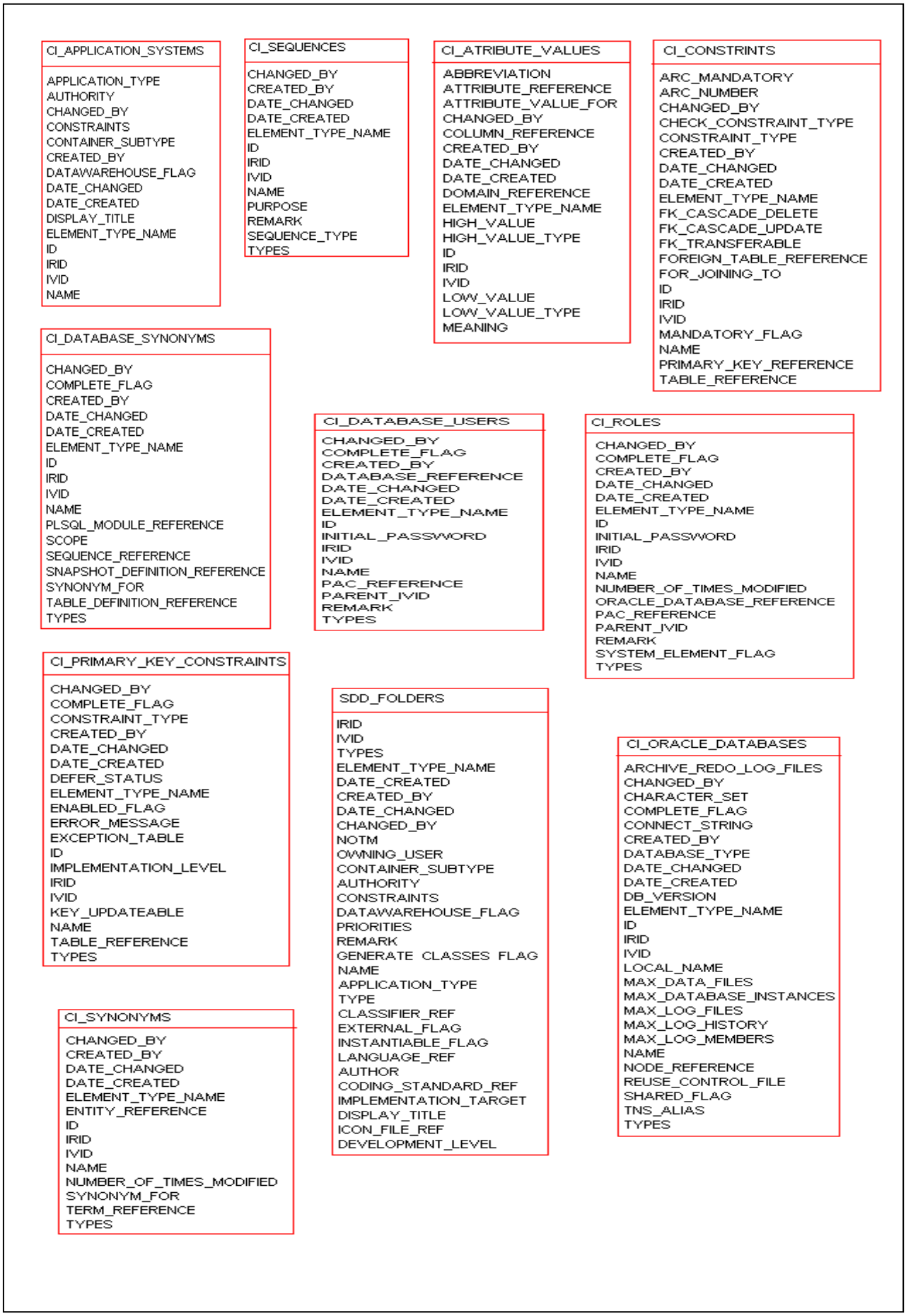
b) instância de dados: dados entrados pelos usuários;

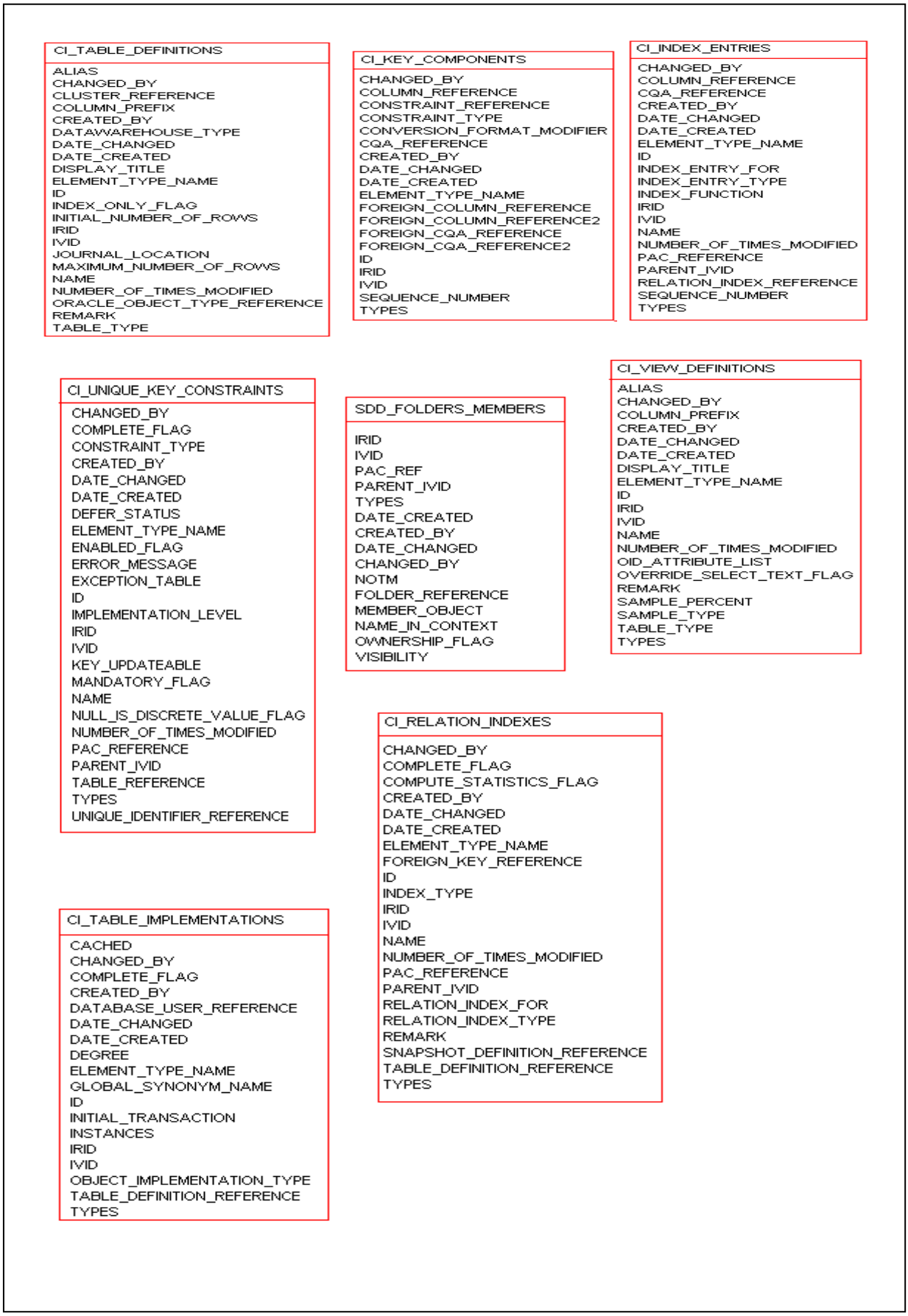
c) interface programável da aplicação: um conjunto de visões SQL da base de dados e packages PL/SQL, funções e procedimentos, classes Java e métodos que possibilitam aos usuários interagir diretamente com o conteúdo do repositório.

Para o desenvolvimento do trabalho foi necessário ivestigar quais são os objetos do repositório que armazenam os dados do modelo físico. Esses objetos foram identificados e sobre esses objetos serão desenvolvidas as *Stored Procedures,* que identificam as divergências do modelo físico. As tabelas do metadados do Oracle não possuem relacionamento, tendo em vista que a presença de chaves estrangeiras é implicita. No quadro 01 pode-se observar os objetos do repositório que serão lidos pelo protótipo.

Quadro 01 – Esquema de objetos do repositório.




CI_APPLICATION_SYSTEMS
APPLICATION_TYPE
AUTHORITY
CHANGED_BY
CONSTRAINTS
CONTAINER_SUBTYPE
CREATED_BY
DATAWAREHOUSE_FLAG
DATE_CHANGED
DATE_CREATED
DISPLAY_TITLE
ELEMENT_TYPE_NAME
ID
IRID
IVID
NAME
CI_SEQUENCES
CHANGED_BY
CREATED_BY
DATE_CHANGED
DATE_CREATED
ELEMENT_TYPE_NAME
ID
IRID
IVID
NAME
PURPOSE
REMARK
SEQUENCE_TYPE
TYPES
CI_ATRIBUTE_VALUES
ABBREVIATION
ATTRIBUTE_REFERENCE
ATTRIBUTE_VALUE_FOR
CHANGED_BY
COLUMN_REFERENCE
CREATED_BY
DATE_CHANGED
DATE_CREATED
DOMAIN_REFERENCE
ELEMENT_TYPE_NAME
HIGH_VALUE
HIGH_VALUE_TYPE
ID
IRID
IVID
LOW_VALUE
LOW_VALUE_TYPE
MEANING
CI_CONSTRINTS
ARC_MANDATORY
ARC_NUMBER
CHANGED_BY
CHECK_CONSTRAINT_TYPE
CONSTRAINT_TYPE
CREATED_BY
DATE_CHANGED
DATE_CREATED
ELEMENT_TYPE_NAME
FK_CASCADE_DELETE
FK_CASCADE_UPDATE
FK_TRANSFERABLE
FOREIGN_TABLE_REFERENCE
FOR_JOINING_TO
ID
IRID
IVID
MANDATORY_FLAG
NAME
PRIMARY_KEY_REFERENCE
TABLE_REFERENCE
CI_DATABASE_SYNONYMS
CHANGED_BY
COMPLETE_FLAG
CREATED_BY
DATE_CHANGED
DATE_CREATED
ELEMENT_TYPE_NAME
ID
IRID
IVID
NAME
PLSQL_MODULE_REFERENCE
SCOPE
SEQUENCE_REFERENCE
SNAPSHOT_DEFINITION_REFERENCE
SYNONYM_FOR
TABLE_DEFINITION_REFERENCE
TYPES
CI_DATABASE_USERS
CHANGED_BY
COMPLETE_FLAG
CREATED_BY
DATABASE_REFERENCE
DATE_CHANGED
DATE_CREATED
ELEMENT_TYPE_NAME
ID
INITIAL_PASSWORD
IRID
IVID
NAME
PAC_REFERENCE
PARENT_IVID
REMARK
TYPES
CI_ROLES
CHANGED_BY
COMPLETE_FLAG
CREATED_BY
DATE_CHANGED
DATE_CREATED
ELEMENT_TYPE_NAME
ID
INITIAL_PASSWORD
IRID
IVID
NAME
NUMBER_OF_TIMES_MODIFIED
ORACLE_DATABASE_REFERENCE
PAC_REFERENCE
PARENT_IVID
REMARK
SYSTEM_ELEMENT_FLAG
TYPES
CI_PRIMARY_KEY_CONSTRAINTS
CHANGED_BY
COMPLETE_FLAG
CONSTRAINT_TYPE
CREATED_BY
DATE_CHANGED
DATE_CREATED
DEFER_STATUS
ELEMENT_TYPE_NAME
ENABLED_FLAG
ERROR_MESSAGE
EXCEPTION_TABLE
ID
IMPLEMENTATION_LEVEL
IRID
IVID
KEY_UPDATEABLE
NAME
TABLE_REFERENCE
TYPES
SDD_FOLDERS
IRID
IVID
TYPES
ELEMENT_TYPE_NAME
DATE_CREATED
CREATED_BY
DATE_CHANGED
CHANGED_BY
NOTM
OWNING_USER
CONTAINER_SUBTYPE
AUTHORITY
CONSTRAINTS
DATAWAREHOUSE_FLAG
PRIORITIES
REMARK
GENERATE CLASSES FLAG
NAME
APPLICATION_TYPE
TYPE
CLASSIFIER_REF
EXTERNAL_FLAG
INSTANTIABLE_FLAG
LANGUAGE_REF
AUTHOR
CODING_STANDARD_REF
IMPLEMENTATION_TARGET
DISPLAY_TITLE
ICON_FILE_REF
DEVELOPMENT_LEVEL
CI_ORACLE_DATABASES
ARCHIVE_REDO_LOG_FILES
CHANGED_BY
CHARACTER_SET
COMPLETE_FLAG
CONNECT_STRING
CREATED_BY
DATABASE_TYPE
DATE_CHANGED
DATE_CREATED
DB_VERSION
ELEMENT_TYPE_NAME
ID
IRID
IVID
LOCAL_NAME
MAX_DATA_FILES
MAX_DATABASE_INSTANCES
MAX_LOG_FILES
MAX_LOG_HISTORY
MAX_LOG_MEMBERS
NAME
NODE_REFERENCE
REUSE_CONTROL_FILE
SHARED_FLAG
TNS_ALIAS
TYPES
CI_SYNONYMS
CHANGED_BY
CREATED_BY
DATE_CHANGED
DATE_CREATED
ELEMENT_TYPE_NAME
ENTITY_REFERENCE
ID
IRID
IVID
NAME
NUMBER_OF_TIMES_MODIFIED
SYNONYM_FOR
TERM_REFERENCE
TYPES


CI_TABLE_DEFINITIONS
ALIAS
CHANGED_BY
CLUSTER_REFERENCE
COLUMN_PREFIX
CREATED_BY
DATAWAREHOUSE_TYPE
DATE_CHANGED
DATE_CREATED
DISPLAY_TITLE
ELEMENT_TYPE_NAME
ID
INDEX_ONLY_FLAG
INITIAL_NUMBER_OF_ROWS
IRID
IVID
JOURNAL_LOCATION
MAXIMUM_NUMBER_OF_ROWS
NAME
NUMBER_OF_TIMES_MODIFIED
ORACLE_OBJECT_TYPE_REFERENCE
REMARK
TABLE_TYPE
CI_KEY_COMPONENTS
CHANGED_BY
COLUMN_REFERENCE
CONSTRAINT_REFERENCE
CONSTRAINT_TYPE
CONVERSION_FORMAT_MODIFIER
CQA_REFERENCE
CREATED_BY
DATE_CHANGED
DATE_CREATED
ELEMENT_TYPE_NAME
FOREIGN_COLUMN_REFERENCE
FOREIGN_COLUMN_REFERENCE2
FOREIGN_CQA_REFERENCE
FOREIGN_CQA_REFERENCE2
ID
IRID
IVID
SEQUENCE_NUMBER
TYPES
CI_INDEX_ENTRIES
CHANGED_BY
COLUMN_REFERENCE
CQA_REFERENCE
CREATED_BY
DATE_CHANGED
DATE_CREATED
ELEMENT_TYPE_NAME
ID
INDEX_ENTRY_FOR
INDEX_ENTRY_TYPE
INDEX_FUNCTION
IRID
IVID
NAME
NUMBER_OF_TIMES_MODIFIED
PAC_REFERENCE
PARENT_IVID
RELATION_INDEX_REFERENCE
SEQUENCE_NUMBER
TYPES
CI_UNIQUE_KEY_CONSTRAINTS
CHANGED_BY
COMPLETE_FLAG
CONSTRAINT_TYPE
CREATED_BY
DATE_CHANGED
DATE_CREATED
DEFER_STATUS
ELEMENT_TYPE_NAME
ENABLED_FLAG
ERROR_MESSAGE
EXCEPTION_TABLE
ID
IMPLEMENTATION_LEVEL
IRID
IVID
KEY_UPDATEABLE
MANDATORY_FLAG
NAME
NULL_IS_DISCRETE_VALUE_FLAG
NUMBER_OF_TIMES_MODIFIED
PAC_REFERENCE
PARENT_IVID
TABLE_REFERENCE
TYPES
UNIQUE_IDENTIFIER_REFERENCE
SDD_FOLDERS_MEMBERS
IRID
IVID
PAC_REF
PARENT_IVID
TYPES
DATE_CREATED
CREATED_BY
DATE_CHANGED
CHANGED_BY
NOTM
FOLDER_REFERENCE
MEMBER_OBJECT
NAME_IN_CONTEXT
OWNERSHIP_FLAG
VISIBILITY
CI_VIEW_DEFINITIONS
ALIAS
CHANGED_BY
COLUMN_PREFIX
CREATED_BY
DATE_CHANGED
DATE_CREATED
DISPLAY_TITLE
ELEMENT_TYPE_NAME
ID
IRID
IVID
NAME
NUMBER_OF_TIMES_MODIFIED
OID_ATTRIBUTE_LIST
OVERRIDE_SELECT_TEXT_FLAG
REMARK
SAMPLE_PERCENT
SAMPLE_TYPE
TABLE_TYPE
TYPES
CI_RELATION_INDEXES
CHANGED_BY
COMPLETE_FLAG
COMPUTE_STATISTICS_FLAG
CREATED_BY
DATE_CHANGED
DATE_CREATED
ELEMENT_TYPE_NAME
FOREIGN_KEY_REFERENCE
ID
INDEX_TYPE
IRID
IVID
NAME
NUMBER_OF_TIMES_MODIFIED
PAC_REFERENCE
PARENT_IVID
RELATION_INDEX_FOR
RELATION_INDEX_TYPE
REMARK
SNAPSHOT_DEFINITION_REFERENCE
TABLE_DEFINITION_REFERENCE
TYPES
CI_TABLE_IMPLEMENTATIONS
CACHED
CHANGED_BY
COMPLETE_FLAG
CREATED_BY
DATABASE_USER_REFERENCE
DATE_CHANGED
DATE_CREATED
DEGREE
ELEMENT_TYPE_NAME
GLOBAL_SYNONYM_NAME
ID
INITIAL_TRANSACTION
INSTANCES
IRID
IVID
OBJECT_IMPLEMENTATION_TYPE
TABLE_DEFINITION_REFERENCE
TYPES

A seguir o detalhamento das tabelas e funções do repositório:

- sdd_folders: essa tabela contém todos os modelos físicos criados no repositório;
- ci_columns: essa tabela contém todas as colunas criadas para as tabelas e as visões;
- ci_constraints: essa tabela contém todas as regras de integridade do modelo, para as tabelas visões;
- ci_key_components: essa tabela contem as informações dos campos que contem a chave primária;
- ci_domains: essa tabela armazena todas os domínios criados no repositório;
- cdi_text: essa tabela armazena todas as descrições e comentários das colunas, tabelas e visões;
- ci_table_definitions: essa tabela contém a definição de todas as tabelas criadas nos modelos existentes no repositório;
- ci_view_definitions: essa tabela contem a definição de todas as visões criadas nos modelos existentes no repositório;
- sdd_folder_members: essa tabela contém as chaves de todos os objetos que pertencem ao modelo de dados;
- ci_attribute_values:essa tabela contém a lista de valores definida para os domínios e as suas descrições;
- ci_foreign_key_constraints: essa tabela contém todas as chaves estrangeiras definidas para as tabelas e as visões;
- ci_index_entries: essa tabela contém a definição dos índices;
- ci_relation_indexes: essa tabela contém os campos da tabela ou visão que fazem parte do índice;
- ci_database_object_grants: essa tabela contém as permissões definidas para cada objetos;
- ci_primary_key_constraints: essa tabela contém todas as chaves primárias definidas para as tabelas e as visões;
- ci_roles: essa tabela contém a definição de todas as roles definidas no repositório;
- ci_sequences: essa tabela contém todas as seqüências definidas no modelo físico;
- ci_sequence_implementations: essa tabela contem as informações a qual objeto essa seqüência pertence;

- ci_database_synonyms: essa tabela contém todos os sinônimos do modelo e
- para qual objeto eles pertencem;
- ci_unique_key_constraints: essa tabela contém a definição das chaves únicas e para qual objeto elas pertencem;
- ci_view_definitions: essa tabela contém a definição da visão;
- ci_view_implementations: essa tabela contém em qual base de dados o objeto ser criado;
- ci_oracle_database: essa tabela contém todos os bancos de dados definidos;
- ci_database_users: essa tabela contém todos os usuários definidos para os bancos de dados.

4 PADRÕES DE DESENVOLVIMENTO DE BANCO DE DADOS E NOMENCLATURA DE OBJETOS

Neste capítulo serão abordadas todas as regras para o desenvolvimento de banco de dados Oracle, passos e padrões que deverão ser seguidos para que o protótipo não encontre divergências no modelo de dados físicos.

Para dar clareza ao modelo de dados, o desenvolvedor deverá se preocupar em descrever com objetividade os objetos que compõem o novo modelo (tabela, visão, *snapshot*, campo etc).

Os termos utilizados nas nomenclaturas de tabelas, visões, *snapshots* e campos deverão ser separados por *Underline* ("_") e ter registro no glossário de termos. Sendo assim o desenvolvedor deverá consultar o glossário de termos antes de definir os nomes de seus objetos. Se o nome não existir, o desenvolvedor deverá consultar o Administrador de Dados para verificar a possibilidade de se criar o termo de acordo com a sua necessidade.

Deve-se procurar utilizar sempre o termo por extenso, abreviando apenas os termos com mais de 10 caracteres ou quando o nome do objeto exceder a 30 caracteres.

A tabela 01 apresenta as regras para criação e nomeação de objetos no modelo de dados físico, conforme pesquisa feita utilizando as metodologias de desenvolvimento da empresa que o acadêmico atua, em questão a WEG, e outras que prestam serviços terceirizados.

Tabela 01 – Regras para criação da base dados

Objeto	**Exemplo**	**Comentário**
Aplicação	REP	*Aaa = iniciais do nome da aplicação*
Usuário proprietário do sistema	REP	Cada sistema terá um usuário proprietário e seu desenvolvimento no Designer e no Oracle será através deste.

Tabela	REPRESENTANTE_REP	A nomenclatura da tabela deverá estar no modo singular precedido por "_" e as iniciais da aplicação. A tabela deverá ter pelo menos um campo. A tabela deverá ter um sinônimo com o mesmo nome e somente um sinônimo. A tabela deverá obrigatoriamente ter uma chave primária (PK) relacionada.
Alias	REPRES	Abreviação da nomenclatura do objeto em até 6 dígitos, não podendo usar os símbolos "_" e "-" . O alías deve ser criado para todos os objetos de tabela e visão.
Chave primária (PK)	REPRES_PK	Deve-se usar o alias da tabela/visão precedido do termo PK (*Primary Key*). A Pk deve ter pelo menos um campo na sua definição.
Chave estrangeira (FK)	REPRES_FILIAL_FK	Deve-se usar o alias da tabela pai precedido do alias da tabela filho mais o termo FK (Foreing Key). A Fk deve ter pelo menos um campo na sua definição. O nome do campo da fk deve ser igual ao campo do objeto relacionado tendo também o mesmo tipo de dados. Toda Fk criada deve possuir um índice relacionado.
Chave estrangeira (FK)	REPRES_FILIAL_FK_01	Quando for necessário um segundo relacionamento entre as mesmas tabelas
Chave única (UK)	REPRES_UK_01	Deve-se usar o alias da tabela, precedido do termo UK (Unique Key) mais uma numeração seqüencial. A Uk deve ter pelo menos um campo na sua definição.
Índice	REPRES_SK_I	Deve se usar o alias da tabela, precedido de "_ SK_I" Todo o índice deverá ter pelo menos um campo.

	REPRES_FILIAL_FK_I	O Índice criado para a chave estrangeira FK deve ter o nome da FK, acrescido de "_I". Cada FK (*validate in server ou both*) deverá ter um índice correspondente, se já não houver algum com suas chaves, ou seja, somente criar o índice se ele não estiver embutido em outro índice ou PK. Justificativa: o índice deve estar na tabela onde está definida a FK, pois caso sejam feitas alterações na tabela pai, com o índice ele somente "loca" os registros filhos correspondentes. Sem o índice, toda a tabela filho ficará locada.
Sinônimos Sinônimos	REPRESENTANTE_REP REP_EMPRESA	Os sinônimos devem ter o mesmo nome do objeto de origem. Devem ter sinônimos: tabelas, visões, snapshots e procedures Os sinônimos se aplicam aos objetos Tabela, Visão, Domínio, e Seqüência todos obrigatórios.
Visões	REP_EMPRESA REP_DUPLICATA_PAGA	O acesso a outras aplicações deve ser feito por visões. A nomenclatura da visão deve ter a inicial da aplicação precedidas do nome do objeto ao qual está acessando As visões devem ter o campo do modelo de dados "*free format select = true*" e a lógica do select definida no "select text" e no "where condition"
Seqüências	SEQ_EMPRESA_REP	A nomenclatura das seqüências deve ter como inicial o termo SEQ precedido do nome do objeto ao qual a seqüência será utilizada. Uma seqüência deve ser utilizada apenas por um objeto.

Campos	CD_REPRES	Os campos devem ter um prefixo que os identifique precedido de uma descrição que identifique o seu conteúdo. Todos os campos com o mesmo nome devem ter também a mesma definição. Todo campo com o prefixo ID_, deve ter um domínio definido. Se o campo tiver um domínio definido o seu prefixo obrigatoriamente deve ser ID. Cada termo que compõe o nome do campo não poderá ter mais de 10 dígitos. Para todos os campos as propriedades "PROMPT, HINT, DESCRIPTION" Veja na tabela XX a lista de prefixos.
Roles (grupos)	R_REP_CONSULTA R_REP_GERAL	A segurança do sistema deverá ser feita por roles do Oracle. Todo sistema deverá ter no mínimo 2 roles: - R_aaa_CONSULTA para possibilitar consultas a partir de outras aplicações; R_aaa_GERAL acesso de consulta e atualização a todas as tabelas, e grant de execute para todas as procedures.
Domínios	ID_FORMA_PAGAMENTO	A nomenclatura dos domínios da aplicação deverá ter como inicial as letras ID precedido de um termo explicativo que esteja presente no glossário de termos, devendo ser igual ao campo que está relacionado.

4.1 NOMENCLATURA DE CAMPOS

Devem ser criados campos para cada finalidade distinta, ou seja, não poderá existir um campo para dois tipos de informação.

A ordem dos campos na tabela, visão, *snapshot* deve obedecer a uma ordem lógica, imaginando como deve ocorrer o preenchimento na tela. Os campos chave devem ser os primeiros na ordem em que estão na PK. Os campos deverão ter uma descrição completa que permita entender sua utilização, objetivo e essência de sua informação. Os campos que fazem parte da PK deverão ter um nome mais significativo, de forma que possa ser utilizado o mesmo nome de campo nas tabelas onde os mesmos estão relacionados e sejam facilmente identificados (ex: não usar CD_REGISTRO e NR_SEQUENCIA, mas sim CD_CLIENTE e NR_LINHA_NOTA_FISCAL).

A nomenclatura do campo deverá estar no singular obedecendo aos termos cadastrados no glossário de termos tendo um prefixo, conforme mostra a tabela 02.

Tabela 02 – Prefixo padrão para nomenclatura de campos

Prefixo	**Descrição**	**Tipo de dados**	**Exemplos**
CD	Código		CD_REPRESENTANTE
DS	Descrição	VARCHAR2	DS_ENDERECO
DT	Data	DATE	DT_CADASTRO
ID	Lista de valores (domínios)		ID_ESTADO_CIVIL
IM	Imagem/foto		IM_ITEM
FA	Fator (divide)		FA_CONVERSAO
NM	Nome		NM_FUNCIONAR
NR	Número		NR_FAX
PR	Percentual		PR_DESCONTO
QT	Quantidade	NUMBER	QT_PECAS
SG	Sigla	VARCHAR2	SG_EMPRESA
TA	Taxa (multiplica)		TA_JUROS_MÊS
TP	Tipo (sem domínio)		TP_CARCACA
TX	Texto	TEXT	TX_ITEM
VL	Valor	NUMBER	VL_FATURADO

4.1.1 PROMPT E HINT

Para todos os objetos deverá ser preenchido o campo *hint* em português e totalmente com letras maiúsculas, e o campo *prompt* em português, com letras minúsculas e inicias em maiúsculo.

O campo *prompt* de campos chave deverá ser descrito conforme a informação que contém, sem o termo código. Exemplo: CD_CLIENTE, prompt = Cliente, hint = CODIGO DO CLIENTE.

4.1.2 DOMÍNIOS

Deve-se utilizar domínio sempre que houver lista de valores estática. O nome do domínio deve ser igual ao nome do campo, exceto para os domínios de uso geral . Exemplo: ID_SIM_NÃO. Não usar domínio quando for uma lista de valores que pode sofrer alterações facilmente. Neste caso criar tabelas a serem mantidas pelo usuário.

4.1.3 DATAS

Para campos do tipo de dados data (*Date*) a definição do ano deve ser de 4 dígitos. Campos com informação apenas do ano deverão ser numéricos com 4 dígitos e devendo consistir o valor informado entre 1900 e 2100. Campos com informação apenas do mês deverão ser numéricos devendo consistir os valores informados.

5 DESENVOLVIMENTO DO PROTÓTIPO

Neste tópico serão abordados a especificação, implementação e funcionamento do protótipo. Também será apresentado um estudo de caso submetido ao protótipo.

5.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O protótipo deverá realizar uma análise completa no repositório de dados do *Case Designer 6.i* identificando possíveis divergências ou erros causados pelo desenvolvedor do modelo de dados. Todas as divergências encontradas serão armazenadas em uma tabela para que se possa gerar relatórios.

As consistências que o protótipo deverá fazer para validar o modelo de dados são:

a) verificar a existência de tabelas;
b) verificar o número máximo de caracteres para o nome da tabela;
c) verificar a existência de campos na tabela;
d) verificar a existência de chaves primarias (PK), verificando seu campos;
e) verificar e validar chaves únicas (UK);
f) verificar e validar chaves estrangeiras (FK);
g) verificar os índices. Os campos dos índices, caso existam, não podem ser iguais aos campos da PK do objeto em questão;
h) verificar a existência do sinônimo com a sua nomenclatura igual à nomenclatura do objeto em questão, e somente um sinônimo por objeto;
i) verificar a existência de campos com a mesma nomenclatura e com o tipo de dados diferente;
j) comparar todos os nomes de objetos (tabelas, visões, sinônimos, procedures etc.) com o dicionário de termos.

O protótipo deverá fazer a validação dos seguintes objetos do modelo dados conforme os padrões de desenvolvimento descritos no capitulo 4:

a) tabelas (colunas, PK, UK,FK, índices e sinônimo);
b) visões (colunas e texto);
c) domínios;
d) seqüências.

5.2 ESPECIFICAÇÃO

A especificação é apresentada através do diagrama de casos de uso, diagrama de classes, diagrama de seqüência e o projeto físico da base dados.

Para a especificação do protótipo utilizou-se, a linguagem de modelagem unificada (UML) que Quatrani (2001), define como linguagem de modelagem universal. O apelo do mercado em criar um padrão de modelagem para sistemas orientados a objeto permitiu que as características de várias técnicas fossem incorporadas nessa linguagem, cujos recursos de visualização e concepção de sistemas têm se tornado imperativo em qualquer ambiente de desenvolvimento. Para dar suporte a UML foi usada a ferramenta CASE *Rational Rose*

Para criação do banco de dados do protótipo foi utilizada a ferramenta *Case Designer 6.i.* Esta ferramenta dá suporte à criação de todos os objetos do banco de dados como tabelas, visões, seqüências, domínios e S*tored Procedures*.

5.2.1 CASO DE USO

Segundo Matos (2002), casos de uso são utilizados para identificar as regras do negócio, sendo uma excelente forma de entender o ponto de vista do usuário simplesmente pelo fato de que modela o que ele precisa executar.

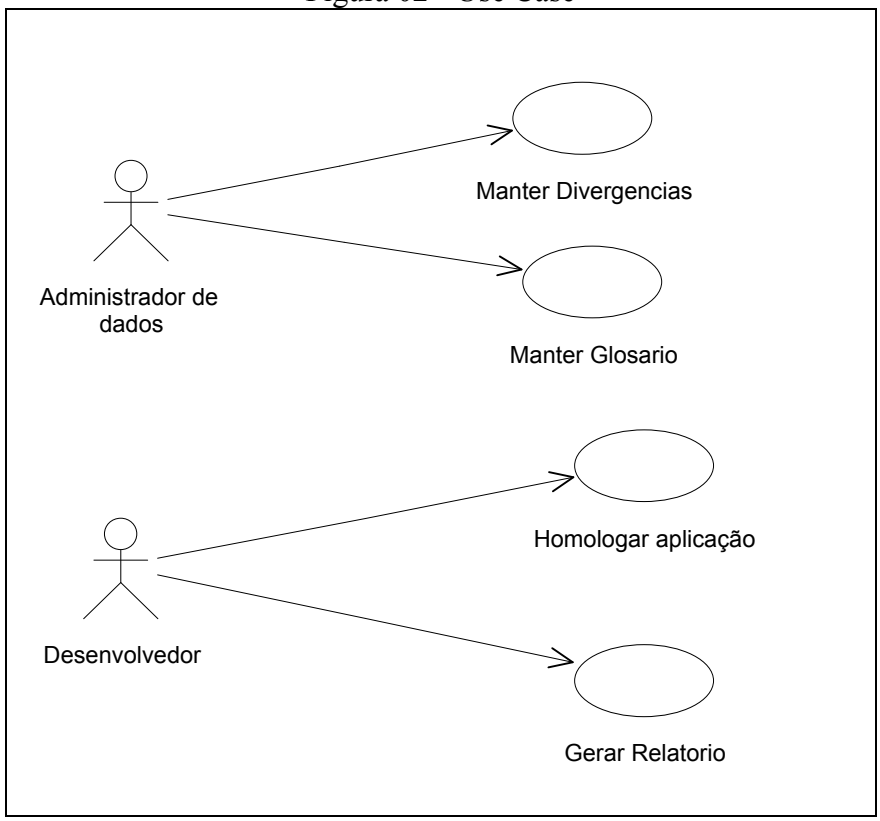
Neste protótipo foram observados quatro casos de uso principais descritos a seguir e podem ser visualizados na fig. 02:

a) manter divergências: nesta etapa o administrador de dados é responsável por identificar e definir quais vão ser as divergências que o sistema irá identificar, devendo ele cadastrar a descrição da divergência no banco de dados;
b) manter termos: nesta etapa o administrador de dados é responsável por cadastrar todos os termos possíveis para nomenclatura dos objetos que compõem um banco de dados. Os termos que o AD irá cadastrar devem estar de acordo com a metodologia definida no capitulo 4;
c) homologar aplicação: nesta etapa o desenvolvedor, após ter desenvolvido o modelo físico, no *Oracle Designer 6.i,* deve executar a homologação escolhendo qual aplicação quer homologar. A homologação consiste em varrer todo o metadados do SGBD Oracle procurando por erros de modelagem, definidos neste trabalho como

divergências. Essas divergências são gravadas em uma tabela que tem como chave o *owner*da aplicação;

d) gerar relatório: nesta etapa o próprio sistema gera um relatório baseado nas divergências encontradas pelo caso de uso homologar aplicação.

Figura 02 - Use Case



5.2.2 DIAGRAMA DE CLASSES

Conforme Matos (2002), uma classe é uma abstração de um conjunto de coisas que possuem características e operações em comum. Ou seja, uma classe é um conjunto de objetos.

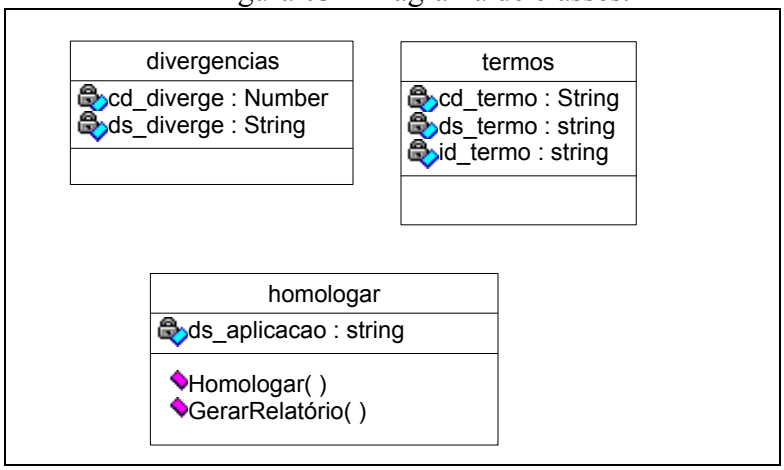
O diagrama de classes mostrado pela fig. 03 mostra as classes, seus atributos e métodos que executa:

a) divergências: esta classe tem os métodos ler, cadastrar, alterar e excluir as possíveis divergências que o Administrador de Dados irá manter;

b) termos: esta classe tem os métodos ler, cadastrar, alterar e excluir, termos e prefixos que são mantidos pelo administrador de dados para a nomenclatura dos objetos;

c) homologar: esta classe tem como propósito iniciar o processo de homologação fazendo uma chamada para uma S*tored Procedure* que é responsável pela leitura do metadados, identificando as divergências, caso existam, e após imprimir um relatório contendo toda a lista de divergências encontradas.

Figura 03 - Diagrama de classes.

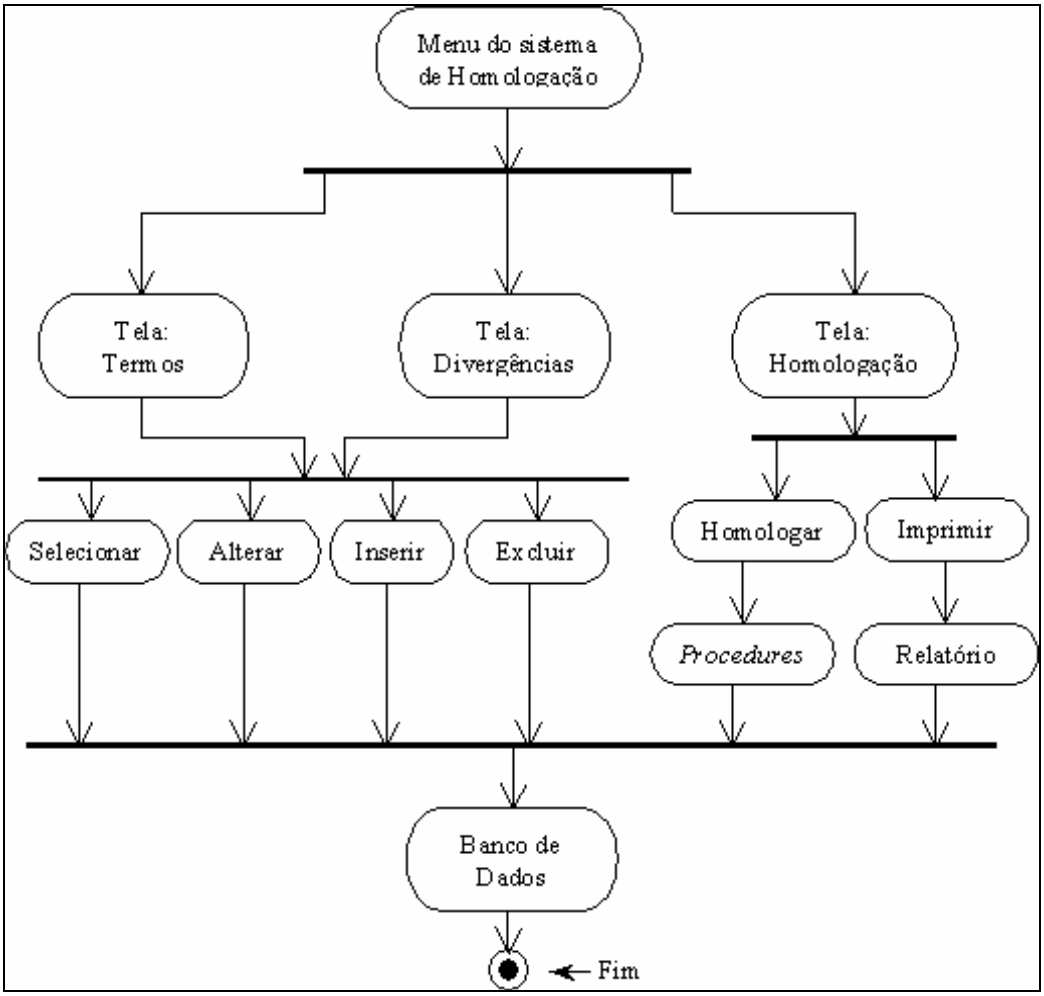


5.2.3 DIAGRAMAS DE ATIVIDADES

Segundo Barros (2000), diagramas de atividade capturam ações e seus resultados. Eles focam o trabalho executado na implementação de uma operação (método), e suas atividades numa instância de um objeto. O diagrama de atividade é uma variação do diagrama de estado e possui um propósito um pouco diferente do diagrama de estado, que é o de capturar ações (trabalho e atividades que serão executados) e seus resultados em termos das mudanças de estados dos objetos.

Na fig. 04 pode-se visualizar o diagrama de atividades do protótipo.

Figura 04 - Diagrama de atividades



O sistema possui uma tela principal com um menu, este menu deve possuir os items:

a) cadastrar termos: esta tela tem as funcionalidades de inserir, excluir, alterar e selecionar dados do banco de dados fazendo estas operaçãoes com a tabela *glossario_termo_hom*;

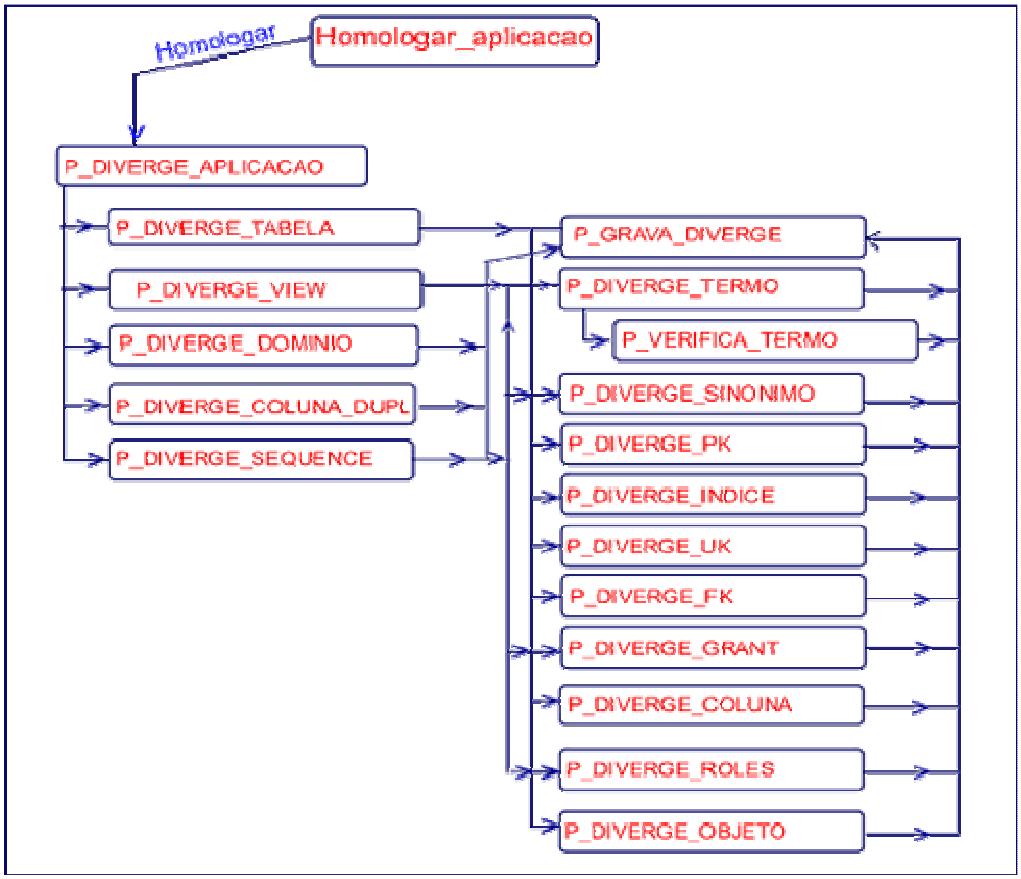
b) cadastrar divergencias : esta tela tem as funcionalidades de inserir, excluir, alterar e selecionar dados do banco de dados fazendo estas operaçãoes com a tabela *lista_diverge_hom*;

c) homologar: esta tela tem as funiconalidades de homologar as aplicações e imprimir homologações processadas. A homologação chama a *stored procedure* P_DIVERGE_APLICACAO que é encarregada de percorrer o repositório e comparar o modelo físico desenvolvido com o padrão pesquisido. Caso encontre

alguma divergência grava-a na tabela *diverge_aplicação_hom.* O processo de impressão lê a tabela *diverge_aplicação_hom,* filtra e relaciona as informações com a tabela lista_diverge_hom e grava em um arquivo texto denominado com o nome da aplicação e o numero do processo.

Para demonstrar o processo de homologação a fig. 07 demonstra todas as S*tored Procedures* que fazem parte do protótipo demonstrando a seqüência de chamadas para cada uma das *procedures*. Cada *Stored Procedure* é demonstrada por um retângulo, contendo seu nome, as setas estão indicando as chamadas de cada S*tored Procedure*, demonstrando assim o fluxo da sua execução.

Figura 05 - Interações das *Stored Procedures*



A seguir o detalhamento das *stored procedures* desenvolvidas para o protótipo:

- p_diverge_aplicacao: este procedimento e encarregado de controlar a homologação, fazendo a chamada para os procedimentos que validam os objetos do tipo tabela, visão, domínio, colunas duplicadas e seqüências;

- p_diverge_tabela: este procedimento é responsável pela verificação das divergências de objetos do tipo tabela;
- p_diverge_view: este procedimento é responsável pela verificação das divergências de objetos do tipo visão;
- p_diverge_domínio: este procedimento é responsável pela verificação das divergências de objetos do tipo domínio;
- p_diverge_coluna_dupl: este procedimento é responsável pela verificação das divergências quando existirem colunas duplicadas com tipo de dados diferentes;
- p_diverge_sequence: este procedimento é responsável por verificar a nomenclatura da *sequence*, comparando a sua nomenclatura com a nomenclatura do objeto onde está associada;
- p_grava_diverge: este procedimento e responsável por gravar as divergências de todos os objetos na base de dados;
- p_diverge_termo: este procedimento e responsável em dividir o nome dos objetos e chamar o procedimento p_verifica_termo. Esse procedimento é responsável em compara o termo com o glossário de termos;
- p_diverge_sinonimo: este procedimento verifica a existência de sinônimos para os objetos do tipo tabela, visão e *stored procedures* verificando também se a nomenclatura desses objetos é igual ao objeto relacionado;
- p_diverge_pk: este procedimento é responsável em verificar as regras de criação de chaves primarias;
- p_diverge_fk: este procedimento é responsável em verificar as regras de criação de chaves estrangeiras;
- p_diverge_uk: este procedimento é responsável em verificar as regras de criação de chaves estrangeiras;
- p_diverge_índice: este procedimento é responsável em verificar as regras de criação de índices;

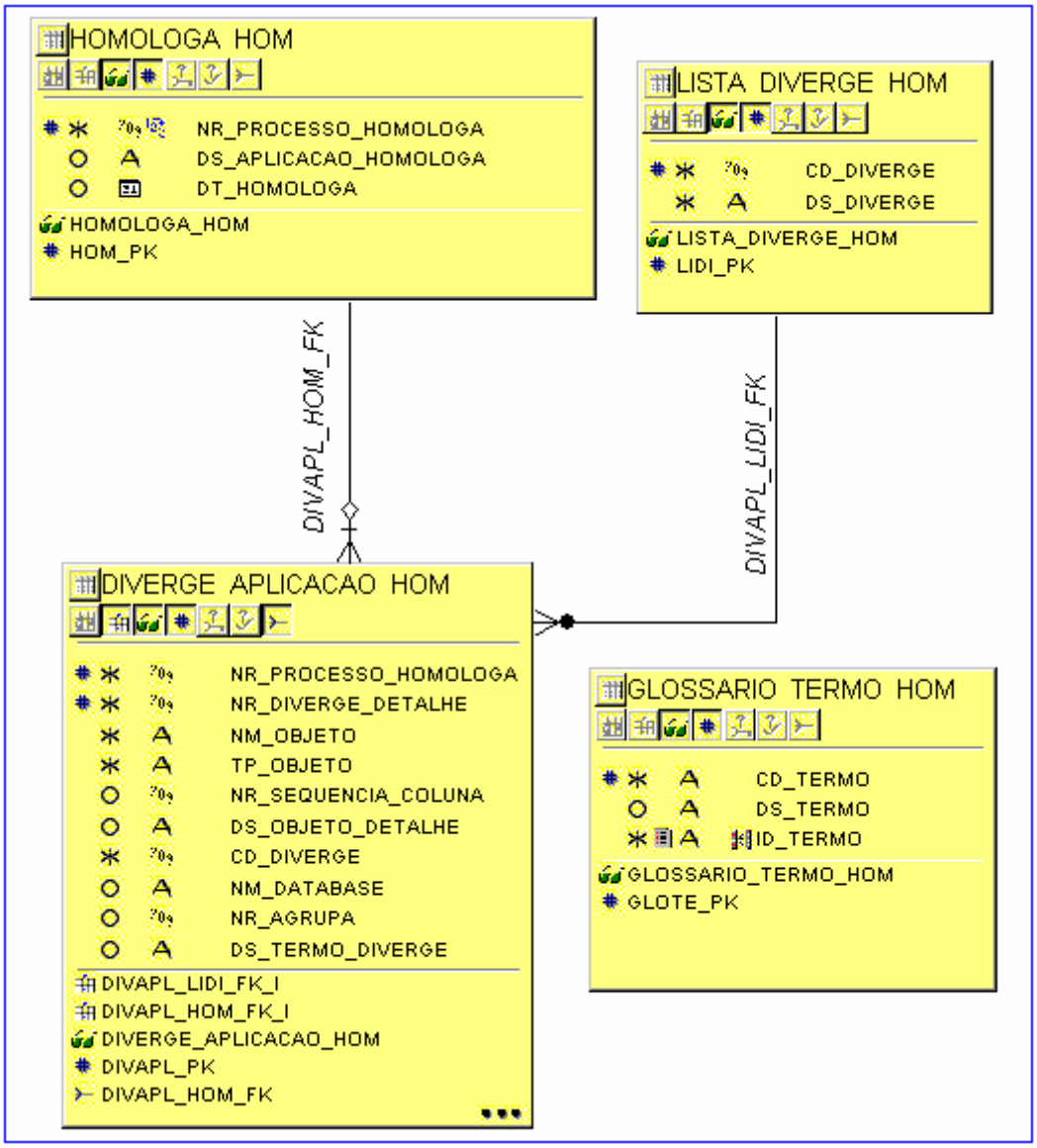
- p_diverge_grant: este procedimento é responsável em verificar objetos que não tenham direito (*grant*) em nenhuma *role*;
- p_diverge_coluna: este procedimento é responsável em validar as colunas definidas para as tabelas e visões;
- p_diverge_roles: este procedimento é responsável em verificar a existências das *roles* padrão (R_<aplicacao>_GERAL e R_<aplicacao>_CONSULTA) e outras que porventura sejam criadas.

5.2.4 PROJETO FISICO

A fig. 06 mostra a estrutura do projeto físico, ou seja, quais as tabelas e suas definições que farão parte do sistema. Esta figura demonstra as quatro tabelas que farão parte do modelo de dados :

a) homologa_hom: foi criada para armazenar todas as homologações que forem sendo executadas. São armazenados: um código seqüencial, o nome da aplicação a ser homologada e a data da homologação;
b) diverge_aplicacao_hom: foi criada para armazenar todas as divergências encontradas pelo protótipo. É esta a tabela que será a base do relatório;
c) lista_diverge_hom: foi criada para armazenar as lista de divergências que serão procuradas. Está tabela pode ser definida como a tradução das regras e padrões de nomenclaturas. É nela que se encontra toda a descrição das divergências;
d) glossario_termo_hom: foi criada para armazenar todos os termos possíveis para o desenvolvedor criar o seu banco de dados.

Figura 06 - Projeto Físico



O dicionário de dados encontra-se no anexo 1.

5.3 TÉCNICAS E FERRAMENTAS UTILIZADAS

Para a implementação do protótipo foi usada a ferramenta de desenvolvimento *Visual Studio.Net* tendo como linguagem de programação o *Visual Basic.net.* Parte da aplicação foi desenvolvida ulilizando-se S*tored Procedures* que executam no próprio banco de dados Oracle e usa a linguagem *PL/SQL.* E tiveram grande utilidade o SQL Plus da ORACLE e a ferramenta TOAD da quest, estas ferramentas foram utilidas para identificação da tabelas do repositório e para teste de *scripts sql* desenvolvidos para o protótipo.

5.3.1 VISUAL BASIC .NET

Conforme Haddad (2001), o *Visual Basic.net*, permite ao desenvolvedor um desenvolvimento rápido e visual de aplicativos tanto para Windows quanto para Web de tal maneira que eles posteriormente possam rodar também em outras plataformas. Para conseguir isso a Microsoft investiu em uma total reestruturação de seu pacote de desenvolvimento. Praticamente tudo mudou, o ambiente de desenvolvimento, as linguagens de programação e a plataforma sobre o qual rodam os aplicativos. Agora existe um ambiente integrado de desenvolvimento para todas as linguagens que compõem o Visual Studio.net: o Visual Basic, o Visual C++ e o Visual C#. Além disso a integração entre estas ferramentas é bastante grande, a ponto de elas poderem compartilhar funções e dados entre si. Isso tornou se possível porque os aplicativos criados com qualquer linguagem do Visual Studio.Net compartilham as mesmas bibliotecas de recursos.

Haddad (2002), afirma que após a reestruturação total do pacote de desenvolvimento da Microsoft o Visual Basic.Net passou a ser verdadeiramente orientado a objetos. O Visual Basic suporta agora herança, interfaces, sobre-carregamentos, construtores, destruidores e compartilhamento de membros.

5.4 IMPLEMENTAÇÃO

A implementação do protótipo foi iniciada atráves da identificação das tabelas do metadados do Oracle e o desesenvolvimento das S*tored Procedures* que realizam a varredura do modelo físico procurando por divergências.

O procedimento principal chamado pelo protótipo é a *Stored Procedure* P_DIVERGE_APLICACAO, que recebe como parâmetro o nome da aplicação. Esse procedimento controla a chamada dos procedimentos para verificação das tabelas, visões, domínios, colunas duplicadas e seqüências.

O primeiro passo é criar um código para o processo de homologação e gravar um registro na tabela *Homologa_hom.* Depois deve-se identificar qual o código da aplicação acessando a tabela *SDD_FOLDERS* (nesta tabela encontram-se todas as aplicações criadas pelos desenvolvedores) do repositório do C*ase Designer*, então chamar as S*tored Procedures* que irão procurar as divergências em tabelas, visões, domínios, seqüências e identificando

também objetos com a nomenclatura idêntica mas com a definição diferente. O quadro 02 mostra o código do procedimento.

Quadro 02 – Procedimento P_DIVERGE_APLICACAO

```
1 CREATE OR REPLACE PROCEDURE P_DIVERGE_APLICACAO
2  (PRM_NM_APLICACAO IN VARCHAR2)IS
3 -- Program Data
4 V_NR_PROCESSO_HOMOLOGA NUMBER;
5 -- PL/SQL Block
6 begin
7 declare
8 --Cursor que busca o codigo da aplicação
9   cursor C1 is select distinct irid
10                  from sdd_folders
11                 where name  = prm_nm_aplicacao;
12
13   begin
14      select SEQ_HOMOLOGA_HOM.nextval
15      into v_nr_processo_homologa
16      from dual;
17
18     INSERT INTO HOMOLOGA_HOM VALUES (V_NR_PROCESSO_HOMOLOGA,PRM_NM_APLICACAO,SYSDATE);

19   for r1 in c1 loop
20       p_diverge_tabela     (r1.irid,v_nr_processo_homologa);
21       p_diverge_view       (r1.irid,v_nr_processo_homologa);
22       p_diverge_dominio    (r1.irid,v_nr_processo_homologa);
23       p_diverge_coluna_dupl(r1.irid,v_nr_processo_homologa);
24       p_diverge_sequence   (r1.irid,v_nr_processo_homologa);
25     end loop;
26   end;

27 end;
```

Conforme mostra o quadro 02 neste código, conforme é demonstrado na linha 20 é chamado o procedimento P_DIVERGE_TABELA. Esse procedimento é encarregado de validar objetos do tipo tabela,

O primeiro passo do procedimento P_DIVERGE_TABELA é executar uma *query sql* para selecionar todas as tabelas que se encontram no modelo de dados em questão. Para fazer isso foi criado um cursor denominado *c_tab.* Essa consulta acessa as tabelas *ci_table_definitions e sdd_folder_members.* Esse cursor é aberto em um *loop* e então para cada tabela encontrada são verificadas as regras de nomenclatura, da tabela, dos campos e de suas *constraint*, é verificado também as regras de composição, da chave primária (PK), chave estrangeira (FK), chave única (UK), índices (SK) e sinônimos dos objetos. O código desse procedimento é visto pelo quadro 03.

Quadro 03 – Procedimento P_DIVERGE_TABELA

```
CREATE OR REPLACE PROCEDURE P_DIVERGE_TABELA
 (PRM_NUM_ID_APLICACAO IN NUMBER
 ,PRM_NR_PROCESSO_HOMOLOGA IN NUMBER )IS
-- Program Data
V_NOME_APLICACAO VARCHAR2(240);
-- PL/SQL Block
BEGIN
declare
    cursor c_tab is
                        select distinct
                                tbl.irid                        tab_id
                               ,tbl.name
                               ,tbl.alias
                               ,tbl.initial_number_of_rows      start_rows
                               ,tbl.maximum_number_of_rows      end_rows
                        from ci_table_definitions tbl,
                                sdd_folder_members   fom
                        where  fom.folder_reference = prm_num_id_aplicacao
                        and    fom.ownership_flag   = 'Y'
                        and    tbl.irid             = fom.member_object ;

    aux_tab_id_impl  ci_table_implementations.irid%type;
    aux_database_id ci_table_implementations.database_user_reference%type;
    aux_tablespace_id ci_table_implementations.tablespace_reference%type;
    aux_grava_Reg_controle  varchar2(1);
  begin
    for r_tab in c_tab loop
    -- Busca Database do objeto e
    -- verifica se existem dois databases definidos para o mesmo objeto
       begin
        select tbi.irid                         tab_id_impl
              ,tbi.database_user_reference      database_id
              ,tbi.tablespace_reference         tablespace_id
          into aux_tab_id_impl,
               aux_database_id,
               aux_tablespace_id
          from ci_table_implementations         tbi
         where tbi.table_definition_reference(+) = r_Tab.tab_id;
         aux_grava_reg_controle := 'N';
       exception
            when no_Data_Found then
                 aux_tab_id_impl   := null;
                 aux_database_id   := null;
                 aux_tablespace_id := null;
            when too_many_rows then
                 -- Busca um dos database/user associados
                 select  tbi.irid                       tab_id_impl
                        ,tbi.database_user_reference    database_id
                        ,tbi.tablespace_reference       tablespace_id
                    into aux_tab_id_impl,
                         aux_database_id,
                         aux_tablespace_id
                   from ci_table_implementations       tbi
                   where tbi.table_definition_reference(+) = r_Tab.tab_id
                   and   rownum = 1;
                -- Inserir registro de controle --
                p_grava_diverge    (prm_nr_processo_homologa,r_tab.name,'TABLE'
                                    ,null,null,null,null,0,null);
                 aux_grava_reg_controle := 'S';
                 p_grava_diverge (prm_nr_processo_homologa,r_tab.name,'TABLE'
                                  ,null,null,73,null,0,null);
        end;
    -- Inserir registro de controle --
      if aux_grava_reg_controle <> 'S'  then
            p_grava_diverge    (prm_nr_processo_homologa,r_tab.name,'
                                TABLE',null,null,null,aux_database_id
                                ,0 ,null);
     end if;

      if (aux_database_id is null) then
         p_grava_diverge (prm_nr_processo_homologa,r_tab.name,'TABLE'
                          ,null,null,54,aux_database_id ,0 ,null);
```

```
      end if;

      if((r_tab.start_rows is null) or (r_tab.end_rows   is null)) then
         p_grava_diverge (prm_nr_processo_homologa,r_tab.name,'TABLE'
                          ,null,null,27,aux_database_id
                          ,0
                          ,null);
      end if;

    -- Verificação de Termo --
      p_diverge_termo(r_tab.name,1,prm_nr_processo_homologa,r_tab.name,'TABLE'
                       ,null,null,64,aux_database_id,0);

    -- Verificação de Sinonimos --
      p_diverge_sinonimo ('TABLE' ,r_tab.name,null,r_tab.tab_id,prm_nr_processo_homologa
                          ,aux_database_id );

    -- Verificação de Primary Key --
      p_diverge_pk       ('TABLE',r_tab.name,r_tab.alias ,r_tab.tab_id
                          ,prm_nr_processo_homologa,aux_database_id );

    -- Verificação de Indices --
      p_diverge_indice   ('TABLE',r_tab.name ,r_tab.alias,r_tab.tab_id
                          ,prm_nr_processo_homologa,aux_database_id );

    -- Verificação de Unique Key --
      p_diverge_uk       ('TABLE',r_tab.name,r_tab.alias,r_tab.tab_id
                          ,prm_nr_processo_homologa
                          ,aux_database_id );

    -- Verificação de Foreign Key --
      p_diverge_fk       ('TABLE',r_tab.name,r_tab.tab_id,prm_nr_processo_homologa
                          ,aux_database_id );

    -- Verificação de Grants --
      p_diverge_grant    ('TABLE',r_tab.name,r_tab.tab_id,aux_tab_id_impl
                          ,prm_nr_processo_homologa ,aux_database_id );

    -- Verificação de Colunas --
      p_diverge_coluna   ('TABLE',r_tab.name,r_tab.tab_id,prm_nr_processo_homologa
                          ,aux_database_id );
    select distinct name
          into   V_NOME_APLICACAO
          from  sdd_folders
          where irid =  prm_num_id_aplicacao;
      if substr(r_tab.name, length(r_tab.name)-   (
length(V_NOME_APLICACAO)),(length(V_NOME_APLICACAO) +1)) <> '_'|| V_NOME_APLICACAO then
                p_grava_diverge (prm_nr_processo_homologa,r_tab.name,'TABLE',null
                           ,null,22 ,aux_database_id,0,null );
      end if;
      -- Verificação de Roles --
        p_diverge_roles ('TABLE',r_tab.name,prm_num_id_aplicacao,r_tab.tab_id
                         ,aux_tab_id_impl,prm_nr_processo_homologa ,aux_database_id );
    -- Verificação de Objeto --
       p_diverge_objeto('TABLE' ,r_tab.name ,r_tab.alias,r_tab.tab_id,aux_tablespace_id
                         ,prm_nr_processo_homologa,aux_database_id );
  end loop;      -- c_tab
end;
END;
```

Conforme demonstrado no quadro 02, linha 21, é chamado o procedimento P_DIVERGE_VIEW. Esse procedimento é encarregado de validar objetos do tipo visão.

O primeiro passo do procedimento P_DIVERGE_VIEW é verificar a existência de visões no modelo de dados que está sendo homologado. Caso existam, selecionam-se todas as visões executando o cursor *cur_vie.* Essa consulta acessa as tabelas *ci_view_definitions e*

sdd_folder_members. Esse cursor é aberto em um *loop* e então para cada visão encontrada, são verificadas as regras de nomenclatura: da visão , dos campos e de suas *constraints.* São verificados também as regras de composição, da chave primária (PK), chave estrangeira (FK), chave única (UK), índices (SK), sinônimos e o texto da visão. O código desse procedimento é visto no quadro 04

Quadro 04 –Procedimento P_DIVERGE_VIEW

```
CREATE OR REPLACE PROCEDURE P_DIVERGE_VIEW
 (PRM_NUM_ID_APLICACAO IN NUMBER
 ,PRM_NR_PROCESSO_HOMOLOGA IN NUMBER ) IS
begin
  declare
    cursor cur_vie is
    select vie.irid                      view_id
          ,vie.name
          ,vie.alias
          ,vie.override_select_text_flag select_text_flag
    from   ci_view_definitions           vie
          ,sdd_folder_members            fom
    where  fom.folder_reference          = prm_num_id_aplicacao
    and    fom.ownership_flag            = 'Y'
    and    vie.irid                      = fom.member_object;
    aux_view_id_impl    ci_view_implementations.irid%type;
    aux_database_id     ci_view_implementations.database_user_reference%type;
    aux_grava_Reg_controle varchar2(1);

 begin

    for r_vie in cur_vie loop
    -- Busca Database do objeto e
    -- verifica se existem dois databases definidos para o mesmo objeto
       begin
          select vii.irid                       view_id_impl
              ,vii.database_user_reference    database_id
          into aux_view_id_impl,
               aux_database_id
          from ci_view_implementations       vii
          where vii.view_definition_reference(+) = r_vie.view_id;

          aux_grava_reg_controle := 'N';

       exception
            when no_Data_Found then
                 aux_view_id_impl  := null;
                 aux_database_id   := null;
            when too_many_rows then
                select vii.irid                       view_id_impl
                      ,vii.database_user_reference    database_id
                into aux_view_id_impl,
                       aux_database_id
                from ci_view_implementations vii
                where vii.view_definition_reference(+) = r_vie.view_id and
                      rownum = 1;

                -- Inserir registro de controle --
                p_grava_diverge    (prm_nr_processo_homologa,r_vie.name
                                   ,'VIEW',null,null,null,null,0,null);

                 aux_grava_reg_controle := 'S';

                 p_grava_diverge (prm_nr_processo_homologa,r_vie.name
                                  ,'VIEW',null,null,73,null,0,null);

        end;

    -- Inserir registro de controle --
```

```
      if aux_grava_reg_controle <> 'S' then
        -- Inserir registro de controle --
        p_grava_diverge(prm_nr_processo_homologa,r_vie.name,'VIEW'
                        ,null ,null ,null,aux_database_id,0,null);
      end if;

    -- Verificação de Termo --
      p_diverge_termo(r_vie.name,1,prm_nr_processo_homologa,r_vie.name
                      ,'VIEW',null ,null,64,aux_database_id,0);

    -- Verificação de Sinonimos --
          p_diverge_sinonimo('VIEW',r_vie.name ,null,r_vie.view_id
                        ,prm_nr_processo_homologa,aux_database_id );

    -- Verificação de Primary Key --
      p_diverge_pk('VIEW',r_vie.name,r_vie.alias,r_vie.view_id
                    ,prm_nr_processo_homologa,aux_database_id );

    -- Verificação de Indices --
      p_diverge_indice('VIEW',r_vie.name,r_vie.alias,r_vie.view_id
                        ,prm_nr_processo_homologa,aux_database_id );

    -- Verificação de Unique Key --
      p_diverge_uk('VIEW',r_vie.name ,r_vie.alias,r_vie.view_id
                    ,prm_nr_processo_homologa,aux_database_id );

    -- Verificação de Foreign Key --
      p_diverge_fk('VIEW',r_vie.name,r_vie.view_id,
                  prm_nr_processo_homologa,aux_database_id );

    -- Verificação de Grants --
      p_diverge_grant('VIEW',r_vie.name ,r_vie.view_id,aux_view_id_impl
                      ,prm_nr_processo_homologa,aux_database_id );

    -- Verificação de Colunas --
      p_diverge_coluna('VIEW',r_vie.name,r_vie.view_id,prm_nr_processo_homologa
                        ,aux_database_id );

    -- Verificação de Roles --
      p_diverge_roles('VIEW',r_vie.name,prm_num_id_aplicacao,r_vie.view_id
                      ,aux_view_id_impl,prm_nr_processo_homologa
                      ,aux_database_id );

    -- Verificação de Objeto --
      p_diverge_objeto('VIEW',r_vie.name,r_vie.alias,r_vie.view_id
                        ,null,prm_nr_processo_homologa ,aux_database_id);
    end loop;      -- cur_vie
  exception
    when  no_data_found then
       null;

  end;
end;
```

Conforme demostrado no quadro 02, linha 22, ocorre o chamado ao procedimento P_DIVERGE_DOMINIO. Esse procedimento é encarregado de validar objetos do tipo domínio.

O primeiro passo do procedimento P_DIVERGE_DOMINIO é executar uma *query sql* para selecionar todos os domínios que se encontram no modelo de dados em questão. Para fazer isso foi criado um cursor denominado *cur_dom.* Essa consulta acessa as tabelas *ci_domains e sdd_folder_members.* Esse cursor e aberto em um loop, e então para cada objeto

do tipo domínio, encontrado, verifica-se as regras de nomenclatura do objeto e a existência de valores para o domínio. O código desse procedimento é visto no quadro 05.

Quadro 05 – Procedimento P_DIVERGE_DOMINIO

```
CREATE OR REPLACE PROCEDURE P_DIVERGE_DOMINIO
 (PRM_NUM_ID_APLICACAO IN NUMBER
 ,PRM_NR_PROCESSO_HOMOLOGA IN NUMERIC )IS
-- Program Data
V_QTD_REG NUMBER(10);
-- PL/SQL Block
begin
  declare
    cursor cur_dom is
    select dom.irid             domain_id
          ,dom.name
    from   ci_domains           dom
          ,sdd_folder_members   fom
    where  fom.folder_reference = prm_num_id_aplicacao
    and    fom.ownership_flag   = 'Y'
    and    dom.irid             = fom.member_object;
--
-- Inicio Processo --
  begin
    for r_dom in cur_dom loop
    -- Inserir registro de controle --
      p_grava_diverge(prm_nr_processo_homologa ,r_dom.name,'DOMAIN' ,null
                      ,null,null,null,0,null);
      if substr(r_dom.name, 1, 3) <> 'ID_'then
         p_grava_diverge(prm_nr_processo_homologa,r_dom.name,'DOMAIN',null,null ,51
                       ,0,0,null );
      end if;

      V_QTD_REG := 0;

      select nvl(count(*), 0)
      into   V_QTD_REG
      from   ci_columns
      where  domain_reference = r_dom.domain_id;



      if V_QTD_REG = 0 then
         p_grava_diverge(prm_nr_processo_homologa,r_dom.name,'DOMAIN',null
                     ,null ,52,0,0,null );
      end if;

      V_QTD_REG := 0;

      select nvl(count(*),0)
      into   V_QTD_REG
      from   ci_attribute_values
      where  domain_reference = r_dom.domain_id;

      if V_QTD_REG = 0 then
         p_grava_diverge(prm_nr_processo_homologa ,r_dom.name,'DOMAIN'
                        ,null ,null,62,0,0 ,null );
      end if;

      V_QTD_REG := 0;

      select nvl(count(*), 0)
      into   V_QTD_REG
      from   cdi_text
      where  txt_type = 'CDIDSC'
      and    txt_ref  = r_dom.domain_id
      and    txt_text is not null;

      if V_QTD_REG = 0 then
         p_grava_diverge(prm_nr_processo_homologa  ,r_dom.name ,'DOMAIN'
```

```
                            ,null  ,null,57,0,0,null );
      end if;

    end loop;      -- cur_dom
  end;

end;
```

Conforme demostrado no quadro 02, linha 23, pode-se visualizar a chamada para o procedimento P_DIVERGE_COLUNA_DUPL. Esse procedimento verifica a existência de um objeto com a mesma nomenclatura e então verifica a sua definição que não pode ser diferente. O código desse procedimento e visto no quadro 06.

Quadro 06 – Procedimento P_DIVERGE_COLUNA_DUPL

```
 CREATE OR REPLACE PROCEDURE P_DIVERGE_COLUNA_DUPL
  (PRM_NUM_ID_APLICACAO IN NUMBER
  ,PRM_NR_PROCESSO_HOMOLOGA IN NUMBER)IS
 -- Program Data
 V_DECIMAL_PLACES NUMBER(4);
 V_MAXIMUM_LENGTH NUMBER(5);
 V_DATATYPE VARCHAR2(16);
 -- PL/SQL Block
 begin
   declare
     cursor cur_dupl is
       select c.name
             ,count(c.name) qtde
       from   ci_columns c
             ,sdd_folder_members fom
      where  c.table_reference    = fom.member_object
      and    fom.folder_reference = prm_num_id_aplicacao
      and    fom.ownership_flag   = 'Y'
      having count(c.name) > 1
      group  by c.name;
-- Inicio Processo --
  begin
    for r_dupl in cur_dupl loop
      begin
        select distinct c.datatype
        into   V_DATATYPE
        from   ci_columns c
              ,sdd_folder_members             fom
        where  c.name                 = r_dupl.name
        and    c.table_reference      = fom.member_object
        and    fom.ownership_flag     = 'Y'
        and    fom.folder_reference   = prm_num_id_aplicacao;
      exception
      when too_many_rows then
        p_grava_diverge(prm_nr_processo_homologa,r_dupl.name,'COLUMN',null,null
                        ,31,0,0,null );
      end;
      begin
        select distinct maximum_length
        into   V_MAXIMUM_LENGTH
        from   ci_columns c
              ,sdd_folder_members             fom
        where  c.name                 = r_dupl.name
        and    c.table_reference      = fom.member_object
        and    fom.ownership_flag     = 'Y'
        and    fom.folder_reference   = prm_num_id_aplicacao;
      exception
      when too_many_rows then
        p_grava_diverge(prm_nr_processo_homologa ,r_dupl.name,'COLUMN',null
                        ,null,32,0,0,null );
      end;

      begin
```

```
        select distinct decimal_places
        into   V_DECIMAL_PLACES
        from   ci_columns c
              ,sdd_folder_members             fom
        where  c.name                = r_dupl.name
        and    c.table_reference     = fom.member_object
        and    fom.ownership_flag    = 'Y'
        and    fom.folder_reference  = prm_num_id_aplicacao;
      exception
      when too_many_rows then
        p_grava_diverge(prm_nr_processo_homologa,r_dupl.name,'COLUMN',null,null,33
                        ,0,0,null );
      end;
    end loop;      -- cur_dupl

  end;

end;
```

Conforme demostrado no quadro 02, linha 24, pode-se visualizar a chamada para o procedimento P_DIVERGE_SEQUENCE. Esse procedimento verifica a existência de um objeto do tipo seqüência. Caso encontre verifica a sua nomenclatura e se é utilizado por algum objeto do tipo tabela. O código desse procedimento é visto no quadro 07.

Quadro 07 – Procedimento P_DIVERGE_SEQUENCIA

```
CREATE OR REPLACE PROCEDURE P_DIVERGE_SEQUENCE
 (PRM_NUM_ID_APLICACAO IN NUMBER
 ,PRM_NR_PROCESSO_HOMOLOGA IN NUMBER ) IS
-- Program Data
V_NAME_SEQUENCE VARCHAR2(30);
V_QTD_REG NUMBER(10);
V_NAME VARCHAR2(30);
V_NOME_OBJETO VARCHAR2(256);
-- PL/SQL Block
begin
  declare
    cursor cur_seq is
      select seq.irid                    seq_id
            ,seq.name
      from   ci_sequences                seq
            ,sdd_folder_members          fom
      where  fom.folder_reference        = prm_num_id_aplicacao
      and    fom.ownership_flag          = 'Y'
      and    seq.irid                    = fom.member_object;

    aux_seq_id_impl        ci_sequence_implementations.irid%type;
    aux_database_id        ci_sequence_implementations.database_user_reference%type;
    aux_grava_Reg_controle varchar2(1);

begin
    for r_seq in cur_seq loop
    -- Busca Database do objeto e
    -- verifica se existem dois databases definidos para o mesmo objeto
       begin
                  select sei.irid                        seq_id_impl
                                ,sei.database_user_reference    database_id
                      into aux_seq_id_impl,
                               aux_database_id
                      from ci_sequence_implementations sei
                  where sei.sequence_reference(+) = r_seq.seq_id;
                               aux_grava_reg_controle := 'N';
       exception
            when no_Data_Found then
                 aux_seq_id_impl   := null;
                 aux_database_id   := null;
            when too_many_rows then
                -- Inserir registro de controle --
```

```
                select sei.irid                         seq_id_impl
                      ,sei.database_user_reference      database_id
                  into aux_seq_id_impl,
                       aux_database_id
                  from ci_sequence_implementations       sei
                 where sei.sequence_reference(+) = r_seq.seq_id
                   and rownum = 1;
                p_grava_diverge    (prm_nr_processo_homologa,r_seq.name,'SEQUENCE'
                                    ,null,null,null,null,0,null);
                aux_grava_reg_controle := 'S';
                p_grava_diverge (prm_nr_processo_homologa,r_seq.name,'SEQUENCE' ,null
                                    ,null,73 ,null,0,null);
     end;

  -- Inserir registro de controle --
    if aux_grava_reg_controle <> 'S'  then
      p_grava_diverge(prm_nr_processo_homologa  ,r_seq.name,'SEQUENCE'  ,null  ,null
                      ,null,aux_database_id,0,null);
     end if;

  -- Verificação de Termo --
    p_diverge_termo(r_seq.name ,1,prm_nr_processo_homologa,r_seq.name ,'SEQUENCE'
                      ,null,null ,64 ,aux_database_id,0);
  -- Verificação de Sinonimos --
    p_diverge_sinonimo('SEQUENCE',r_seq.name,null,r_seq.seq_id   ,prm_nr_processo_homologa
                        ,aux_database_id );
  -- Verificação de Grants --
    p_diverge_grant('SEQUENCE',r_seq.name  ,r_seq.seq_id,aux_seq_id_impl
                       ,prm_nr_processo_homologa ,aux_database_id );
  -- Verificação de Utilização das Colunas da Sequence --
    begin
      select cic.name
      into   V_NAME_SEQUENCE
      from   ci_columns cic
      where  cic.sequence_reference = r_seq.seq_id;
   exception
       when too_many_rows then
         p_grava_diverge(prm_nr_processo_homologa ,r_seq.name,'SEQUENCE'
                         ,null,null ,aux_database_id,0 ,null);

       when no_data_found then
          p_grava_diverge(prm_nr_processo_homologa ,r_seq.name,'SEQUENCE'  ,null
                         ,null,24,aux_database_id,0,null);
    end;

  -- Verificação de Utilização da Sequence --
    if substr(r_seq.name, 1, 4) <> 'SEQ_' then
       p_grava_diverge(prm_nr_processo_homologa,r_seq.name,'SEQUENCE' ,null
                        ,null ,25,aux_database_id,0,null);
    else
       V_NOME_OBJETO := substr(r_seq.name, 5,length(r_seq.name) - 4);
       V_NAME := ' ';

       begin

         select distinct td.name
         into   V_NAME
         from   ci_table_definitions td
               ,ci_columns           c
         where  td.name              = V_NOME_OBJETO
         and    td.id                = c.table_reference
         and    c.sequence_reference = r_seq.seq_id;
       exception
         when no_data_found then
            p_grava_diverge(prm_nr_processo_homologa,r_seq.name,'SEQUENCE' ,null
                            ,null,25,aux_database_id,0,null);
       end;
    end if;

  -- Verificação de Descrição da Sequence --
     V_QTD_REG := 0;

    select nvl(count(*), 0)
    into   V_QTD_REG
```

```
      from   cdi_text
      where  txt_type  = 'CDIDSC'
      and    txt_ref   = r_seq.seq_id
      and    txt_text is not null;

      if V_QTD_REG = 0 then
         p_grava_diverge(prm_nr_processo_homologa,r_seq.name,'SEQUENCE',null
                         ,null,57,aux_database_id,0,null );
      end if;
     -- Verificação de Roles --
      p_diverge_roles('SEQUENCE'r_seq.name,prm_num_id_aplicacao ,r_seq.seq_id,aux_seq_id_impl
                      ,prm_nr_processo_homologa,aux_database_id );
    end loop;      -- cur_seq
  end;
end;
```

Para implementação do protótipo, além das S*tored Procedures* foram implementadas três classes:

a) divergencias: essa classe controla a inserção, alteração, delecão e a exclusão de dados da tabela *lista_diverge_hom.*

b) termos: essa classe controla a inserção, alteração, delecão e a exclusão de dados da tabela *glossário_termo_hom.*

c) Homologar: essa classe controla a homologação das aplicações desenvolvidas no *case desigenr 6.i* chamando a *Stored Procedure* P_DIVERGE_APLICACAO e imprime o relatório da homologação contendo as divergências encontradas.

O código das classes desenvolvidas econtra-se no anexo 2.

5.4.1 OPERACIONALIDADE DA IMPLEMENTAÇÃO

O protótipo é composto por 3 camadas:

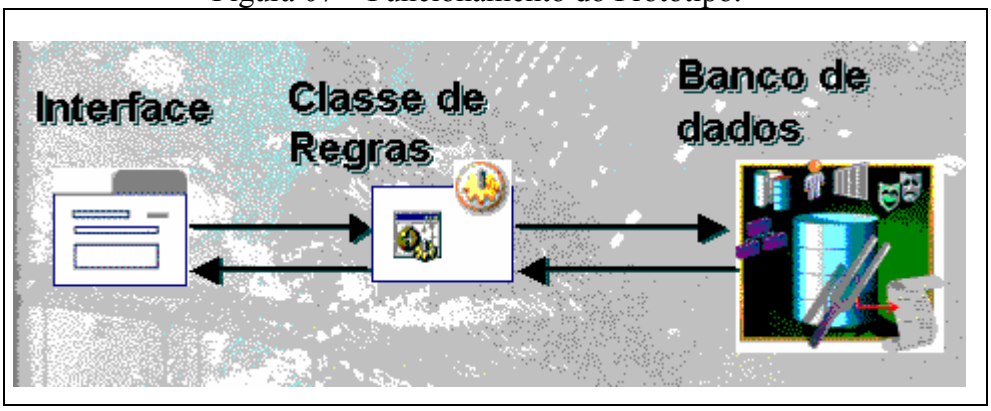
a) interface: apresentação das telas e interface com o usuário do sistema,

b) classe de regras: responsável pelo processamento, interpreta a solicitação recebida da interface , passado o comando DML para o banco de dados.

c) banco de dados: responsável pela execução dos camandos recebidos da classe de regras.

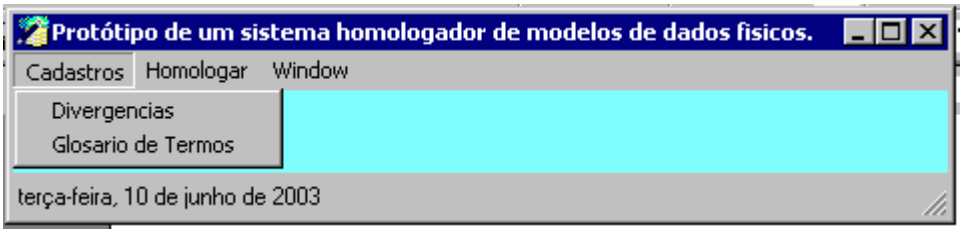
Resumidamente, o protótipo possui a interface que passa uma mensagem para a classe de regras que por sua vez interpreta a solicitação, chamando o banco de dados que executa e retorna a solicitação para a classe de regras, que então devolve para a interface. Veja na figura 07 o funcinamento do protótipo.

Figura 07 – Funcionamento do Protótipo.



Conforme mostra a fig. 08, o protótipo é composto por uma tela inicial que contém um menu com chamadas para as telas de cadastros e a tela de homologação.

Figura 08 - Tela principal do sistema.



Para se realizar a homologação de um modelo físico de banco de dados Oracle desenvolvido no C*ase Designer 6.i,* deve-se realizar duas etapas.

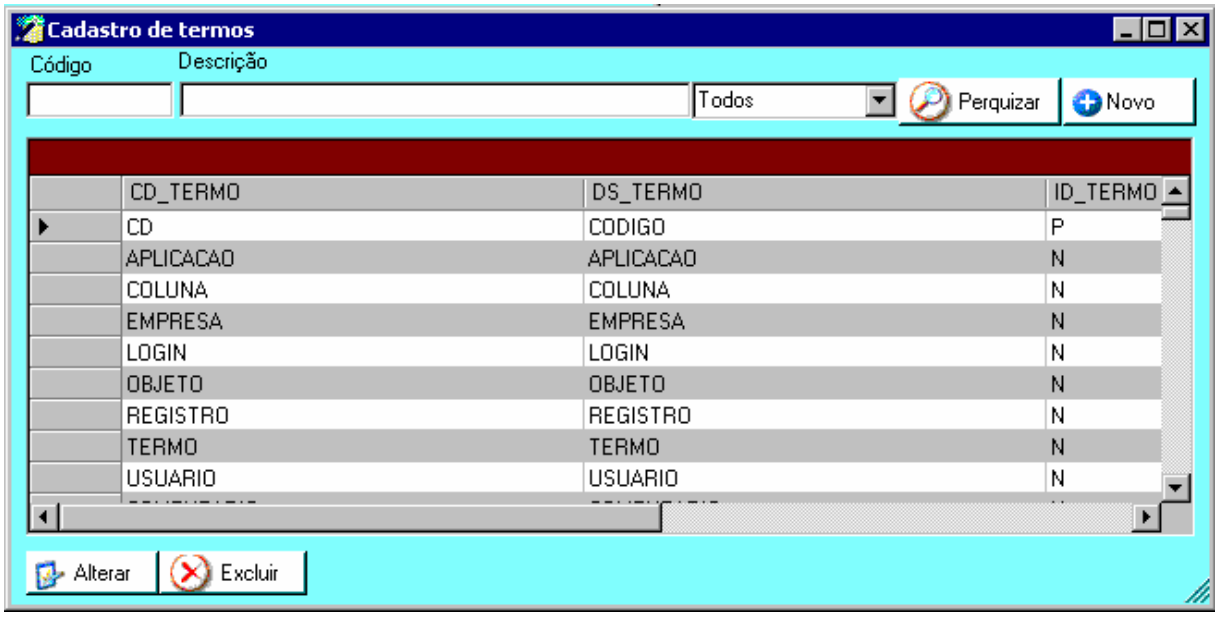
Na primeira etapa, o administrador de dados deve cadastrar o texto das divergências na base de dados. O cadastro é realizado na tela “Cadastro de divergências” que é acessada pelo menu Cadastro/Divergências. Essa tela pode ser vista pela fig. 09.

Figura 09 - Tela “Cadastro de Divergencias”

A tela mostrada pela fig. 09 “Cadastro de divergências”, permite ao administrador de dados realizar as operações de: inserção, alteração, exclusão e pesquisa.

Na segunda etapa o administrador de dados deve popular o glossário de termos, que consiste em informar o padrão de nomenclatura de objetos do banco de dados e alguns descritores diversos. Esses termos serão usados para validação da nomenclatura. A demonstração da tela “Cadastrar glossário de termos” é feita pela fig 10.

Figura 10 - Tela “Cadastro de termos”

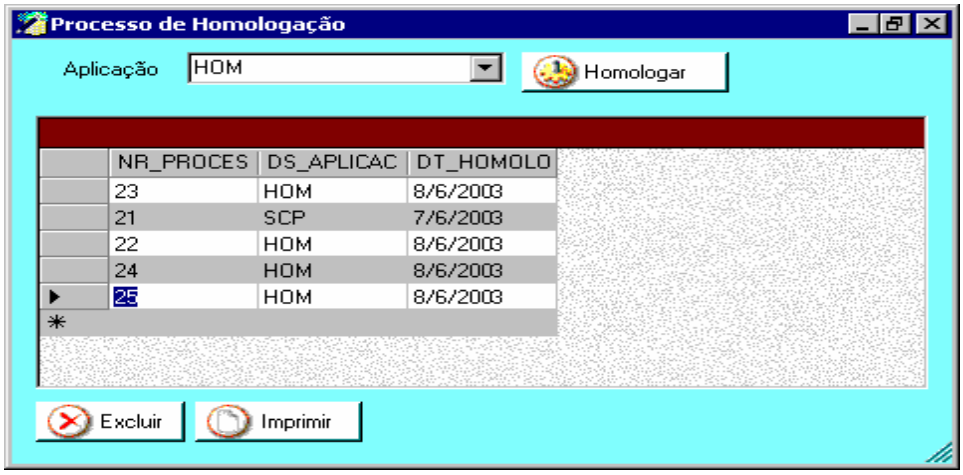


A tela mostrada pela fig. 10 "Cadastro do glossário de termos", permite ao administrador de dados realizar as operações de: inserção, alteração, exclusão e pesquisar.

Caso não venha a ser realizado os cadastros de divergências e termos, a homologação não poderá ser feita, pois esses cadastros são essenciais, pois definem os padrões de desenvolvimento do banco de dados.

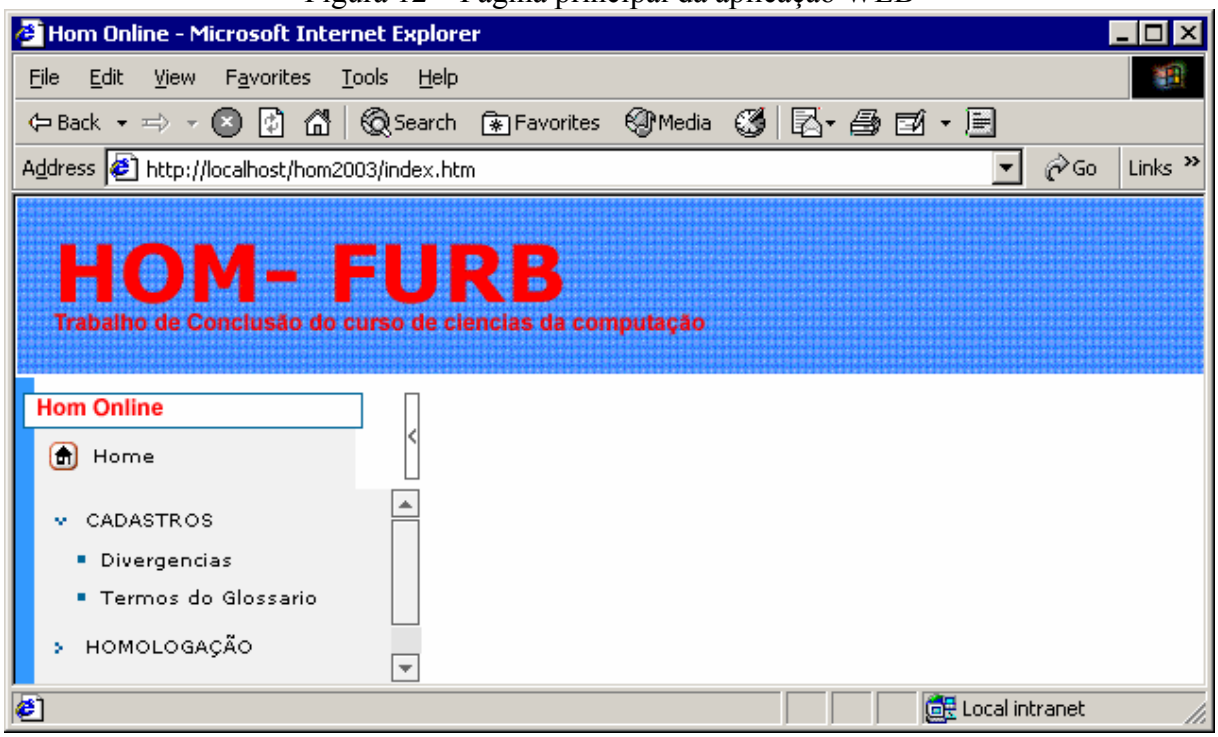
Finalmente para realizar a homologação o desenvolvedor do modelo físico deve acessar o menu principal do sistema ecolhendo o item "Homologar". Este item abre a tela de homologação, que trará um campo do tipo *combobox* com todas as aplicações desenvolvidas no C*ase Designer 6.i.* Escolhe-se a aplicação e pressiona-se o botão "homologar. Neste momento o sistema chama o processo de homologação retornado para o sistema o código da homologação, que é demonstrado no"Grid de aplicações homologadas". O desenvolvedor então poderá imprimir o relatório contendo as divergências encontradas, devendo selecionar a aplicação desejada no grid com um clique e pressionar o botão "Imprimir". Para excluir os processo de homologação antigos basta selecionar o registro no grid com um clique do *mouse* na linha desejada e pressionar o botão Excluir. Estas funcionalidades são apresentadas na fig. 11.

Figura 11 - Tela "Homologar aplicação"



Para demonstrar o uso da OO durante o desenvolvimento do protótipo, foi desenvolvida a aplicação para WEB com todas as funcionalidades da aplicação para *desktop*. Para isso foi necessário desenvolver apenas a interface da aplicação, utilizando as mesmas classes e o memo banco. A fig. 12 mostra o pagina principal da aplicação WEB.

Figura 12 – Pagina principal da aplicação WEB



O menu lateral possui link para as paginas de cadastros e também para a pagina da homologação. Veja na figura 13 o cadastro de divergencias e na figura 14 o cadastro de termos.

Figura 13 – Cadastro de divergencias WEB

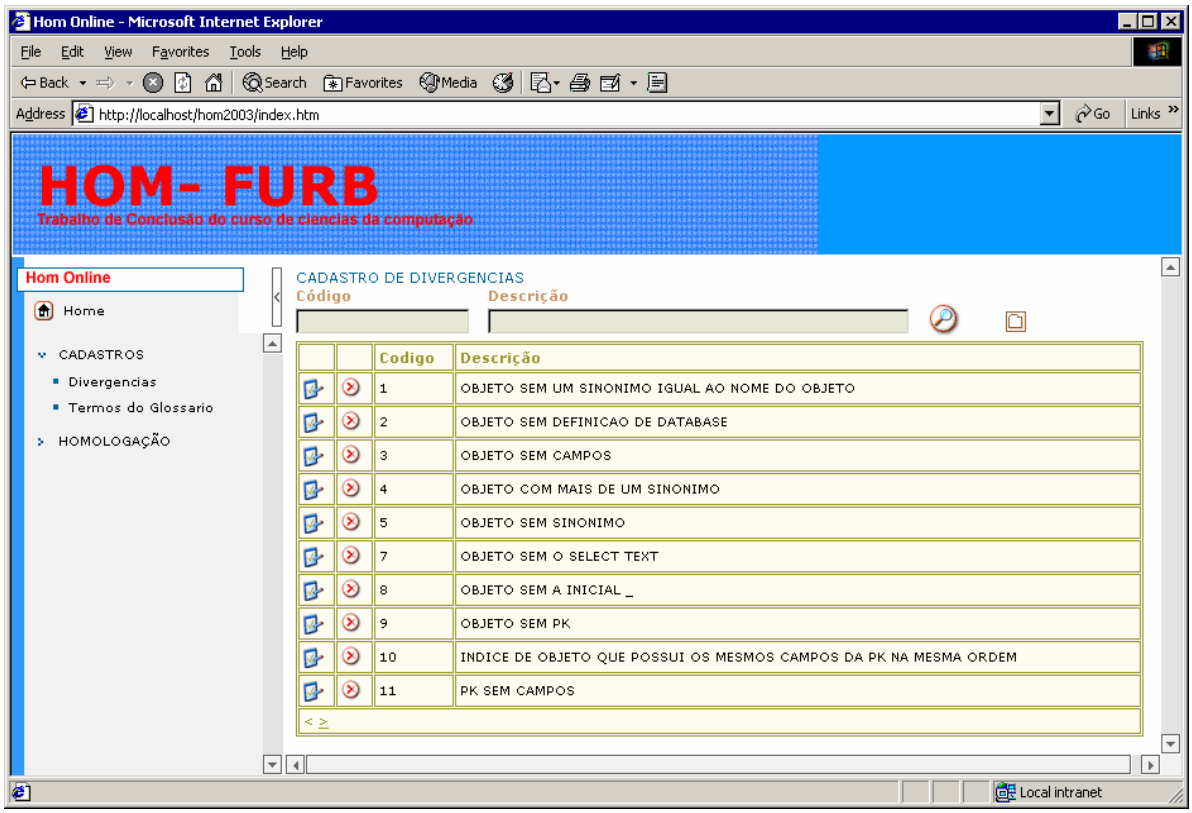
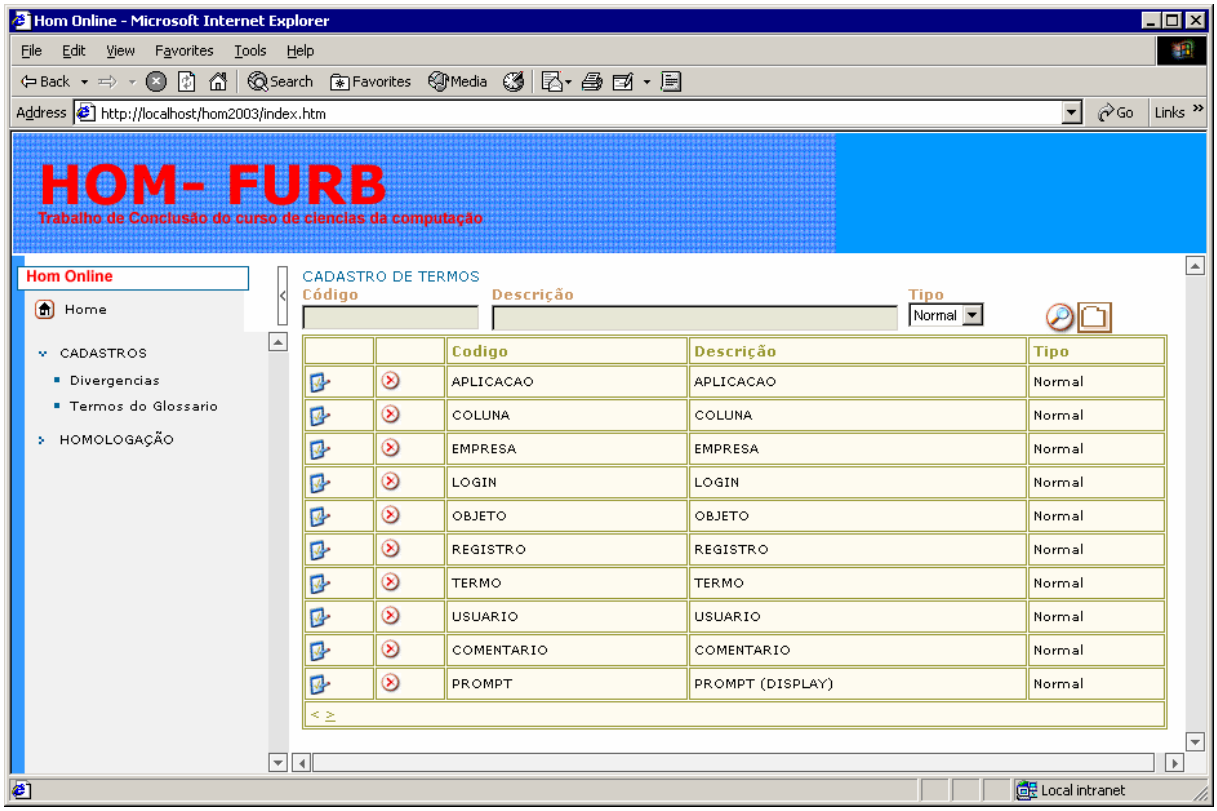
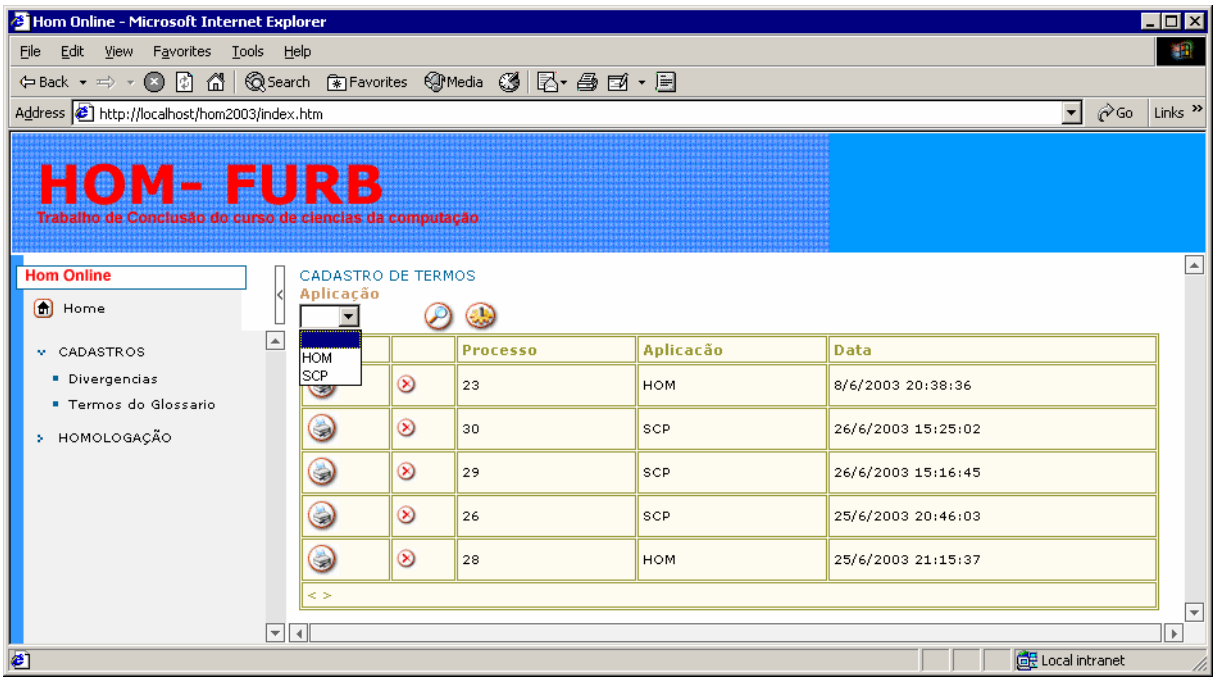


Figura 14 – Cadastro de termos WEB



A pagina da homologação pode ser visualizada na fig. 15.

Figura 15 – Pagina de homologação WEB



5.5 RESULTADOS E DISCUSSÃO

Neste item é apresentado um estudo de caso com o modelo físico do banco de dados implementado no *Case Designer 6.i.* O estudo de caso será submetido ao sistema de homologação, onde será gerado o relatório das divergências encontradas. Este processo será executado a quantidade de vezes necessárias para que o sistema seja homologado.

5.5.1 ESTUDO DE CASO

Conforme Grahl (2003), uma entidade ambientalista decidiu criar um banco de dados sobre as pescas realizadas na sua região de atuação, a fim de disponibilizar dados de interesse dos pescadores, entidades de pescadores e a comunidade em geral. Para isso deseja-se realizar um censo onde possam ser coletadas as seguintes informações:

a) dados sobre embarcações: proprietário, nome da embarcação, comprimento, inscrição na capitania dos portos, ano construção. Sobre o proprietário são cadastrados o nome, endereço, CPF, apelido e município;

b) as embarcações podem ser para pesca artesanal ou industrial. Quando for barco para pescaria industrial, devem ser armazenados ainda dados como a capacidade de

estocagem e se a embarcação possui tanque de isca. No caso da pesca artesanal devem ser informados ainda o tipo de material do casco e o tipo de propulsão do barco (motor, vela, remo, etc);

c) para cada embarcação serão armazenados os diversos tipos de petrechos de pesca utilizados (rede, caniço, etc) e também o tipo de conservação do pescado (refrigerado, sem refrigeração , etc).

Cabe aos pesquisadores o cadastramento das espécies de animais encontrados na área marítima considerada pelo sistema. Sobre cada espécie, é levantado: o codigo, nome científico e o nome popular.

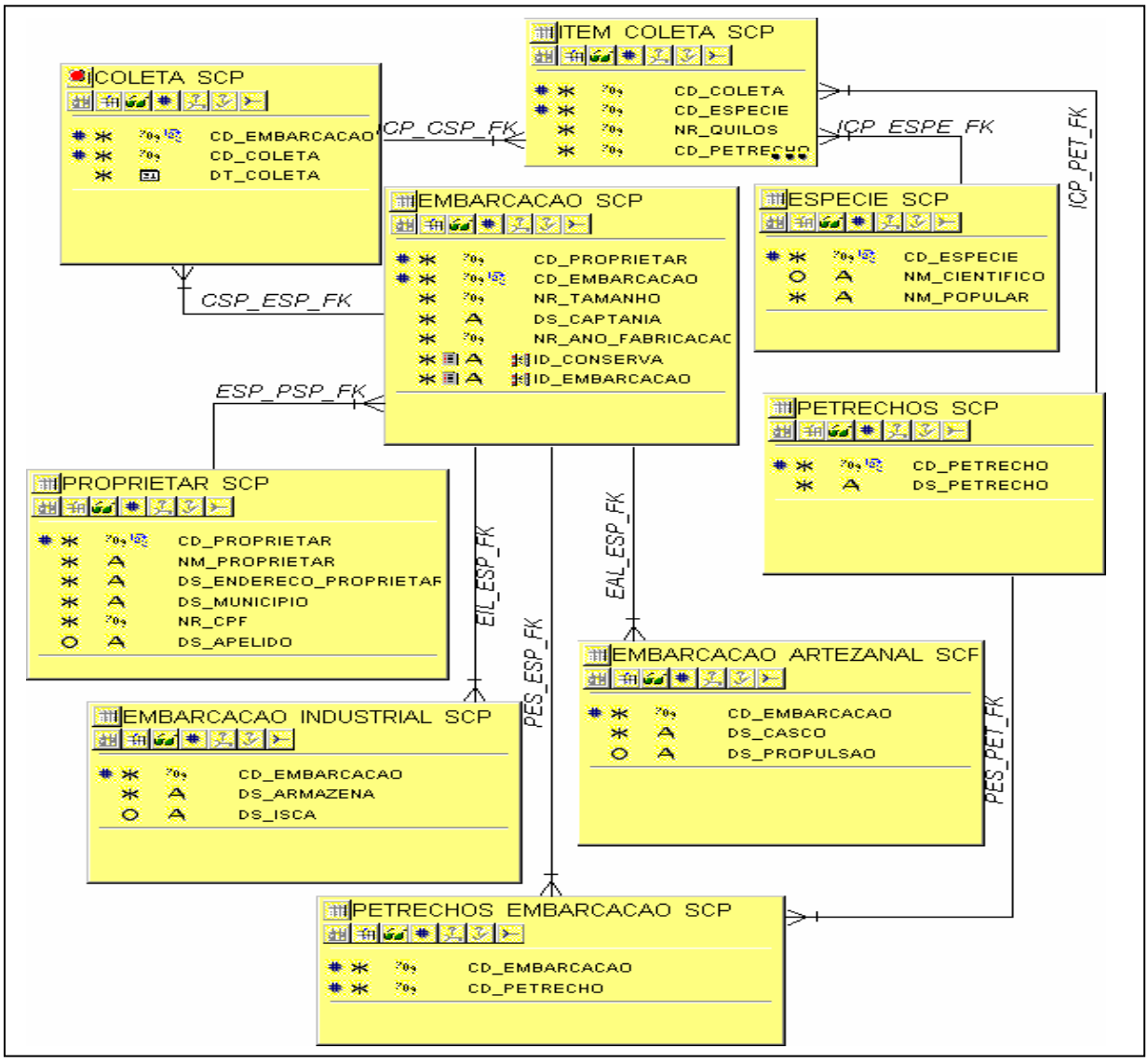
Os fiscais vão informar os dados coletados sobre as pescas, que foram anotadas nos pontos de desembarque, registrando a data, hora, embarcação e para cada espécie capturada, será registrada a quantidade em quilos obtida e o petrecho utilizado para sua captura.

Sobre as pescas realizadas (desembarques), o sistema disponibilizará aos usuários em geral dois relatórios mensais: quantidade total pescada por espécie e embarcações com maior quantidade de pesca (quilos) utilizando rede.

5.5.2 RESOLUÇÃO DO ESTUDO

Após realizada a análise do estudo de caso foi elaborado o modelo de dados físico. O modelo foi desenvolvido na ferramenta CASE *Designer 6.i, e* a sigla do sistema foi definida como “SCP” que significa “Sistema de controle de Pescas”. O modelo físico do sistema SCP pode ser ser visto pela fig. 16.

Figura 16 - Modelo de dados físico do estudo de caso.



O modelo físico, mostrado pela fig. 16, possui algums erros de modelagem, deixados propositalmente. Para identificar os erros será submetido ao sistema de homologação que verificará a consistência com o padrão de desenvolvimento.

O quadro 08 apresenta o relatório de divergências apontado pelo sistema de homologação, após a submissão do mesmo as rotinas de checagem.

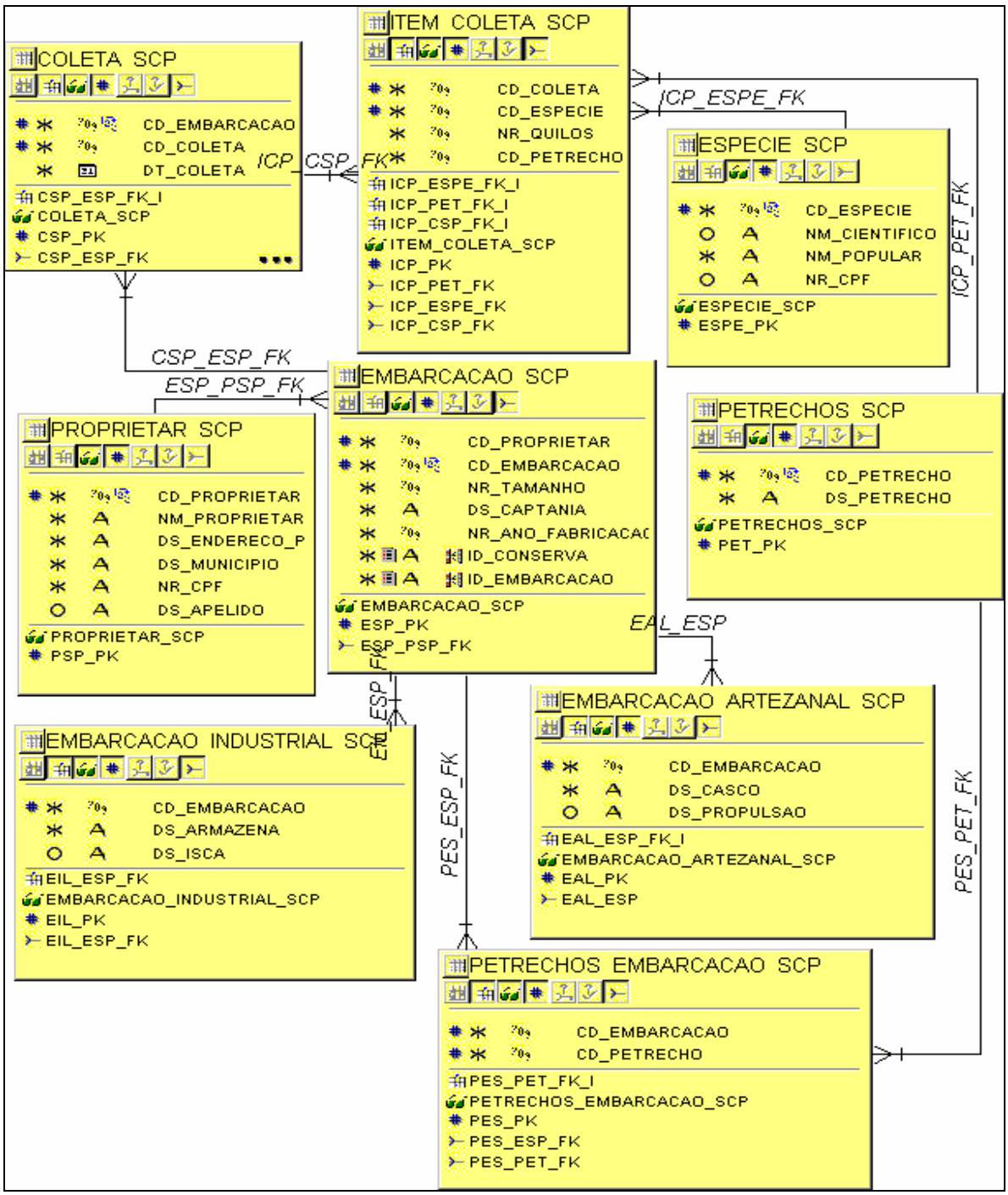
Quadro 08 – Relatório de Divergências

```
_____________________________________________________________________________________________
                                                                        Geração: 6/6/2003 00:10:44
                        RELATORIO DAS DIVERGÊNCIAS                      Pagina.:  1

_____________________________________________________________________________________________

 Processo.....:  9
 Aplicação.....: SCP
 Data Homogação.: 5/6/2003 22:46:24
_____________________________________________________________________________________________

TIPO DE OBJETO: COLUMN
_____________________________________________________________________________________________

 NOME DO OBJETO:  NR_CPF
            CAMPO QUE POSSUE O MESMO NOME COM DIFERENTE DATATYPE
            CAMPO QUE POSSUE O MESMO NOME COM DIFERENCA DE MAXIMUM LENGHT

_____________________________________________________________________________________________

TIPO DE OBJETO: TABLE
_____________________________________________________________________________________________


Nome do Objeto: EMBARCACAO_SCP
              -->OBJETO SEM START ROWS OU SEM END ROWS
              -->TERMO NAO ENCONTRADO NO GLOSSARIO DE TERMOS -- SCP
              -->OBJETO SEM DATABASE OBJECT GRANTS
              -->NAO ENCONTRADA ROLE R_<INICIAL DA APLICACAO>_GERAL
              -->NAO ENCONTRADA ROLE R_<INICIAL DA APLICACAO>_CONSULTA
              -->OBJETO SEM DESCRIPTION
              -->OBJETO SEM DEFINICAO TABLESPACE

            Coluna: CD_PROPRIETAR
                  -->TERMO NAO ENCONTRADO NO GLOSSARIO DE TERMOS -- PROPRIETAR
                  -->CAMPO SEM HINT
                  -->CAMPO SEM DESCRIPTION

            Coluna: CD_EMBARCACAO
                  -->CAMPO SEM HINT
                  -->CAMPO SEM DESCRIPTION

            Coluna: NR_TAMANHO
                  -->CAMPO SEM HINT
                  -->CAMPO SEM DESCRIPTION

            Coluna: DS_CAPTANIA
                  -->TERMO NAO ENCONTRADO NO GLOSSARIO DE TERMOS -- CAPTANIA
                  -->CAMPO SEM HINT
                  -->CAMPO SEM DESCRIPTION

            Coluna: NR_ANO_FABRICACAO
                  -->CAMPO SEM HINT
                  -->CAMPO SEM DESCRIPTION

            Coluna: ID_CONSERVA
                  -->TERMO NAO ENCONTRADO NO GLOSSARIO DE TERMOS -- CONSERVA
                  -->CAMPO SEM DESCRIPTION
                  -->CAMPO SEM HINT

            Coluna: ID_EMBARCACAO
                  -->CAMPO SEM HINT
                  -->CAMPO SEM DESCRIPTION

            CONSTRAINT/INDEX: ESP_PSP_FK
                  -->INDICE DE OBJETO QUE POSSUI OS MESMOS CAMPOS DA PK NA MESMA ORDEM
                  -->INDICE COM NOME DE DIFERENTE DE: <NOME DA FK>_I OU <ALIAS OBJETO
                        CORRENTE>_SK_<NUMERO INICIADO DE 01>
```

```
_____________________________________________________________________________________________________
Nome do Objeto: PETRECHOS_EMBARCACAO_SCP
        -->OBJETO SEM START ROWS OU SEM END ROWS
        -->TERMO NAO ENCONTRADO NO GLOSSARIO DE TERMOS -- PETRECHOS
        -->TERMO NAO ENCONTRADO NO GLOSSARIO DE TERMOS -- SCP
        -->OBJETO SEM DATABASE OBJECT GRANTS
        -->NAO ENCONTRADA ROLE R_<INICIAL DA APLICACAO>_GERAL
        -->NAO ENCONTRADA ROLE R_<INICIAL DA APLICACAO>_CONSULTA
        -->OBJETO SEM DESCRIPTION
        -->OBJETO SEM DEFINICAO TABLESPACE

    Coluna: CD_EMBARCACAO
        -->CAMPO SEM HINT
        -->CAMPO SEM DESCRIPTION

    Coluna: CD_PETRECHO
        -->TERMO NAO ENCONTRADO NO GLOSSARIO DE TERMOS -- PETRECHO
        -->CAMPO SEM HINT
        -->CAMPO SEM DESCRIPTION

    CONSTRAINT/INDEX: PES_ESP_FK_I
        -->INDICE DE OBJETO QUE POSSUI OS MESMOS CAMPOS DA PK NA MESMA ORDEM

_____________________________________________________________________________________________________
TIPO DE OBJETO: VIEW
_____________________________________________________________________________________________________
Nome do Objeto: SCP_ESPECIE
        -->TERMO NAO ENCONTRADO NO GLOSSARIO DE TERMOS -- SCP
        -->OBJETO SEM SINONIMO
        -->OBJETO SEM PK
        -->PK SEM CAMPOS
        -->OBJETO SEM DATABASE OBJECT GRANTS
        -->NAO ENCONTRADA ROLE R_<INICIAL DA APLICACAO>_GERAL
        -->NAO ENCONTRADA ROLE R_<INICIAL DA APLICACAO>_CONSULTA
        -->OBJETO SEM DESCRIPTION

    Coluna: CD_ESPECIE
        -->CAMPO SEM HINT
        -->CAMPO SEM DESCRIPTION

    Coluna: NM_POPULAR
        -->TERMO NAO ENCONTRADO NO GLOSSARIO DE TERMOS -- POPULAR
        -->CAMPO SEM HINT
        -->CAMPO SEM DESCRIPTION

_____________________________________________________________________________________________________
TIPO DE OBJETO: SEQUENCE
_____________________________________________________________________________________________________
Nome do Objeto: SEQ_COLETA_SCP
        -->TERMO NAO ENCONTRADO NO GLOSSARIO DE TERMOS -- COLETA
        -->TERMO NAO ENCONTRADO NO GLOSSARIO DE TERMOS -- SCP
        -->OBJETO SEM DATABASE OBJECT GRANTS
        -->OBJETO SEM DESCRIPTION
        -->NAO ENCONTRADA ROLE R_<INICIAL DA APLICACAO>_GERAL
        -->NAO ENCONTRADA ROLE R_<INICIAL DA APLICACAO>_CONSULTA

_____________________________________________________________________________________________________
TIPO DE OBJETO: DOMAIN
_____________________________________________________________________________________________________
Nome do Objeto: ID_CONSERVA
        -->OBJETO SEM DESCRIPTION

Nome do Objeto: ID_EMBARCACAO

        -->OBJETO SEM DESCRIPTION
```

Pode-se verificar que foram encontradas muitas divergências na primeira homologação do modelo de dados físico do sistema "SCP", divergências que foram corrigidas. A figura 17 mostra o modelo físico do sistema SCP após correções realizadas.

Figura 17 - Modelo de dados físico após correção.



Após executar novamente a homologação foi gerado mais um relatório, onde pode-se verificar que não foi mais encontrada nenhuma divergência, e o sistema concluiu que a

aplicação “SCP” está “HOMOLOGADA”. Veja no quadro 09 o relatório contendo a confirmação da homologação do sistema.

Quadro 09 – Sitema SCP Homologado.

Geração: 6/6/2003 00:10:44
RELATORIO DAS DIVERGENCIAS Pagina.: 1

Processo.....: 9
Aplicação.....: SCP
Data Homogação.: 5/6/2003 22:46:24

Não foram encontradas mais divergencias para está aplicação

A aplicação SCP está HOMOLOGADA

6 CONSIDERAÇÕES FINAIS

6.1 CONCLUSÃO

Foram encontradas dificuldades na pesquisa bibliográfica relacionada à homologação de modelos de dados físicos, bem como a documentação do metadados que gerou bastante trabalho para identificação da estrutura de armazenamento do modelo.

O objetivo principal deste trabalho que foi desenvolver um protótipo de uma ferramenta de Homologação de Modelos de dados Físicos desenvolvidos no *Case Designer 6.i,* foi atentido. Os objetivos específicos também foram atendidos, uma vez que, o protótipo a partir da leitura do repositório do *Case Designer 6.i* é capaz de identificar todas as divergências e regras descritas nos padrões propostos, gerando também um relatório contendo tais divergências.

Os padrões propostos visam padronizar o projeto fisico de banco de dados dando clareza à nomenclatura e definindo a composição de cada objeto do banco de dados. Cabe ressaltar que cada empresa pode elaborar seus padrões mas eles devem existir para uma melhor legibilidadade no desenvolvimento.

Em relação à ferramenta de desenvolvimento Visual Studio.Net verrificou se que a mesma tem um suporte muito forte à orientação a objetos e ao tratatamento de erros. Isso possibilita uma interface mais limpa e um desenvolvimento mais ágil, sendo possível ainda reutilizar os componentes desenvolvidos. A reutilização das classe do protótipo foi demostrada implementando a aplicação para desktop e para WEB.

6.2 LIMITAÇÕES

O protótipo está limitado a homologar apenas modelo de dados físico desenvolvidos na ferramenta *Case Designer 6.i* da Oracle.

Outra limitação é a necessidade de alterar ou atualizar o programa a cada nova regra implementada.

6.3 EXTENSÕES

Como trabalhos futuros sugere-se a implementação deste para outros SGBDs, validando outros tipos de objetos, bem como validar também extruturas geradas a partir de *scripts sql*. Outra sugestão seria deixar a ferramenta mais flexível, criando-se um banco de regras e a partir desse banco a ferramenta indicasse as divergências.

REFERÊNCIAS BIBLIOGRÁFICAS

AULT, Michael R. **Oracle 7** - Adminitração & Gerenciamento. Rio de Janeiro: Infobook, 1995.

BARROS, Pablo. **UML:** Linguagem de modelagem unificada. dez 2000. Disponível em <http://www.unit.br>. Acessado em 30 out. 2002.

BOOCH, Grady. **UML guia do usuário.** Rio de Janeiro: Campus, 2000.

BORBINHA, José. **Metadados**, out 2003. Disponível em <http://metadados.bn.pt>. Acessado em 10 mai. 2003.

CERICOLA, Vicent Oswald. **Oracle:** banco de dados relacional e distribuído. São Paulo: Makron Books 1995.

FERNANDES, Lucia. **Oracle8/8i curso completo.** Rio de Janeiro: Axcel Books, 2000.

GRAHL, Everando Artur. **Notas de Aula.** fev 2002. Disponível em: <http://www.inf.furb.br/~egrahl/>. Acesso em 10 mai. 2003.

HADDAD, Renato. **Visual Basic.net:** conceitos e aplicações**.** São Paulo: Erica, 2001

HARRINGTON, Jan L. **Projetos de banco de dados relacionais:** teoria e pratica. Rio de Janeiro: Campus, 2002.

HEUSER, Carlos Alberto. **Projeto de banco de dados.** Porto Alegre: Sagra Luzzatto, 2000.

MATOS, Alexandre Veloso. **UML**: Pratico e Descomplicado. São Paulo: Erica, 2002.

PAULA, Wilson de Pádua . **Engenharia de software**: fundamentos, métodos e padrões. Rio de Janeiro: LTC-Livros Técnicos e Científicos Editora S.A, 2001.

QUATRANI, Terry. **Modelagem visual com Rational Rose 2000 e UML**. Rio de Janeiro: Ciência Moderna Ltda, 2001.

RAMALHO, José Antonio. **Oracle8i.** São Paulo: Berkeley Brasil, 1999.

REZENDE, Denis Alcides. **Engenharia de software e sistemas de informação.** Rio de Janeiro: Brasport, 2002.

SHERER, Douglas. **Oracle8i:** dicas e técnicas. Rio de Janeiro: Campus, 2000.

ORACLE. **Oracle designer 6.i tutorial.** jul. 2000. Disponível em: <http://otn.oracle.com/docs/products/designer/doc_library/6i_release42/CTUT72/dsgnr_tuttitle_65.htm >. Acesso em: 18 nov. 2002.

VALDAMERI, Alexander Roberto. **Notas de Aula.** fev 2003. Disponível em: <http://www.inf.furb.br/~arv/>. Acesso em 10 mai. 2003.