Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis - IMESA

ERIK CORREA SANTOS

TÉCNICAS DE MODELAGEM DE BANCO DE DADOS

ASSIS

2015

ERIK CORREA SANTOS

TÉCNICAS DE MODELAGEM DE BANCO DE DADOS

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação.

Orientador: Prof. Dr. ALEX SANDRO ROMEO DE SOUZA POLETTO

ASSIS

2015

FICHA CATALOGRÁFICA

SANTOS, Erik Correa.

Técnica de Modelagem de Banco de Dados / Erik Correa Santos. Fundação Educacional do Município de Assis – FEMA: Assis 2015.

Nº. 71

Orientador**:** Prof. Dr. Alex Sandro Romeo de Souza Poletto

Trabalho de Conclusão de Curso - Instituto Municipal de Ensino Superior de Assis – IMESA.

1. Modelagem 2. Banco de Dados 3. Normativas. 4. IDEF1X 5. UML 6. Formas Normais

CDD: 001.6

Biblioteca da FEMA

TÉCNICAS DE MODELAGEM DE BANCO DE DADOS

ERIK CORREA SANTOS

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, analisado pela seguinte comissão examinadora:

Orientador: Prof. Dr. Alex Sandro Romeo de Souza Poletto

Analisador: Prof. Guilherme de Cleva Farto

Assis

2015

DEDICATÓRIA

Dedico todo meu esforço primeiramente a Deus, e a minha família que sempre me apoiou nesta trajetória, em especial ao meu pai Robson, porque sem Eles, eu nada seria!

AGRADECIMENTOS

Quero agradecer primeiramente, a Deus por ter me dado força e saúde para que eu possa superar as dificuldades encontradas.

A minha mãe Vera e irmãos Davi e Daniel, por acreditar em meu potencial e ter me apoiado em toda a trajetória do curso.

Ao meu pai, que infelizmente não esta mais entre-nos, mais sei que onde estiver esta orgulhoso em ver um filho realizar seu sonho de se formar em um curso superior.

A minha avó Antonia e avô Laurindo, com seus humildes conhecimentos, me ajudaram compreender melhor a vida, e sempre apoiando no que fosse preciso.

A minha namorada Patrícia, pelo seu apoio e carinho nesses anos de convívio.

Aos meus amigos, que tive o prazer de conhecer no decorrer desses anos, e pela força nos estudos compartilhando seus conhecimentos, sempre com muito empenho e humor. Esses anos de convívio não serão esquecidos.

A FEMA, seu corpo docente, que oportunizaram o aumento de meus conhecimentos na área cursada.

Ao meu Orientador acadêmico Professor Doutor Alex Sandro Romeo de Souza Poletto, e Prof. Guilherme de Cleva Farto, que analisou este trabalho.

E a todos que direto ou indiretamente fizeram parte de minha formação, o meu muito obrigado.

.

“Não deixe de fazer algo que gosta devido à falta de tempo; a única falta que terá, será desse tempo que infelizmente não voltará jamais”

Mario Quintana.

RESUMO

O presente trabalho de conclusão de curso tem como objetivo pesquisar e compreender a importância da modelagem de dados no desenvolvimento de software, alcançando melhores resultados. Este trabalho vai apresentar a normativa IDEF1X, bem como os diagramas e conceitos relacionados à UML, no sentido de avaliar os métodos utilizados nas modelagens de dados estruturais e orientado a objetos, utilizados no desenvolvimento de software. Apesar desse processo de desenvolvimento de software ser bastante comentado no meio acadêmico, muitos deixam de dar a devida importância na aplicação das técnicas de modelagem, e na maioria das vezes até surge um novo olhar para o software desenvolvido, o que pode ser bom ou ruim, dependendo da analise. Para concluir a pesquisa, será elaborado um projeto normalizado, no sentido de avaliar essas duas técnicas de modelagem de dados, que contribuirá como os resultados finais dessa pesquisa.

Palavras- Chave: 1.Modelagem 2.Banco de Dados 3.Normativas 4.IDEF1X 5.UML 6.Formas Normais

ABSTRACT

This final term paper aims to search for and understand the importance of data modeling in software development, achieving better results. The IDEF1X rules will be presented, as well as diagrams and concepts regarding to UML, in order to evaluate the methods used in modeling of structural and object-oriented data in software development. Despite this process of software development being quite discussed in academia many fail in giving modeling techniques the due importance and most of time a new look at the software developed might appear, which can be bad or good depending on the analyses. To conclude this research it will be elaborated a project to assess two data modeling techniques which will collaborate to the final result of this paper.

Key-words: 1.Modeling 2. Database. 3. Rules 4. IDEF1X 5.UML 6.Normal forms

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

BD	Banco de Dados
BDOO	Banco de Dados Orientado ao Objeto
BDOR	Banco de Dados Objeto Relacional
BDR	Banco de Dados Relacional
DBA	Administrador de Banco de Dados
DER	Diagrama Entidade Relacionamento
DML	Linguagem de manipulação de Dados
IDEF1X	Definição de integração para modelagem de informações
LPOO	Linguagem de Programação Orientada a Objetos
MER	Modelo Entidade-Relacionamento
NIST	Instituto Nacional de Padrões e Tecnologia
OO	Orientação Objetos
SGBD	Sistema Gerenciador de Banco de Dados
SGBDR	Sistema Gerenciador de Banco de Dados Relacionais
SGBDOO	Sistema Gerenciador de Banco de Dados Orientado a Objetos
SGDBOR	Sistema de Banco de Dados Orientado Relacional
SQL	Linguagem de Consulta Estruturada
UML	Linguagem Unificada de Modelagem

SUMÁRIO

1. INTRODUÇÃO

É comum nos dias de hoje ouvir falar sobre informação, ou a sociedade do conhecimento, porque tudo ao nosso redor se torna informação.

Por essa razão sentimos a necessidade de armazenar certos dados, por isso, os bancos de dados adquiriram uma grande importância em nosso meio, tornando-se indispensáveis para a organização, além de ser um meio rápido e fácil de armazenar e de recuperar informações.

Essa organização torna-se cada vez mais trabalhosa, ainda mais com relação ao inter-relacionamento entre as informações, e com as dificuldades de colocá-las em ordem, dentre outros. Para solucionar esses problemas foram criados os Sistemas de Banco de Dados (SGBD), onde é possível guardar grande quantidade de informações e assim manipulá-los conforme necessidade.

Os Sistemas de Banco de Dados fazem uma grande diferença no dia-dia, sendo indispensáveis para a organização dos dados, visto que se torna impossível o controle por intermédio do uso de papel, como era feito antigamente. Além disso, para que estas informações sejam armazenadas com qualidade, é necessário um bom projeto de banco de dados.

E para seu software ser de qualidade e bem projetado é essencial fazer um planejamento detalhado, como a arquitetura a ser utilizado, estimar custos, tempo e ferramentas, técnicas que serão utilizadas no decorrer do projeto.

1.1 OBJETIVO

O objetivo principal desde trabalho é pesquisar e compreender os conceitos das técnicas de modelagem, e as ferramentas empregadas nesse processo.

Dentre varias técnicas de modelagem existente no mercado, vamos estudar em particular sobre a normativa IDEF1X e os conceitos da UML.

O estudo de caso será a Implementação de um banco de dados seguindo os procedimentos de normalização.

Com o desenvolvimento desde estudo de caso, será possível realizar uma comparação entre essas duas técnicas em um banco de dados orientado a objeto e outro relacional, no sentido de apresentar as vantagens e desvantagens, caso existam, e que cada uma dessas técnicas pode oferecer naatividade na modelagem de dados.

Para chegar a esses objetivos, o projeto foi divido em duas partes:

1º Fase: Pesquisar sobre a normativa IDEF1X, Pesquisar sobre a UML, Pesquisar sobre Formas Normais, Pesquisar sobre Banco de Dados Relacional, Pesquisar sobre Banco de Dados Orientados a Objetos.

2º Fase: Analisar e comparar as duas técnicas de modelagem, Implementar dois Banco de Dados (Relacional e OO) com esses conceitos, Analisar a diferença caso exista entre os dois projetos, testar, validar e descrever os resultados obtidos.

1.2 JUSTIFICATIVAS

Esse trabalho vem para contribuir com a conscientização da importância da modelagem de dados, sendo considerada por muitos desenvolvedores a fase delicada e importante no desenvolvimento de um banco de dados.

Porem uma modelagem mal estruturada pode conter anomalias em seu banco de dados, como redundância dos dados, e integridade das informações.

A modelagem é muito importante, já que é a atividade responsável pelo projeto de banco de dados, contudo um DBA deve realizar um bom levantamento teórico, obtendo domínio sobre a modelagem e suas técnicas, algo primordial para quem quer trabalhar com desenvolvimento de software.

1.3 MOTIVAÇÃO

A motivação para desenvolvimento desse projeto é que as empresas estão cada vez mais exigentes na aquisição de software, e quando tratamos de modelagem é de fato, um requisito para adquirir produtos de qualidade.
É importante destacar que é mais fácil e pratico em um projeto considerar essa fase, aplicando uma modelagem bem estruturada, do que perder tempo e dinheiro com manutenção em seu projeto futuramente, e a perca de confiança e satisfação do usuário, ainda mais com a grande quantidade de concorrentes no mercado de software.

1.4 ESTRUTURAS DO TRABALHO

Este trabalho esta estruturado nas seguintes partes:

- Capitulo 1 – Introdução
- Capitulo 2 – Conceitos sobre Banco de Dados
- Capitulo 3 – Formas Normais (Normalização de Dados)
- Capitulo 4 – Técnicas de Modelagem de Dados
- Capitulo 5 – Proposta dos Trabalhos - Comparações das Técnicas
- Capitulo 6 – Estudo de Caso
- Capitulo 7 – Considerações Finais
- Referencias

2. CONCEITOS SOBRE BANCO DE DADOS

Banco de dados (BD) são coleções de dados interligados entre si e organizados para fornecer informações, muitos consideram dados e informações como palavras sinônimas, mais existe diferenças.

Definimos Dados, como sendo aspectos do mundo real que sentimos a necessidade de armazenar informação a respeito, e na maioria das vezes os dados não fazem sentidos sozinhos.

Informação consiste na junção dos dados de forma organizada para ter sentido, e gerar conhecimento.

Todo dado relativo a outro dado, chamamos de metadados. Assim metadados são informações que acrescem aos dados e que tem o objetivo de informar sobre eles para tornar mais fácil a organização.

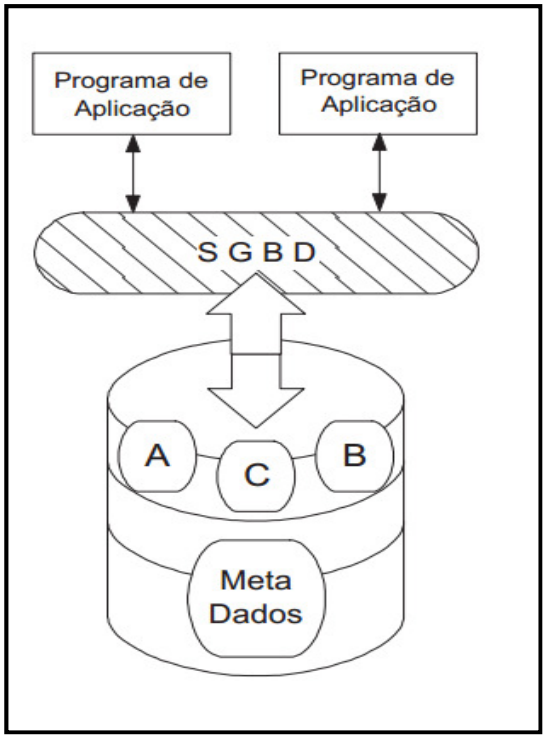

Figura 1 – Representação de um Sistema de Banco de Dados

Segundo Date (2003, p.6), Banco de Dados é um sistema computadorizado cuja finalidade geral é armazenar informações e permitir que os usuários busquem e atualizem essas informações quando as solicitar.

Um banco de dados pode ser local, ou seja, instalado em uma máquina (Cliente) e utilizado apenas por um usuário ou centralizado (servidor), e fica acessível por vários usuários simultaneamente, como mostra a Figura 2.

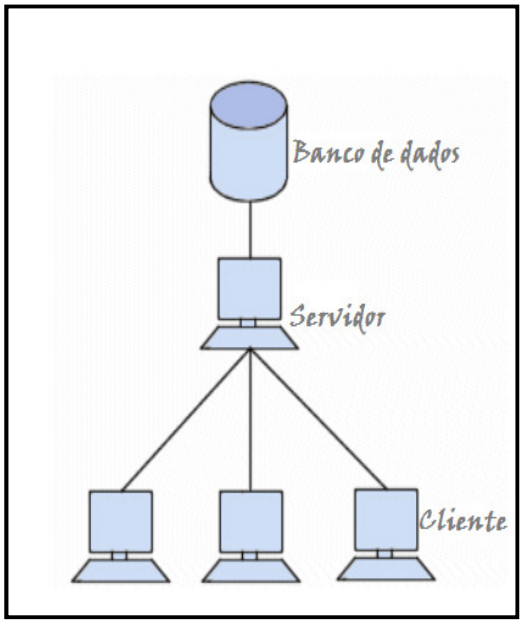

Figura 2: Modelo Centralizado

Existem vários modelos de banco de dados, como descrevemos abaixo:

Modelo Pleno ou Dedutivo: Consiste em matriz simples, bidimensional, composta por elementos de dados, inteiros, números reais, enfim esse modelo é à base das planilhas eletrônicas, como exemplo Excel, OpenOffice, entre outras.

Modelo em Rede: Os primeiros trabalhos usando esse modelo foi em 1964, essa tecnologia permite que várias tabelas sejam usadas simultaneamente através dos apontadores. Assim as tabelas são ligadas através de referencias.

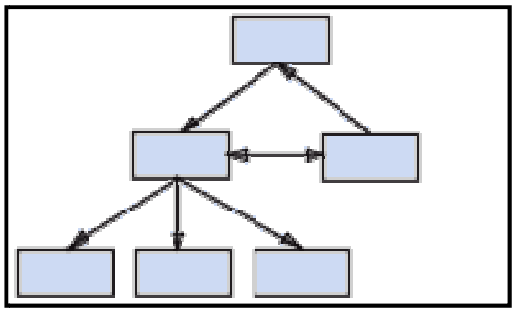

Figura 3 – Modelo em Rede

Modelo Hierárquico: Nesse modelo, os dados são conectados e limitados a uma estrutura semelhante a arvore.

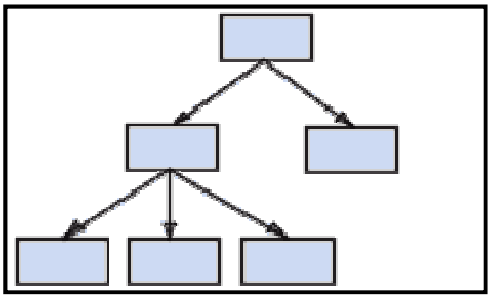

Figura 4 – Modelo Hierárquico

Modelo Relacional: Consistem em estruturas de dados (Tabelas), relações, uma coleção de operadores, álgebra e cálculos, restrições de integridades.

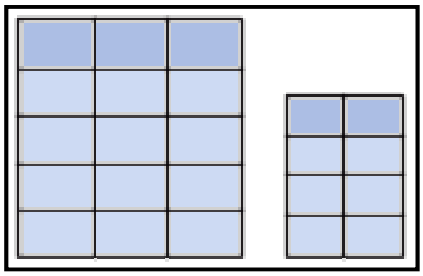

Figura 5 – Modelo Relacional

Modelo Orientado a Objeto: Cada informação é armazenada em forma de objetos, quer dizer de estruturas chamadas classes que apresentam dados membros.

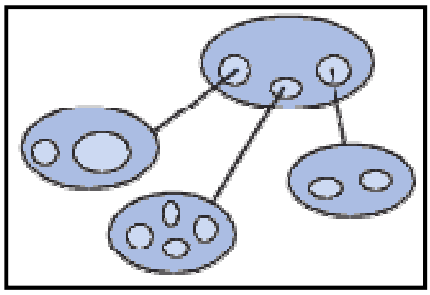

Figura 6 – Modelo Objeto

Modelo Objeto-Relacional: É semelhante ao Banco de dados relacional com alguns conceitos da orientação a objeto.

A decisão entre qual o modelo de banco de dados a ser utilizado vai depender do tipo de dados que você pretende armazenar. Nesse trabalho vamos estudar mais a fundo o Modelo relacional e Orientados a Objetos, que são os mais utilizados hoje.

Para administrar essas informações armazenadas em um Banco de Dados, necessitamos de um sistema gerenciador de Banco de Dados (SGBD) - do inglês Data Base Management System (DBMS). Tem como objetivo “fornecer uma maneira de recuperar informações de banco de dados que seja tanto conveniente quanto eficiente” (Silberschatz et al., 2006).

Esse sistema disponibiliza uma interface gráfica para que seus usuários possam incluir alterar, excluir, e consultar dados armazenados.

Dessa forma, tudo que realizamos em algum Banco de Dados passa pelo SGBD, ele é responsável por tudo, sem ele nada funciona.

No Capitulo 2.1 vamos tratar sobre o banco de dados relacional, e no Capitulo 2.3 vamos tratar do banco de dados orientado a objetos, analisando as características particulares de cada uma, e as diferenças entre elas.

2.1. BANCO DE DADOS RELACIONAIS

O Modelo de banco de dados relacional surgiu por meado da década de 70, por um matemático e pesquisador da IBM chamado Edgar Frank Codd, que escreveu o artigo “A Relational Model of Data for Large Shared Data Bank”, onde apresentou uma forma de usuários sem conhecimento técnico armazenarem e extraírem grandes quantidades de informações dos bancos de dados.

Mesmo o artigo sendo o marco do banco de dados relacional, ele foi explorado só no final da década de 70, onde foi criado um sistema baseado em suas idéias, chamado de “System R”, junto com ele a linguagem de consulta estruturada SQL (Structured Query Language) que se tornou a linguagem padrão para banco de dados relacionais.

Em 1985, Codd definiu 12 regras para definir o que é necessário para que SGBD fosse considerado relacional, sendo elas:

Regra 0: Todas as regras baseiam-se na noção de que para que um Bando de Dados seja considerado Relacional, ele deve utilizar os recursos relacionais exclusivamente para seu gerenciamento.

Regra 1: Informação – Todas as informações de um BDR devem ser representadas logicamente como valores de coluna em linhas dentro das tabelas.

Regra 2: Garantia de Acesso – Deve-se garantir que todos os valores de uma tabela possam ser acessados por meio de uma combinação de nome de tabela, valor de chave primaria e o nome de coluna.

Regra 3: Tratamento Sistemático de Nulos – Os nulos devem ser representados e tratados de modo sistemático, independente do tipo de dados.

Regra 4: Catálogo On-Line Dinâmico com Base no Modelo Relacional – Os metadados devem ser armazenados e gerenciados como dados comuns, ou seja, em tabelas no interior do BD. Esses dados devem estar disponíveis aos usuários autorizados, utilizando a linguagem relacional padrão do BD.

Regra 5: Sub-linguagem Ampla de Dados – O BDR pode suportar varias linguagens. No entanto deve suportar uma linguagem declarativa bem definida com suporte para definição de dados, definição de visualização, manipulação de dados (interativa ou por programa), restrições de integridade, autorização e gerenciamento de transações (iniciar, comprometer e desfazer).

Regra 6: Atualização de Visualização – Qualquer visualização que teoricamente possa ser atualizada deve ser por meio do sistema.

Regra 7: Inserção, atualização e exclusão de alto nível – O BD deve dar suporte a configuração do nível de inserções, atualizações e exclusões. Ou seja, a capacidade de manipular um conjunto de dados através de um comando, deve-se estender as operações de linguagem de manipulação de Dados (DML) como insert, updat e delete.

Regra 8: Independência Física de Dados – Aplicativos e recursos ad hoc não são afetados logicamente quando os métodos de acesso ou as estruturas de armazenamento físico são alterados.

Regra 9: Independência Lógica de Dados – Aplicativos e recursos ad hoc não são afetados logicamente quando de alterações de estruturas de tabelas que preservem os valores originais da tabela (alteração da ordem ou inserção de colunas). Alterações nas relações e nas views causam pouco ou nenhum impacto nas aplicações.

Regra 10: Independência de Integridade – Deve ser possível que todas as restrições de integridade relacional sejam definidas na linguagem relacional e armazenadas no catálogo de sistema, não no nível aplicação. As aplicações não devem ser afetadas quando ocorrer mudanças nas restrições de integridade.

Regra 11: Independência de Distribuição – Os usuários finais e aplicativos não conhecem nem são afetados pela localização dos dados (BD Distribuídos VS. BD Locais).

Regra 12: Não transposição das Regras – Se o sistema dá suporte a acesso de baixo nível aos dados, não deve haver um modo de negligenciar as regras de integridade do BD.

Segundo Silberschatz (2006, p.61), “O modelo relacional estabeleceu-se como o primeiro modelo de dados para aplicações comerciais”.

Hoje é o modelo de banco de dados mais utilizado no mercado, que conquistou essa posição pela sua simplicidade, revelou-se ser mais flexível e adequado na manipulação dos dados.

A estrutura fundamental do modelo relacional é a relação, que utiliza uma coleção de tabelas que se relacionam entre si, e que possui diversas colunas, cada coluna possui unicamente e exclusivamente um nome.

De maneira geral, o objetivo do projeto de banco de dados relacional é armazenar um conjunto de relação (tabelas), sendo aplicadas algumas restrições que permite armazenar essas informações sem redundâncias, alem de manipular e recuperar informações com facilidade.

Essas restrições de integridade e relacionamentos entre as tabelas são mantidas basicamente pela utilização das chaves primarias (Primary Key), Chave estrangeiras (Foreign key), que vamos analisar no Capitulo 5.

2.2. ORIENTAÇÃO A OBJETO

Segundo Larman (2004), a orientação a Objeto é um paradigma para o desenvolvimento de aplicações. Ela aproxima a linguagem de computadores ao mundo real, visando realizar a modelagem e codificações de uma forma mais natural. Com ela, o sistema fica organizado como uma tabela de objetos que interagem entre si.

A Linguagem Orientada a Objeto é uma extensão da Linguagem Modular e teve seu inicio nos anos 70, com a linguagem SIMULA-68, e logo em seguida nasceu o SMALLTALK, criada pela Xerox, a qual popularizou a linguagem orientada a objeto (OO).

Esse paradigma veio ganhando ao longo do tempo muita importância, nos últimos anos é a linguagem mais utilizada na construção de software, para um modelo ser considerado orientado a objeto ele deve conter alguns conceitos básicos: abstração, encapsulamento, herança e polimorfismo, que serão brevemente explicados:

A abstração é a capacidade de modelar coisas do mundo real, é a habilidade de concentrar-se nos aspectos principais e ignorar qualquer característica menos importante, e faz o que se chama de classe. Uma classe descreve um grupo de objetos com propriedades (atributos), similares comportamentos (operações), relacionamentos comuns com outros objetos e uma semântica comum. (ROCHA, 2001).

Segundo Elmasri e Navathe, “O conceito de encapsulamento é uma das principais características das linguagens e dos sistemas OO. Ele esta relacionado também com os conceitos de tipos abstratos de dados ocultarem a informação nas linguagens de programação”.

Encapsular significa que as variáveis serão acessadas por métodos definidos em sua estrutura, aumentando segurança e produtividade.

Herança é o mecanismo pela qual a linguagem de programação orientada a objetos (LPOO), fornece a possibilidade de reaproveitamento de código.

O polimorfismo é a capacidade que um objeto tem ora de se comportar de uma maneira, ora de outra.

Os conceitos da orientação a objetos já vem sido discutidos há muito tempo, vários autores e pesquisadores como Peter Coad, Edward Yourdon e Roger Pressman, abordaram a orientação a objetos como um grande avanço no desenvolvimento de sistemas.

Esses conceitos discutidos que Cood, Yourdon, e Pressman entre outros, discutiram e definiram em suas publicações foram que:

- A orientação a objetos é uma tecnologia para a produção de modelos que especifiquem o domínio do problema de um sistema.

- Quando construídos corretamente, sistemas orientados a objetos são flexíveis a mudanças, possuem estruturas bem conhecidas e provem a oportunidade de criar e implementar componentes totalmente reutilizáveis.
- Modelo orientado a objetos são implementados utilizando uma linguagem de programação orientada a objetos. A engenharia de software orientada a objeto é muito mais que utilizar mecanismos de SUS linguagem de programação, é saber utilizar da melhor forma possível todas as técnicas de modelagem orientada a objetos.
- A orientação a objetos não é só teoria, mas uma tecnologia de eficiência e qualidade comprovada usada em inúmeros projetos e para diferentes tipos de sistemas.

A orientação a objetos requer um método que integra o processo de desenvolvimento e a linguagem de modelagem com a construção de técnicas e ferramentas adequadas, esses métodos vão ser tratados no Capitulo 5.7 sobre a UML.

2.3. BANCO DE DADOS ORIENTADO A OBJETO

Hoje, o banco de dados orientado a objetos é um fator emergente que integra banco de dados e a tecnologia de orientação a objetos.

Um banco de dados orientado a objeto é um banco em que cada informação é armazenada em forma de objetos, e só pode ser manipuladas através de métodos definidos pela classe que esteja o objeto (GALANTE; MOREIRA; BRANDÃO, 2009).

Os mesmos conceitos de banco de dados OO, é o mesmo das linguagens OO, com apenas uma diferença: a persistência dos dados.

Para tanto, os bancos de dados orientados a objetos (BDOO) foram propostos para atender as limitações existentes no modelo relacional, por exigir características mais complexas, e a necessidade de armazenar novos tipos de dados, como

exemplo: imagens e vídeos, estruturas maiores, que o modelo relacional não suporta.

Outro motivo é o aumento da grande quantidade de linguagem de programação orientado a objeto, realizando o acesso direto aos dados a partir da linguagem de programação, sem exigir uma linguagem de consulta relacional como interface ao bando de dados.

Ou seja, um software desenvolvido em linguagem OO, seria necessário traduzir para a linguagem que o banco de dados relacional entenda o que torna a tarefa muito tediosa. Essa tarefa é conhecida como “Perda por Resistência”. (ELMASRI 2005).

O modelo OO oferece aumento na produtividade, segurança e facilidade de manutenção, como são objetos modulares, as mudanças podem ser feitas internamente sem afetar outras partes dos programas.

Esse modelo ganhou mais espaço nas áreas cientificas, engenharias, arquiteturas, geoprocessamento e de telecomunicações, já nas áreas comerciais não teve grandes impactos.

Em 2004, (ODMG – Object Data Management Group), com Object Query Language (OQL) padronizou uma linguagem de consulta para objetos, tornando assim o acesso mais rápido, sem a necessidade de junções, já que o acesso é realizado através de ponteiros.

2.4. MODELO OBJETO RELACIONAL

A demanda por estruturas complexas foi de fato importante para a criação dos bancos de dados orientados a objetos, evoluindo assim os SGBD’s.

Então surge o banco de dados Objeto Relacional (BDOR), que faz a união do bom desempenho do BDR, e as melhores condições de armazenamento do BDOO.

O modelo de dados Objeto-Relacional é uma extensão do modelo Relacional, as extensões incluem mecanismos para permitir aos usuários estender o banco de dados com tipos e funções específicas da aplicação.

Entretanto, o BDOO é um banco de dado que facilita a aproximação do mundo real, devida a OO e suas características (heranças, encapsulamento, abstração, polimorfismo), e que permite a manipulação de dados complexo, mais logo foram identificadas suas limitações, principalmente com desempenho comparado ao banco de dados relacional.

É através dessa vantagem que emprega um modelo que coloca a orientação a objetos em tabelas, unindo dois paradigmas em um só.

Utilizando assim os conceitos de herança, super tabelas, supertipos, reutilização de códigos, encapsulamento, controle de identidades de objetos (OID), referencia a objetos, consultas avançadas e alta proteção dos dados.

Com a criação desse novo modelo, vem à necessidade de uma linguagem padrão para manipulação dos dados em um SGBDOR. É a partir daí que surge o SQL-3, uma extensão do SQL-2 complementado com características do modelo objeto-relacional.

Apesar de ser um conceito novo, os SGBDOR tem sido uma das promessas para substituir os SGBDR, e SGBDOO, se tornando um banco de dados ideal para atender as necessidades atuais.

2.5. ETAPAS DE UM PROJETO DE BANCO DE DADOS

O objetivo do projeto de bancos de dados é gerar um conjunto de relações que permita armazenar informações sem redundância desnecessária, e também recuperar facilmente informações (SILBERSCHATZ; KORTH; SUDARSHAN, 2006).

Pode-se dizer que as fases necessárias para a construção de um Banco de Dados são quatro, conforme ilustra a Figura 15 (SILBERSCHATZ; KORTH; SUDARSHAN, 2006).

A *primeira fase* do projeto, conhecida por “especificação das necessidades do usuário”, ou “levantamento dos requisitos”, tem por finalidade caracterizar todos os dados necessários na perspectiva do usuário.

Para isso, coloca-se como primordial: levantar as necessidades dos usuários; possuir um bom entendimento da organização; conhecer a área de negócio na qual a organização está inserida; ficar atento às mudanças organizacionais, com base nas quais o banco de dados será modelado, proporcionando assim, uma maior certeza na tomada de determinadas decisões. É absolutamente essencial entender os requisitos dos usuários e adequar-se às suas necessidades a fim de obter-se um Banco de Dados bem sucedido.

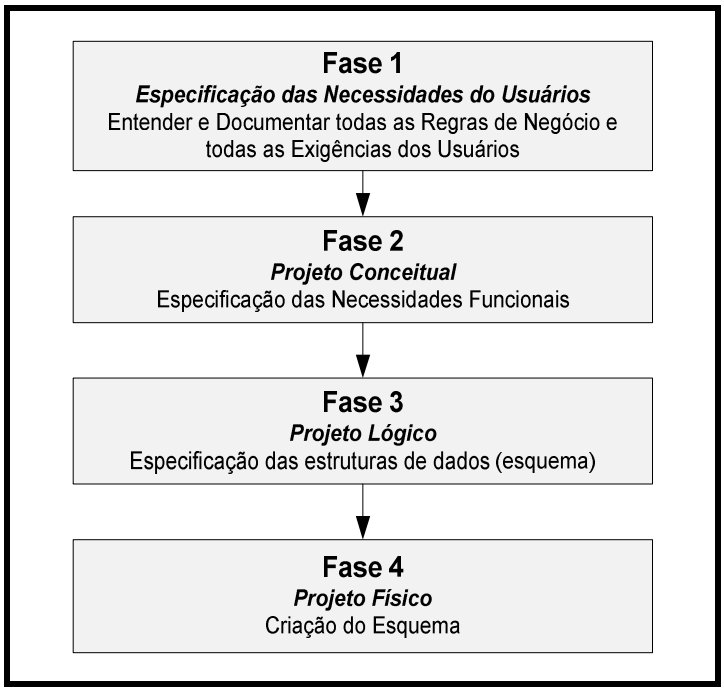

Figura 7. Fases do Projeto de um Banco de Dados

Na *segunda fase*, o projetista seleciona o tipo de modelo de dados e, por meio da aplicação de seus conceitos, transcreve as necessidades especificadas em um esquema conceitual de banco de dados, chamado “projeto conceitual”, proporcionando uma visão detalhada da organização. Nessa fase, deve-se enfocar a descrição dos dados e de seus relacionamentos, independentemente das restrições

de implementação. Os detalhes físicos de armazenamento não devem ser levados em consideração nesse momento.

A *terceira fase*, conhecida por “projeto lógico”, tem seu início a partir do modelo conceitual, levando em consideração uma das abordagens de banco de dados possíveis, tais como a **abordagem relacional** e a **abordagem orientada a objetos**. Nesta fase são descritas as estruturas que estarão contidas no banco de dados, de acordo com as possibilidades da abordagem. Contudo, não se considera, ainda, nenhuma característica específica de um sistema gerenciador de banco de dados.

A fase de projeto lógico contempla, também, as “especificações das necessidades funcionais”, nas quais os usuários descrevem os tipos de operações a serem realizadas sobre os dados.

Na *quarta fase*, a do “projeto físico”, o projeto lógico é mapeado para o modelo de implementação de dados; modelo este, especificamente dependente dos recursos do sistema gerenciador de banco de dados que será usado. Tais recursos incluem as formas de organização de arquivos e as estruturas internas de armazenamento, tais como: tamanho de atributos, índices, tipos de atributos, nomenclaturas, dentre outros.

3. FORMAS NORMAIS (NORMALIZAÇÃO DE DADOS)

Neste capítulo será abordada toda a fundamentação teórica sobre modelagem de dados aplicando os padrões das Formas Normais, essas regras vão ser utilizadas no relacionamento em IDEF1X no capitulo 6.

A modelagem de dados tem a finalidade de manipular e armazenar dados de forma segura e sempre mantendo a consistência dos dados.

Segundo Heuser (1999, Pág.131) uma forma normal é uma regra que deve ser obedecida por uma tabela para que esta seja considerada “bem projetada”. Permitindo um armazenamento consistente, colaborando significativamente para a estabilidade do projeto.

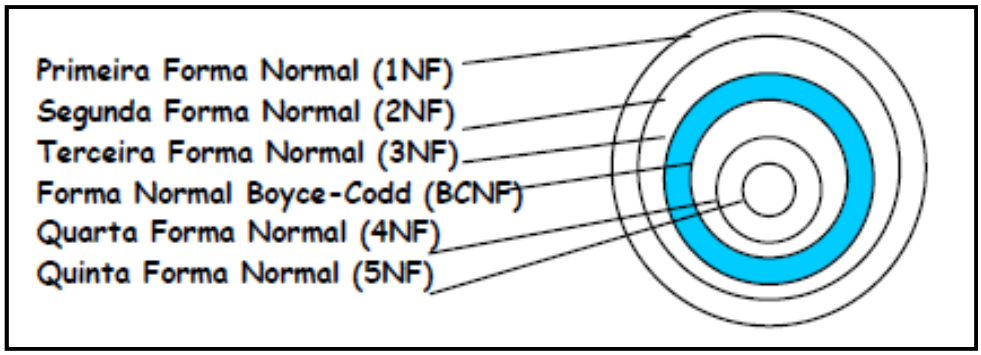

Figura 8. Fases da Normalização

As três primeiras fases (1NF, 2NF, 3NF) foram criadas por Codd (1972), grandes partes dos projetos aderem essas primeiras fases, que já mantém uma estrutura organizada.

Na fase de Boyce-Codd criada em 1974, nada mais é que uma complementação da 3FN, resolvendo assim problemas que ainda permaneciam.

Sendo que na quarta forma normal criada por Fagin (1977), em seguida a quinta forma normal criada em 1979, garantem uma estabilidade no banco de dados, reduzindo de forma significativa problemas que poderiam aparecer no projeto.

Objetivo da Normalização

- Minimização de redundância e inconsistência.
- Facilidade de manipulação do banco de dados.
- Facilidade de manutenção

3.1 Primeira Forma normal – 1FN

Essa fase determina que, se e somente se os atributos contiverem somente valores atômicos, ou seja, que não admite repetições.

Os procedimentos para identificação dessa anomalia são:

a) Identificar a chave primaria da entidade

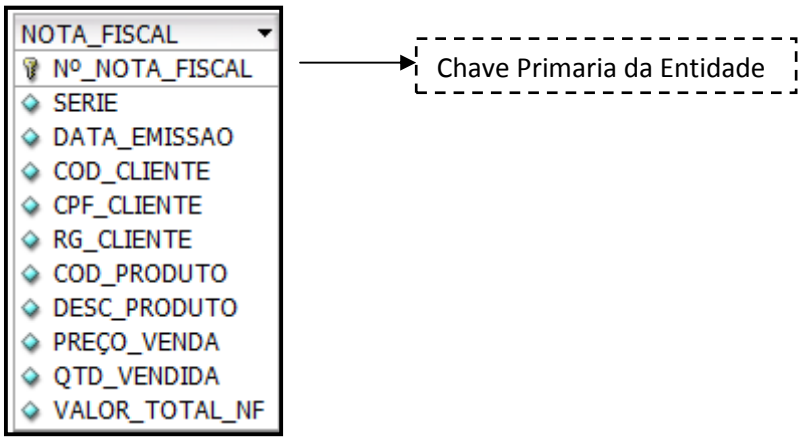

Figura 9 – Exemplo de entidade não normalizada

b) Identificar o grupo repetitivo e remove-lo da entidade

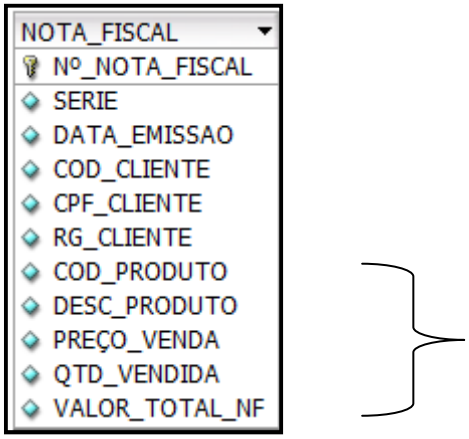

Figura 10 – Identificação de atributos repetitivos

c) Criar uma nova entidade com a chave primaria da entidade original, como mostra a Figura 3.

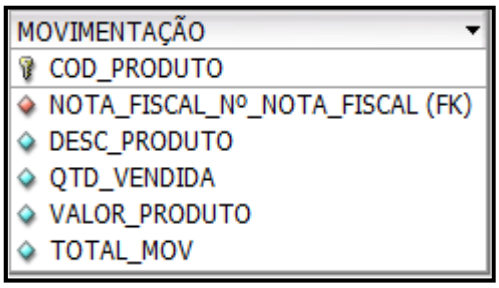

Figura 11– Exemplo de Relação Entre Tabelas

3.2 Segunda Forma Normal – 2FN

Uma tabela esta na segunda forma normal (2FN) se estiver na primeira forma normal (1FN), e seus atributos não chaves forem totalmente dependentes da chave primaria. Ou seja, devemos excluir os campos que são dependentes somente de parte da chave composta.

Os procedimentos para identificação dessa anomalia são:

A) Identificar os atributos que não são funcionalmente dependentes de toda a chave primaria.

- COD_PRODUTO
- DESC_PRODUTO
- PREÇO_VENDA

B) Remover da entidade todos esses atributos identificados e criar uma nova entidade com eles.

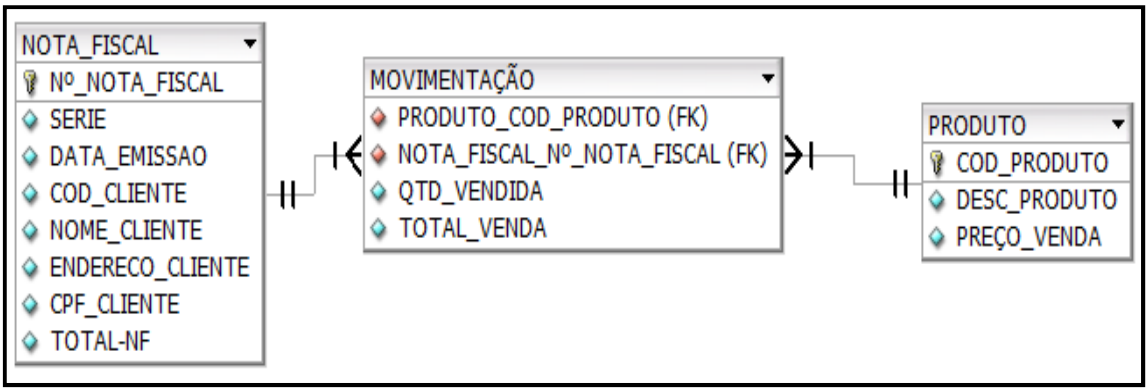

Figura 12 – Tabela na segunda Forma Normal – 2FN

Esse exemplo é uma continuação da 1FN, e em seguida na Figura 11, sendo aplicados os conceitos da segunda forma normal 2FN.

Nesse exemplo da Figura 10, a descrição do produto toda vez que for alterada, tem que mudar em todas as vendas realizadas para esse produto. Para solucionar essa anomalia é criada uma tabela de produto, pra quando for alterar a descrição, ele alterar em todas as tabelas, por causa da descrição ser dependente da chave primaria código do produto.

A segunda forma normal trata dessas anomalias, evitando redundância e inconsistência dos dados.

3.3 Terceira Forma Normal – 3FN

Uma tabela esta na terceira forma normal, se e somente se, estiver na segunda forma normal, e se nenhuma coluna não chave depender de outra coluna não chave.

Os procedimentos para identificação dessa anomalia são:

a) Precisamos remover os campos que não sejam dependentes diretamente da chave primária e sim dependente de outro campo que normalmente será a chave primária de outra tabela e chave estrangeira da tabela original.

- COD_CLIENTE
- NOME_CLIENTE
- ENDEREÇO_CLIENTE
- CPF_CLIENTE

b) Criar outra tabela que será chave estrangeira da tabela original.

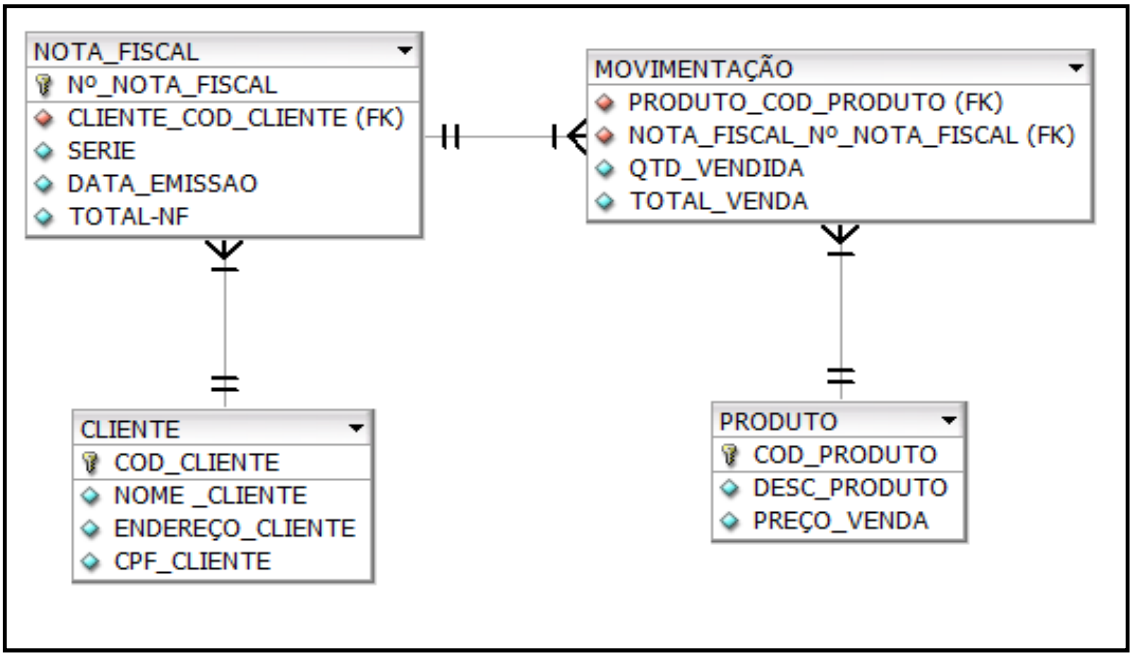

Figura 13 – Tabela na Terceira Forma Normal – 3FN.

3.4 Forma Normal de Boyce-Codd – BCNF

A forma normal de Boyce-Codd é considerada mais simples do que anterior, e foi criada para complementar a 3FN, evitando erros que poderiam ser causados. A forma normal de Boyce-Codd é aceita somente se estiver na 3FN, porem uma 3FN não esta necessariamente na BCNF.

A terceira forma normal não tratou uma anomalia onde podem ocorrer casos de uma relação ter mais de uma chave candidata.

Um exemplo de chave candidata esta na Figura 12, na tabela de cliente, em que o código que representa a chave primária, o atributo CPF que pode ser também uma chave candidata, que determina ser valor único, pelo fato de não existir outra pessoa com o mesmo numero de CPF.

3.5 Quarta Forma Normal – 4FN

Uma tabela esta na 4FN, se e somente se, estiver na 3FN e não existirem dependências multivaloradas, ou seja, pode surgir esse problema quando o atributo não chaves recebe mais de um valor (valores Múltiplos) para a mesma chave.

Os procedimentos para identificação dessa anomalia são:

a) Verificar se existem atributos não chaves multivaloradas e independentes associados ao mesmo valor da chave.

b) Desfazer os atributos não chaves multivalorados criando novas tuplas herdando a chave original.

Um exemplo de uma tabela que possuem valores multivalorados.

CÓD_CATEGORIA	COD_PRODUTO	CÓD_FORNECEDOR
100	50	1
101	55	2
100	1	3
101	12	4

Figura 14 – Tabela com atributos Multivalorados

Cada produto é então associado a uma categoria e a um fornecedor, para identificar qual fornecedor que vende produtos de uma determinada categoria. Essa tabela se encontra na terceira forma normal 3FN, mas apresentam atributos não chaves de multivalorados.

Para solucionar esse problema é aplicada a quarta forma normal 4FN, sendo assim, deve criar duas tabelas com o código da categoria em comum.

CÓD_CATEGORIA	COD_PRODUTO
100	50
101	55
100	1
101	12

CÓD_CATEGORIA	CÓD_FORNECEDOR
100	1
101	2
100	3
101	4

Figura 15 – Tabelas na quarta forma normal – 4FN

3.6 Quinta Forma Normal – 5FN

Uma tabela esta na quinta forma normal, se e somente se, estiver na quarta forma normal 4FN, e uma entidade estará na quinta forma normal 5FN, se não for possível reconstruir as informações originais a partir de registro menores, voltando a sua foram original, esse é um método raramente utilizado por programadores.

4. TECNICAS DE MODELAGEM DE DADOS

Quando um novo sistema precisa ser desenvolvido, envolve alguns fatores importantes, uma vez que é necessário levantar e analisar requisitos, identificar os objetos, entidades, e analisar como eles se relacionam.

Um dos elementos que contribuem para o sucesso da analise do software, é a utilização da modelagem de dados, que possibilita uma perspectiva diferente de quem analisa o projeto.

Quanto maior a complexidade do sistema a ser desenvolvido, maior a necessidade de boas técnicas de modelagem e profissionais capacitados, com uma boa fundamentação teórica e pratica de modelagem.

A modelagem em si é a simplificação da realidade, ou seja, um modelo que explique as características, funcionamento e o comportamento do software, facilitando assim o entendimento dos participantes do projeto.

No desenvolvimento de um software, seja ele do mais simples ao complexo, fico impossível o desenvolvimento sem a utilização de uma linguagem de modelagem. Um dos motivos, é que os softwares são sistemas dinâmicos, ou seja, nunca estão completamente finalizados, a tendência é crescer, evoluir de tamanho e complexidade, acrescentando sempre novas rotinas ou melhorando as existentes, dependendo da necessidade do usuário. Sendo assim fica evidente a necessidade de uma linguagem de modelagem, uma documentação bem detalhada e atualizada para agilizar e facilitar o projeto.

Abaixo vamos realizar uma comparação entre as técnicas de modelagem IDEF1X e UML. Esta comparação é em termo de sintaxe e símbolos de cada técnica, para descrever entidades/objetos, atributos, relacionamentos, identificadores únicos e restrições entre relacionamentos.

4.1 IDEF1X - DEFINIÇÕES DE INTEGRIDADE DE MODELAGEM DE INFORMAÇÃO

O IDEF1X é geralmente utilizado por ferramentas CASE, para desenvolver e implementar modelos de dados gráficos e lógicos de um banco de dados.

Essa linguagem e métodos para modelagem da informação é baseada no modelo entidade-relacionamento (MER), publicado como padrão norte-americano pelo NIST (Instituto Nacional de Padrões e Tecnologia, 1993).

O processo de modelagem em IDEF1X compreende várias fases importantes de uma empresa, para isso contam com 16 estruturas:

- IDEF0 – Modelo de Funções (Processos)
- IDEF1 – Modelo da Informação (Dados)
- **IDEF1X – Modelagem de Dados**
- IDEF2 – Modelo Dinâmico (Comportamento)
- IDEF3 - Descrição do processo de captura
- IDEF4 - Design orientado a objetos
- IDEF5 - Descrição Captura
- IDEF6 - Projeto Justificativa Captura
- IDEF7 - Informações Método de Auditoria do Sistema
- IDEF8 - Modelagem de Interface do Usuário
- IDEF9 - Dirigido ao cenário das Informações Sys Projeto Spec
- IDEF10 - Implementação de Modelagem de Arquitetura
- IDEF11 - Modelagem de Informações de Artefato
- IDEF12 - Modelagem organização
- IDEF13 - Três esquema de mapeamento do projeto
- IDEF14 - Projeto de Redes

A normativa IDEF1X compreende as cinco fases mais importantes na modelagem de dados, representando a semântica da informação dentro de uma organização ou de um sistema específico.

A avaliação a seguir, será baseada nessas cinco fases, representadas abaixo:

- Fase 0 – Iniciação de projeto
- Fase 1 – Definição de entidades
- Fase 2 – Definição de Relacionamento
- Fase 3 – Definição de chaves
- Fase 4 – Definição de atributos

Antes de citar essas fases, vamos esclarecer o que é modelo entidade e relacionamento.

Peter Chen introduzido pela primeira vez modelagem entidade / relacionamento (MER) em 1976, é um modelo conceitual que representa de uma forma direta as características de relacionamento dos dados. É utilizado no processo de planejamento da base de dados. Sua representação visual é dada pelo DER (Diagrama Entidade Relacionamento), que facilita a visualização da estrutura dos dados que se pretende armazenar.

Era uma brilhante idéia que revolucionou a nossa forma de representar dados. Foi uma primeira versão somente, no entanto, muitas pessoas, desde então, têm tentado melhorar. Uma verdadeira multiplicidade de técnicas de modelagem de dados que foram desenvolvidas.

Como foi citado por COUGO (1997), ele descreve o mundo como sendo, um mundo cheio de “coisas” que possuem suas características próprias e que se relacionam entre si.

Essas “coisas” definiram-se sendo entidades, logo abaixo no Item 4.1.3 é claramente explicada sua definição.

4.1.2. Inicializações do Projeto

Nessa etapa, o analista de sistema se encontra na primeira fase para começar a desenvolver o projeto que é o modelo conceitual, que prepara documentos, cronograma de modo a organizar o projeto, e fazem o levantamento de requisitos junto ao cliente.

4.1.3 Entidade

Uma entidade caracteriza-se por algo que exista, ou seja, um objeto do mundo real que necessitamos guardar informação a seu respeito, podendo ser concretos, como no caso de pessoas, objetos – ou abstrata, como eventos, conceitos, tornando assim uma tabela em nosso banco de dados.

Entidades são representadas por retângulos com cantos arredondados ou quadrados.

Os retângulos arredondados representam entidades "dependentes", aquelas cujo identificador único inclui pelo menos um relacionamento com outra entidade.

Entidades "independentes", cujos identificadores são não derivadas a partir de outras entidades, são mostradas com cantos quadrados.

4.1.4 Relacionamento

É uma estrutura que indica a associação de elementos de duas ou mais entidades, e para auxiliar nesse processo vamos utilizar os conceitos das Formas Normais, que são regras a ser seguidas para manter um banco de dados normalizado.

Em IDEF1X, as relações são assimétricas: diferentes símbolos são usados, dependendo da cardinalidade do relacionamento. Ao contrário das outras notações, os símbolos não podem ser analisados em termos de opcionalidade e cardinalidade de forma independente. Cada conjunto de símbolos descreve uma combinação de possibilidade e cardinalidade da entidade próxima a ele.

Essa relação define como são os comportamentos de um objeto, quais suas restrições, dependências e o acesso a outros objetos.

Essa estrutura conta com três tipos, relacionamento um-pra-um, um-para-muitos e muitos-para-muitos, cada um tem sua particularidade.

a) Um-para-Um, uma entidade X, esta associada apenas a uma entidade Y, e uma entidade Y este apenas associado a uma entidade X, indicando um relacionamento unívoco entre si, como mostra o exemplo.

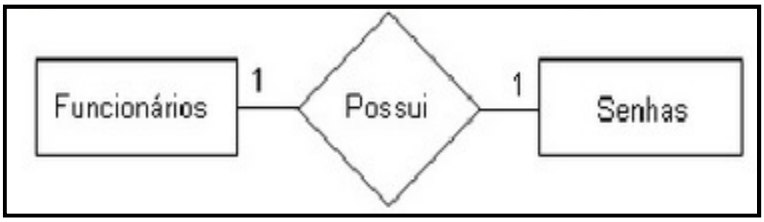

Figura 16 – Relacionamento Um – Para - Um

b) Um-para-Muitos, uma entidade X esta associada a qualquer numero de entidade Y, enquanto uma entidade Y, esta associada no Maximo a uma entidade X.

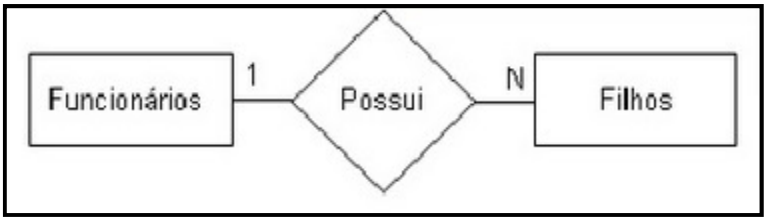

Figura 17 – Relacionamento Um – Para - Muitos

c) Muitos-Para-Muitos, Uma entidade em X esta associada a vários numero de entidade em Y, e uma entidade em Y esta associada a várias entidades em X.

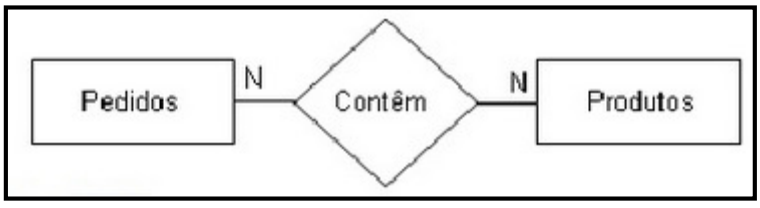

Figura 18 – Relacionamento Muitos – Para - Muitos

4.1.5 Chaves

É um conceito básico e importante na modelagem de dados para estabelecer relações entre linhas de tabelas de um banco de dados relacional. Na modelagem temos que definir as restrições de integridade através das chaves, podendo utilizar as chaves primarias, chaves estrangeiras, chaves candidatas, cada um com sua particularidade.

Chave Primaria – (PK): Segundo SILBERSCHATZ (2006), chave primaria é um conjunto de um ou mais atributos que, tomados coletivamente, nos permitem identificar de maneira exclusiva, de maneira única, uma entidade em um conjunto de entidades.

Dessa maneira ela oferece segurança, fazendo com que não tenha cadastro de dados repetidos e essa chave não permite valor nulo. A chave primaria é um importante objeto quando se aplica as regras de normalização.

Chave Estrangeira – FK: Conforme HEUSER (1999), uma chave estrangeira é uma coluna ou uma combinação de colunas, cujos valores aparecem necessariamente na chave primaria de uma tabela. Ou seja, este tipo de chave é utilizado para criar o relacionamento entre tabelas, quando um atributo de outra tabela é chave primaria, e precisa se relacionar com outra tabela, se torna chave estrangeira.

Chave Candidatas / Alternada: Essa chave é apenas conceitual, ou seja, ela não é implementada. Ela existe quando mais de um atributo possuem propriedade de identificação única.

Os atributos com essa característica poderiam ser primarias já que possuem por natureza a identificação única, como por exemplo: RA, CPF, Titulo Eleitor, RG, entre outros.

4.1.6 Atributos

Os atributos são propriedade que definem uma entidade, como por exemplo, os atributos de uma entidade produto, é seu código, descrição, valor unitário, marca peso, etc..

Os atributos podem ser simples, composto, multivalorado ou determinante, cada um com sua particularidade, os mais comuns são os atributos simples.

Atributo Simples ou Mono valorado: Que recebe um único valor e não é um atributo chave.

Atributo Composto: Seu conteúdo é dividido em vários atributos, um exemplo fácil e simples de entender, é o endereço, que pode ser divido em rua, endereço, numero, ou um nome de uma pessoa, pode ser dividido em nome e sobrenome.

Atributo Multivalorado: É tratado na quarta forma normal, um exemplo é uma pessoa, às vezes a mesma pessoa possui mais de um numero de telefone.

Atributo Determinante: será chave primaria em uma entidade, um exemplo é o CNPJ de um fornecedor, ou o CPF de um cliente, que identifica de forma única cada um deles, ou seja, não pode haver outra pessoa ou fornecedor.

4.2. UML

A UML (Unified Modeling Language – linguagem de Modelagem Unificada) é uma linguagem visual ou notação de diagramas utilizada para modelar softwares baseados no paradigma de orientação a objeto.

O grande problema na época no desenvolvimento de software utilizando a orientação a objeto é que não existia uma notação padronizada. Cada metodologia existente na época possuía seus próprios conceitos, gráficos e terminologias, resultando uma grande confusão entre os desenvolvedores.

Diante dessa diversidade de conceitos, Booch, Rumbaugh e Jacobson, que são conhecidos como “os três amigos” decidiram criar a UML, quando foi lançada, muitos desenvolvedores comemoraram, pois ela uniu as melhores idéias de metodologias existentes na época.

Tornou-se nos últimos anos, a linguagem padrão de modelagem adotada internacionalmente pela indústria de engenharia de software, é controlada pela Object Management Group (OMG), é a norma da indústria para descrever graficamente o software.

Mas deve-se lembrar que a UML, é uma linguagem de modelagem independente, não esta vinculada a nenhuma linguagem de programação e a nenhum processo de desenvolvimento.

A sua utilização é para auxiliar os desenvolvedores a definirem as características do sistema, como seus requisitos, comportamento, sua estrutura lógica, dinâmica de seus processos, e ate mesmo na questão de equipamentos sobre o qual o sistema devera ser implantado.

De acordo com “os três amigos”, há quatro objetivos principais para se criar modelos:

1. Eles ajudam a visualizar o sistema como ele é ou como desejamos que fosse.
2. Eles permitem especificar a estrutura ou comportamento de um sistema.

3. Eles proporcionam um guia para a construção do sistema.
4. Eles documentam as decisões tomadas no projeto.

A UML é composta por muitos elementos de modelo que representam as diferentes partes de um sistema. Os elementos são usados para criar diagramas, que representam uma determinada parte, ou um ponto de vista do sistema, ela aborda o caráter Estrutural: Estática e Comportamental: Dinâmica, presente em todo software.

A modelagem Estrutural: Estática, utiliza os diagramas:

- **Diagrama de Classe**: Mostra Classes, interfaces, colaboração e seus relacionamentos.
- **Diagrama de Objetos**: Mostra um conjunto de objetos e seus relacionamentos. Ilustram as estruturas de dados, registros estáticos de instancia encontrados nos diagramas de classes.
- **Diagrama de Componentes**: Mostra os componentes de programação de alto nível.
- **Diagrama de Implantação**: Mostra um conjunto de nós e seus relacionamentos, esse diagrama esta relacionado com o diagrama de componentes, possui um nó contem um ou mais componentes.

A modelagem Comportamental: Dinâmica, utiliza os diagramas:

- **Diagrama de Caso de Uso**: Mostra autores (pessoas ou outros usuário do sistema), caso de uso (os cenários onde eles usam o sistema), e seus relacionamentos.
- **Diagrama de Seqüência**: Um diagrama de seqüência é um diagrama de interação que da ênfase a ordenação temporal das mensagens.
- **Diagrama de Atividades**: Mostra atividades e mudanças de uma atividade para outra com os eventos ocorridos em alguma parte do sistema.

- **Diagrama de Estados**: Mostra estados, mudanças de estado e eventos num objeto ou parte do sistema.
- **Diagrama de Colaboração**: Mostra objetos e seus relacionamentos, colocando ênfase nos objetos que participam na troca de mensagens.

Alguns exemplos de modelos de elementos são as classes, objetos, estados, pacotes e componentes. Os relacionamentos também são modelos de elementos, e são usados para conectar outros modelos de elementos entre si.

Cada diagrama captura uma perspectiva diferente e um determinado elemento poderá existir.

Entre esses diagramas, vamos trabalhar em particular o diagrama de classe, que faz a união de tudo que estudamos ate agora.

4.2.1 Diagrama de Classe

Os diagramas de classes exibem um conjunto de classes, interfaces e colaborações, bem como seus relacionamentos. Esse diagrama é encontrado com maior freqüência em sistemas de modelagem orientados a objeto e abrangem uma visão estática da estrutura dos sistemas.

Um diagrama de classe pode apresentar três perspectivas diferentes, cada um para um tipo de observador diferente. São elas:

- Conceitual:
    - Representa os conceitos do domínio em estudo.
    - Perspectiva destinada ao cliente.
- Especificação:

- Tem Foco nas principais interfaces da arquitetura, nos principais métodos, e não como eles iram ser implementados.
- Perspectiva destinadas as pessoas que não precisam saber detalhes de desenvolvimento, tais como gerentes de projeto.

- Implementação: (A mais utilizada de todas)
  - Aborda vários detalhes de implementação, tais como navegabilidade, tipo dos atributos, etc.
  - Perspectiva destinada ao time de desenvolvimento.

4.2.2 Classe

As classes são os blocos de construção mais importantes de qualquer sistema orientado a objeto. Uma classe é uma descrição de um conjunto de objetos que compartilham os mesmo atributos, operações, relacionamentos e semântica.

As classes são representadas graficamente por retângulos, dividido em três partes como mostra a figura 19, o primeiro contendo o nome da classe, o segundo que são os atributos, e a terceira são as operações, ou seja, os métodos de manipulação de dados.

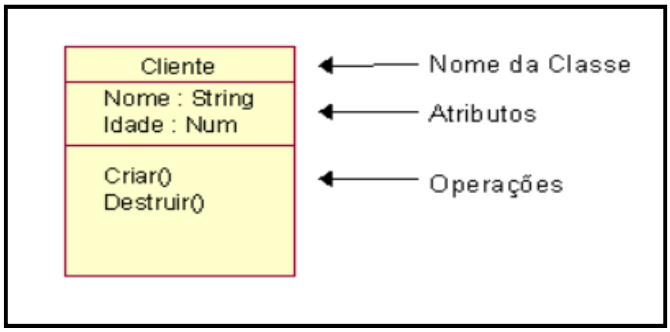

Figura 19 – Representação Visual de uma Classe em UML

Existem duas representações gráficas para as classes, uma é a classe concreta como ilustra a Figura 19, que são aquelas onde todos os métodos são implementados e instanciados os objetos em nosso Código.

A outra é classe abstrata, é uma representação de algum conceito, a única diferença entre elas é o estilo da fonte do nome da classe que muda para itálico.

4.2.3 Atributos

Um atributo representa alguma propriedade do item que esta sendo modelada, uma classe pode ter qualquer numero de atributos, ou ate mesmo nenhum.

Geralmente o nome de um atributo é um substantivo, ou uma expressão que representa alguma propriedade da classe correspondente, é típico aparecer com o primeiro caractere com a letra maiúsculo,

Na UML, atributos são mostrados com pelo menos seu nome, e podem também mostrar o seu tipo, valor inicial e outras propriedades. Os Atributos podem também ser exibidos com sua visibilidade, conforme a Figura 20:

+ Indica atributos *Públicos* (Significa que ele pode ser acessado de qualquer lugar da classe ou sub-classe)

Indica atributos *Protegidos* (Significa que ele pode ser acessado de dentro da própria classe, ou em suas classes-filha)

- Indica atributos *Privados* (Significa que ele só pode ser acessado de dentro da própria classe)

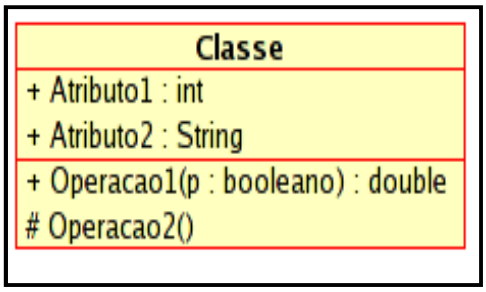

Figura 20 – Atributos e Operações

4.2.4 Operações

Uma operação (métodos) é a implementação de um serviço que pode ser solicitado por algum objeto da classe para modificar o comportamento. Uma classe pode ter qualquer número de operações ou ate mesmo não ter operações. O nome de uma operação pode ser um texto, como os nomes das classes. Na pratica seu nome é um verbo, e geralmente aparece com a primeira letra em maiúsculo. As operações como os atributos mostrar sua visibilidade, conforme a Figura 20.

4.2.5 Relacionamentos

Os relacionamentos ligam as classes/objetos entre si, criando relações lógicas entre as entidades existentes. Na modelagem orientada a objetos, existem três tipos de relacionamentos: Associações, Generalizações e Dependência.

Cada um desses relacionamentos fornece uma forma diferente de combinações de abstrações, esses relacionamentos são representados graficamente como um caminho, com tipos diferentes de linhas para diferenciar os tipos de relacionamentos.

- Associação: É um relacionamento estrutural que especifica objetos de um item conectados a objetos de outro item. A partir dessa associação conectando duas classes, você é capaz de navegar do objeto de uma classe ate o objeto de outra classe e vice-versa. Não muito comum você pode ter associações conectando mais de duas classes, que chamamos associações enésimas.

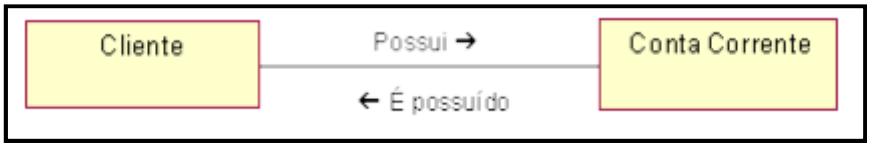

Figura 21 – Exemplo de Associação

- Generalização: Esses relacionamentos entre itens gerais chamaram de superclasses ou classes-mãe, e tipos mais específicos desses itens é chamados de subclasses ou classes-filha. A generalização significa que o objeto da classe-filha pode ser usado em qualquer local em que a classe-mãe ocorra, mais não vice-versa.

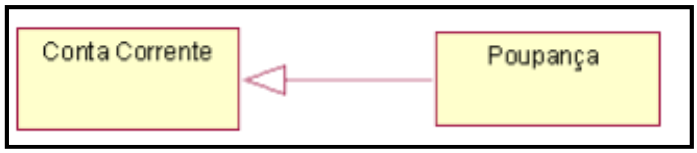

Figura 22 – Exemplo de Generalização

- Dependência: Nesse relacionamento de utilização, determina que uma classe, usa as informações e serviços de outra classe, mais não ao contrario, ou seja, usa-se sempre que quiser indicar algum item dependendo de outro.

Figura 23 – Exemplo de Dependência

4.2.6 Multiplicidade

A multiplicidade de uma classe é o numero de instancias possível que uma classe pode ter em uma única instancia da outra classe a qual ela está associada.

- 0...1 (zero ou Um) – Indica que apenas uma instancia de classe se relaciona com as instâncias da outra classe.

- 1 (Um) – Indica que apenas um objeto da classe se relaciona com os objetos da outra classe.

- 0...* (Zero ou muitos) - Não existe um limite para o numero de instancias.

- 1...* - Indica que há pelo menos um objeto envolvido no relacionamento

5. PROPOSTA DO TRABALHO

A proposta desse projeto foi a de desenvolver os estudos das técnicas de modelagem de banco de dados, destacando as diferenças. Cada técnica tem suas vantagens e desvantagens, mas apresentam a mesma tendência: se esforçam para auxiliar os desenvolvedores em todos os processos dos projetos.

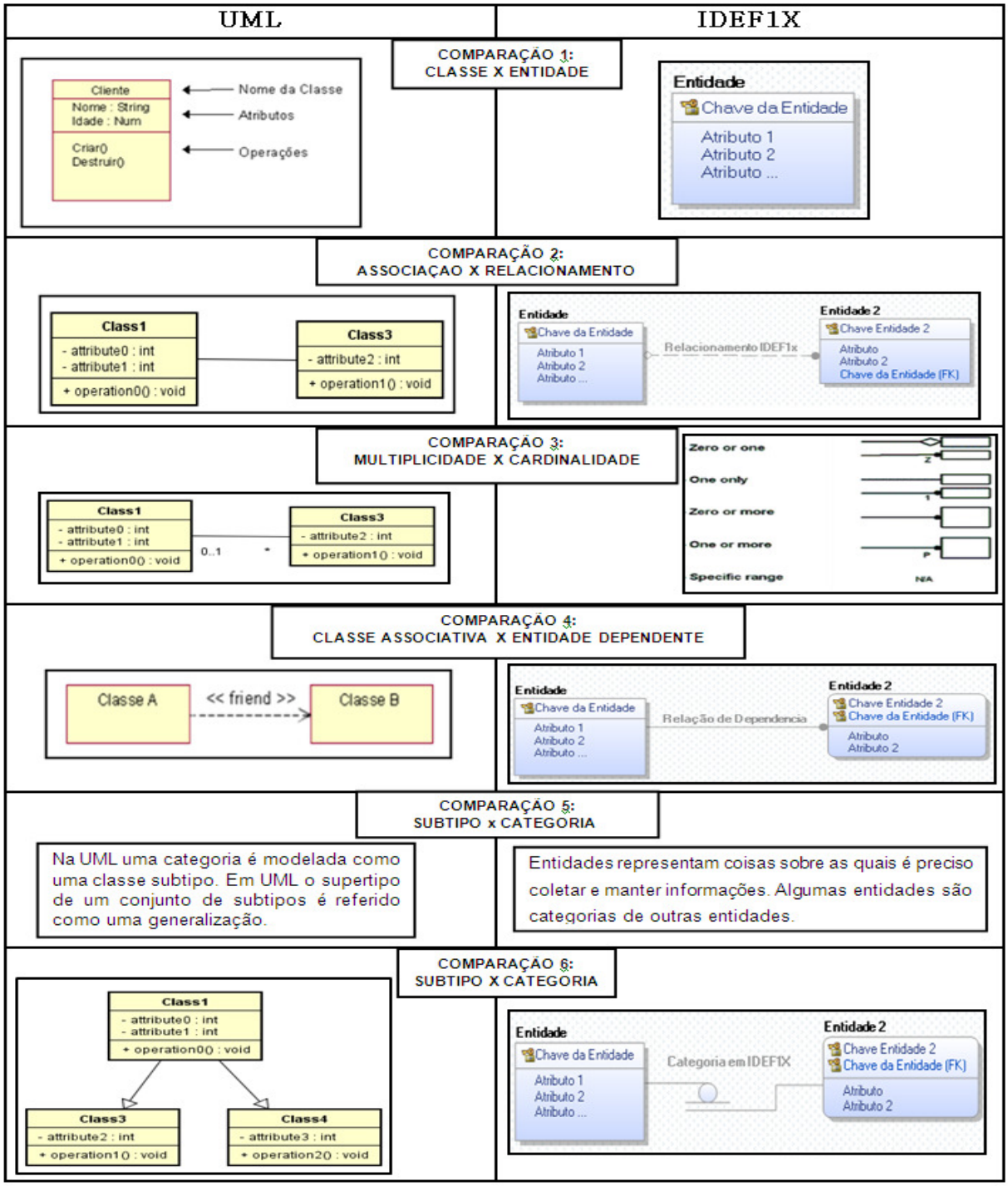

Figura 24 – Proposta do Trabalho

5.1 Comparação 1: Classe x Entidade

Na UML diagrama de classe é usado para modelar os requisitos de dados. Uma classe UML é modelada como uma caixa e duas linhas horizontais sólidas são usadas para separar a caixa em três secções. A seção superior contém o nome da classe, a seção do meio contém os atributos da classe e a seção inferior contém os métodos ou comportamento suportados pela classe.

A principal diferença entre uma classe UML e uma entidade IDEF1x é a presença de métodos. Um método representa um serviço que pode ser solicitado para uma classe de afetando o comportamento (Isto é, executa operações sobre os seus atributos). Uma entidade IDEF1X pode ser pensado como uma classe UML sem operações.

Entidade:

Representa um conjunto de “coisas”, como pessoas, lugares, abstrações, ou coisas que são relevantes para uma empresa ou pessoa. Essas coisas compartilham um conjunto de características pelas quais eles podem ser descritos de forma exclusivas. Um membro individual do conjunto é referido como uma instancia de entidade.

Em IDEF1X uma entidade é modelada como uma caixa e uma única linha horizontal sólida são usadas para separar a caixa em duas secções. A seção de tipo contém um ou mais atributos que compreendem o identificador exclusivo (Chave Primaria) para a entidade.

A seção inferior contém os atributos que não sejam primários, que caracterizam a entidade. A cada entidade é atribuído um nome único, que é colocada por cima da caixa.

IDEF1X é uma linguagem de modelagem baseada no modelo entidade-Relacional (MER) e, como tal, utiliza o conceito de uma chave primaria, que é emprestado do modelo relacional, para identificar instancias de uma entidade.

A chave primária de uma entidade é definida como um ou mais atributos, cujo valor identifica unicamente cada instancia da entidade. Por exemplo, a entidade pessoa pode ter os atributos; pessoa-identificador, nome e endereço, onde pessoa-identificador é a chave primaria.

5.2 Comparação 2: Associação x Relacionamento

Na UML um relacionamento é chamado de uma associação. Uma associação é modelada usando uma linha sólida que liga duas classes. A UML utiliza identificadores de objeto (OID) para identificar instancias de uma classe (objetos) e não suporta os conceitos de heranças, chaves primárias e chaves estrangeiras.

Relacionamento:

Em IDEF1X uma relação é modelada como uma linha desenhada entre as entidades por meio de um círculo escuro numa extremidade da linha. A entidade locada pelo circulo escuro é chamada entidade criança e a entidade na outra extremidade da linha é chamada entidade-mãe.

As linhas de relacionamento têm um aspecto diferente baseada na idéia de herança chave primaria.

Herança é definida da seguinte forma: Se existe uma relação entre duas entidades, os atributos que formam a chave primária da entidade-mãe são herdados como atributos da entidade filho. Estes atributos herdados são conhecidos como chaves estrangeiras.

Se um exemplo de uma entidade de criança é unicamente identificado pela associação com a entidade-mãe, em seguida, a relação é dita ser uma relação de identificação, e alinha de relação é modelado como uma linha sólida.

Se todos os casos, de uma entidade de criança podem ser identificados exclusivamente sem conhecer a instancia associada da entidade-mãe, em seguida, a relação é dita ser uma relação não-identificação, e a linha de relação é modelado como uma linha tracejada.

5.3 Comparação 3: Multiplicidade x Cardinalidade

A cardinalidade em UML é chamada Multiplicidade, refere-se ao numero de objetos em uma classe que pode estar relacionado a um objeto na classe relacionada.

Na multiplicidade UML é modelado como segue:

- Uma gama, como 1..12, ou
- Um único valor, como 4 ou
- Um intervalo, como 1,4,8,32, ou
- Uma combinação de uma faixa e um intervalo, como 1..5,9,14

Na multiplicidade UML pode ser usado em ambos os lados de uma linha de associação. Um asterisco (*) é utilizado para denotar uma multiplicidade de “muitos”, Os símbolos * e 0 por si só, são lidas como “zero, um ou mais”.

Uma linha de associação sem qualquer multiplicidade é lido como “muitos para muitos” (note que este difere de uma linha de relacionamento em IDEF1X sem quaisquer cardinalidade, que é lido como “um para um”).

Cardinalidade:

Em um modelo de dados IDEF1X relações entre as entidades estão limitados pela adoro em cada extremidade da linha de relacionamento. Estas restrições são chamadas cardinalidade.

Cardinalidade representa o numero máximo de vezes que uma instancia de entidade, podem ser relacionados com o conjunto de instancias em outra entidade. Em IDEF1X cardinalidade é modelado como se segue:

•Cada Instancia entidade-mãe pode ter zero; uma ou mais instancias de entidade filho associados, ou

•Cada instancia de entidade-mãe deve ter pelo menos um ou instancias de entidade filho mais associados, ou

•Cada instancia de entidade pai pode ter nenhum ou no máximo uma instancia de entidade filho mais associados, ou

Cada instancia entidade-mãe está associado com algum numero exato de instancia de entidade criança.

5.4 Comparação 4: Classe Associativa x Entidade Dependente

Na UML uma entidade dependente é modelada com uma classe de associação.

A classe associativa é modelada como uma classe (caixa) ligada a uma linha de associação (e que é uma linha sólida), com uma linha tracejada.

Entidade Dependente:

Em IDEF1X a entidade filho em um relacionamento é dito ser dependente de existência da entidade-mãe.

Em que uma instância da entidade filho só pode existir se a instância associada da entidade-mãe existir. Por exemplo, em uma relação entre entidades cliente e política, um cliente é segurado para zero, um ou mais políticas e cada política deve garantir exatamente uma pessoa.

Em IDEF1X uma entidade dependente é modelada como uma caixa com cantos arredondados.

5.5 Comparação 5: N-Ary Associação x Entidade Dependente:

Uma associação N-ária é uma associação entre três ou mais classe. Na UML uma associação N-ário é modelada conectando cada classe que participa na associação N-ário usando uma linha de associação (que é uma linha sólida).

Entidade Dependente:

Em IDEF1X uma relação N-ário é modelada usando uma entidade dependente em que a entidade dependente é dependente de todas as entidades que participam do relacionamento.

5.6 Comparação 6: Subtipo x Categoria

Na UML uma categoria é modelada como uma classe subtipo. Em ambas UML e IDEF1X o supertipo de um conjunto de subtipos é referido como uma generalização e as construções utilizadas para representar uma relação subtipo/ supertipo é referido como uma hierarquia de generalização. A criança ou subtipo na hierarquia herda todas as características de um dos pais ou supertipo. Na UML uma associação subtipo / supertipo é modelada conectando cada classe subtipo ao supertipo classe com uma ponta de seta bloco.

CATEGORIA:

Entidades representam coisas sobre as quais é preciso coletar e manter informações. Algumas entidades são categorias de outras entidades.

Por exemplo, um banco recolhe e mantém as informações sobre contas de clientes. Algumas das informações coletadas sobre contas são aplicável tanto a verificação e contas poupança, por exemplo, nome e endereço do proprietário.

No entanto, algumas informações são únicas para qualquer conta corrente ou poupança. Portanto, as entidades de poupança de conta, e verificar ing-conta são categorias da entidade conta. Em IDEF1X essas entidades são modeladas usando um relacionamento categorização, que também é chamado um subtipo, ou relacionamento. Em IDEF1X uma categoria é modelada utilizando um circulo na linha que liga o supertipo aos seus subtipos.

O nome do atributo entidade genérico usado como o discriminador é escrito ao lado do circulo. Uma ou duas linhas horizontais sólidos são desenhados sob o circulo. Uma única linha significa que pode haver mais subtipos, para esta hierarquia supertipo, do que as mostradas no modelo de dados. Duas Linhas significam que a hierarquia supertipo é completa e não existem outros subtipos do que as mostradas no modelo de dados.

Cada técnica tem pontos fortes e fracos na forma como aborda cada público. Como isso acontece, a maioria é muito mais orientada para os designers do que são para o usuário. Estes produzem modelos que são muito complexo e se concentrar em fazer que todas as restrições possíveis são descritas. Infelizmente, esta é muitas vezes à custa de legibilidade.

Entre os mais importantes dos princípios do desenho gráfico é que cada símbolo deve ter apenas um significado, que se aplica onde quer que esse símbolo é usado, e que cada conceito deve ser representada por apenas um símbolo.

IDEF1X é uma boa ferramenta para a base do projeto de banco de dados, mas não segue as regras de bom design gráfico, conforme a Figura abaixo.

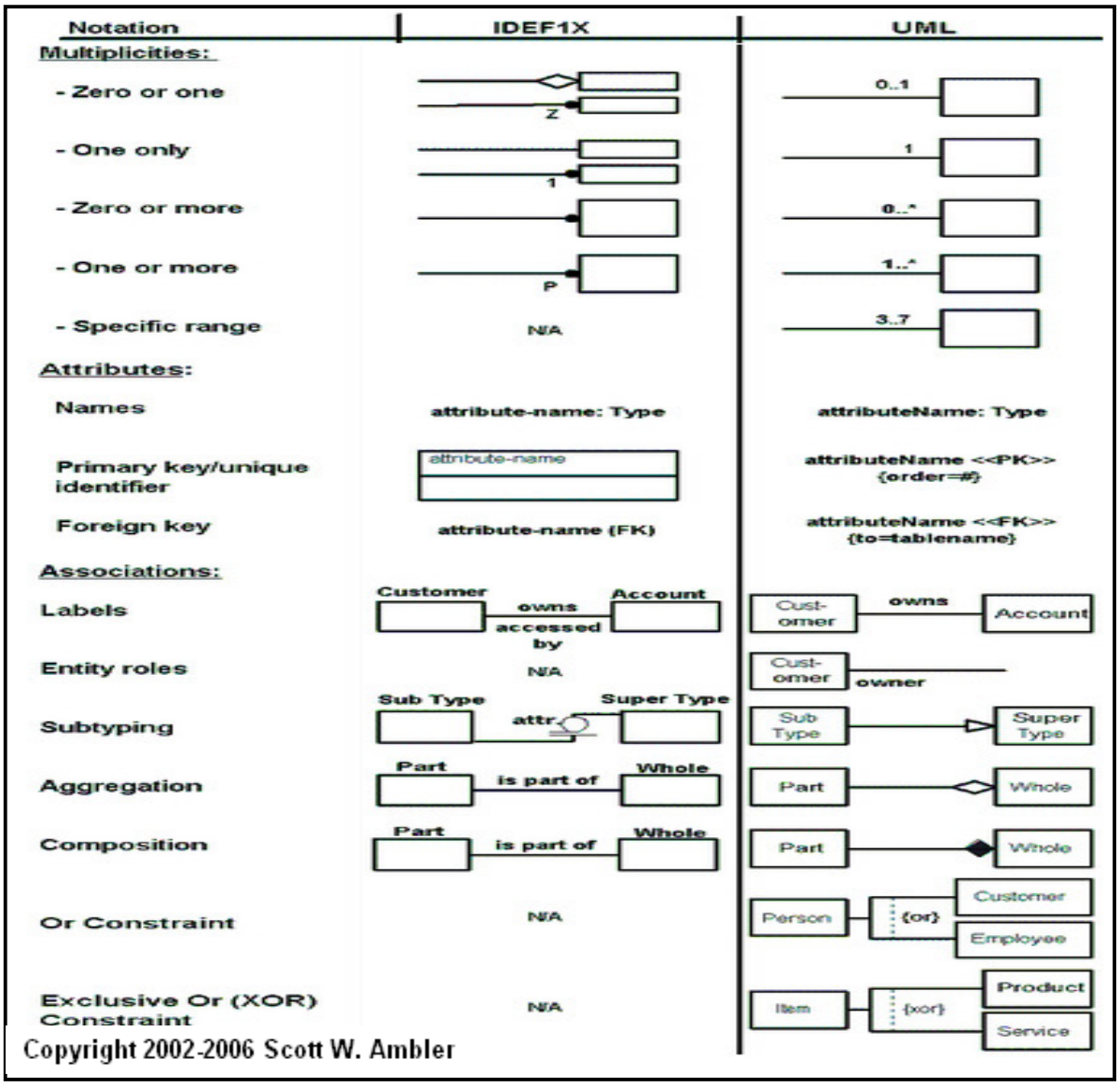

Figura 25 – Notações IDEF1X & UML

Um conceito que deve ser representada por um único símbolo requer vários juntos e requer símbolos diferentes sob circunstâncias diferentes. Ou seja, situações particulares podem ser representadas por mais do que um conjunto de símbolos, sendo que o mesmo símbolo pode significar coisas diferentes, dependendo do contexto. O símbolo é usado para descrever uma determinada situação é fortemente dependente do contexto dessa situação e de como o relacionamento vai ser implementado, e não apenas na própria situação.

Por exemplo, o ponto sólido pode significar qualquer coisa, dependendo do contexto, e a mesma situação particular pode ser representada por mais do que um conjunto de símbolos. Além disso, uma correção, um erro em um relacionamento no diagrama IDEF1X geralmente desencadeia mudanças em cascata.

O diagrama IDEF1X, modela os mesmos elementos e relacionamentos vistos em UML, o IDEF1X também representa informações que não aparecem no DER, como o tipo de dado dos elementos simples (varchar, int...) e em respeito ao uso do elemento se ele é obrigatório ou não (null, not null).

Diante do estudo realizado, em livros e internet, que mesmo a normativa IDEF1X ser mais antiga, sua aplicação é mais difícil do que a UML, considerada nova perto da normativa.

6. ESTUDO DE CASO

Para fins de comparação vamos modelar nosso exemplo, vamos fazer o uso da ferramenta Astha, para modelar em UML, e para IDEF1X vamos utilizar a ferramenta ERwin.

Para o desenvolvimento do nosso diagrama de classe, precisaremos de um cenario qualquer onde realizaremos o passo a passo até identificar todas as classes, a partir dessa etapa poderemos efetuar as ligações e cardinalidade.

CENÁRIO:

Uma determinada faculdade deseja informatizar seu sistema de matrículas, abaixo segue a analise de requisitos:

- A universidade oferece varios cursos.
- O Coordenador de um curso define as disciplinas que serão oferecidas
- Várias disciplinas são oferecidas em um curso.
- Varias turmas podem ser abertas para uma mesmo disciplina, pórem o numero de estudantes inscritos deve ser entre 3 e 10.
- Estudantes selecionam 4 disciplinas.
- Quando um estudante matricula-se para um semestre, o Sistema de Registro Academico é notificado.
- Professores usam o sistema para obter a lista dos alunos matriculados em suas disciplinas. O Coordenador tambem.

Dessa forma vamos identificar as classes do nosso cenario:

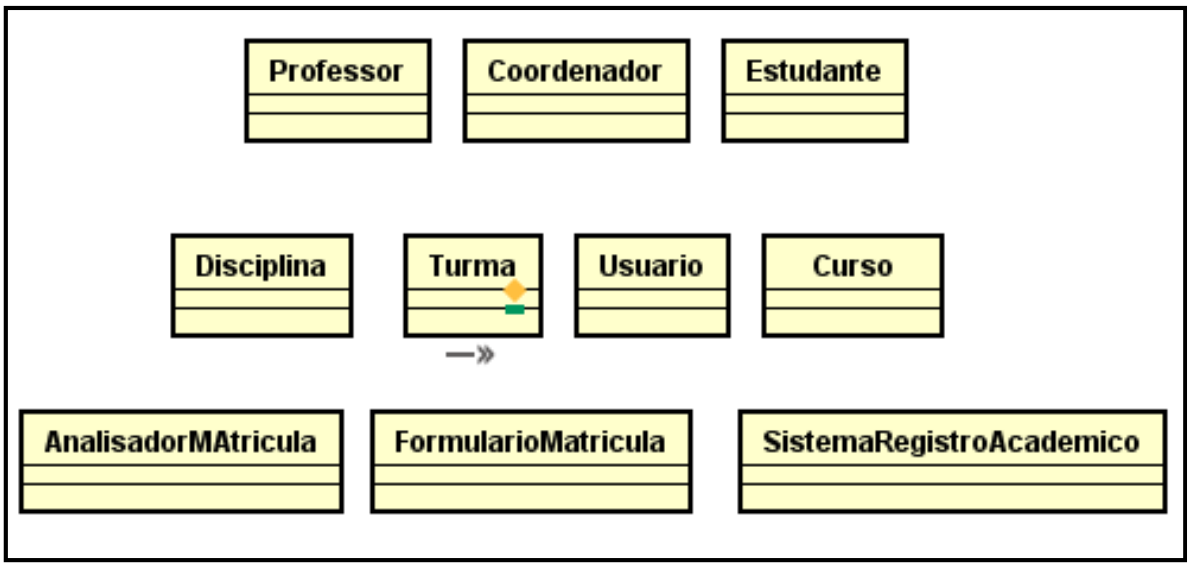

Figura 26 – Identificação das Classes – UML

Analisando a Figura 26 e 27, encontramos uma diferença visual na representação das entidades / classe, conforme explica o quadro do capitulo 5, em sua primeira comparação.

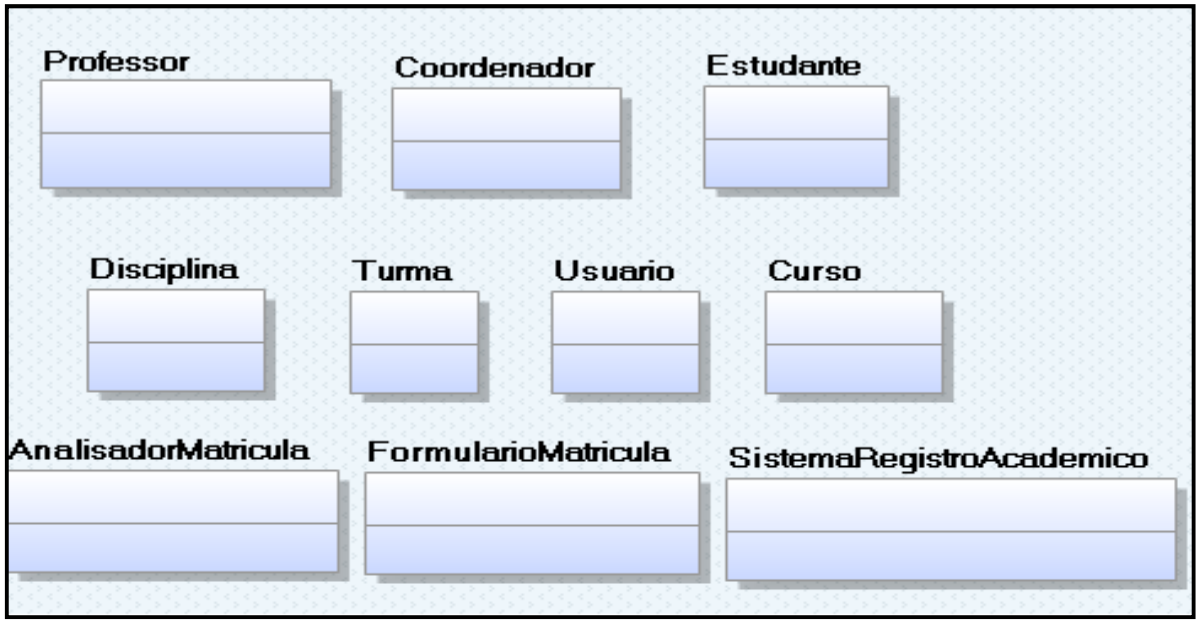

Figura 27 – Identificação das Entidades – IDEF1X

Seguindo os requisitos estabelecidos identificamos os principais relacionamentos, o diagrama de classe fica da seguinte forma:

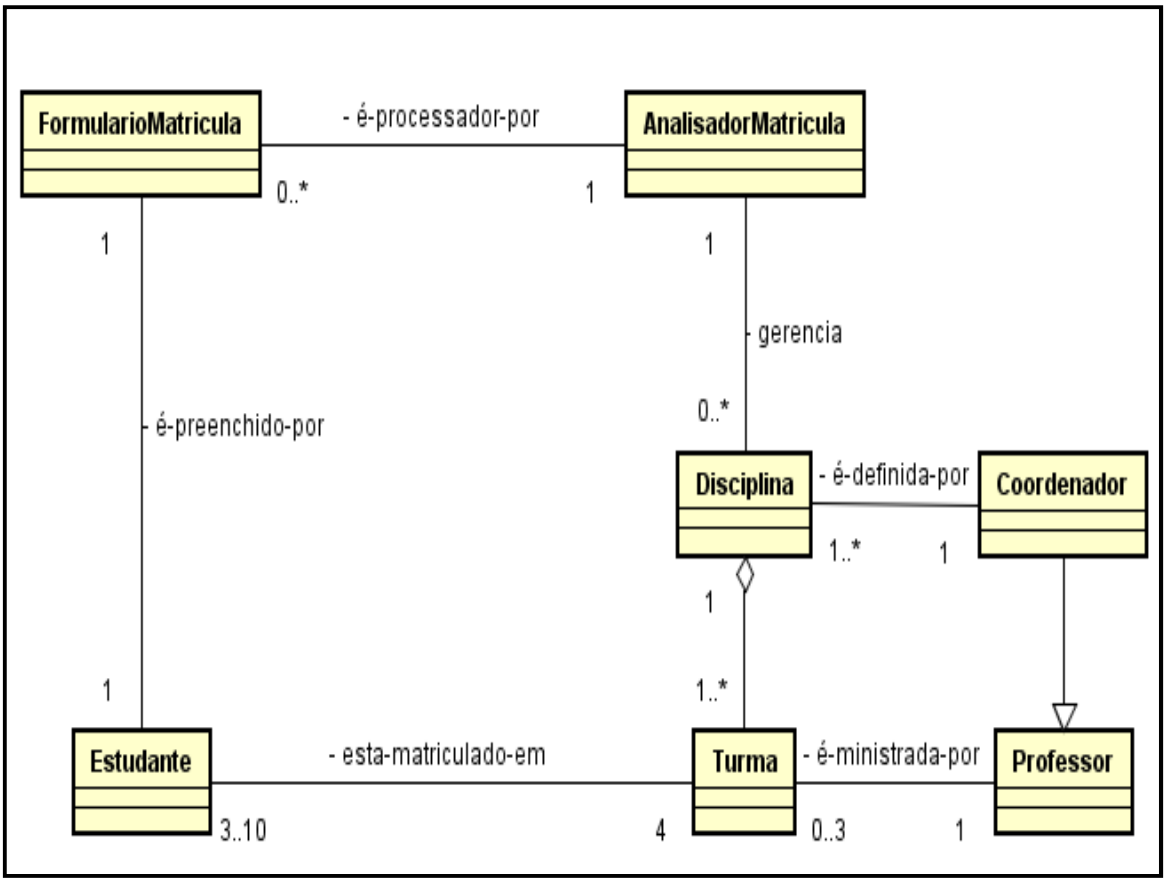

Figura 28 – Principais Relacionamentos - UML

Para complementar nosso diagrama de classe, vamos adicionar os atributos de cada classe, esses atributos pode mudar ate o final do diagama, ficando da seguinte forma nosso diagrama:

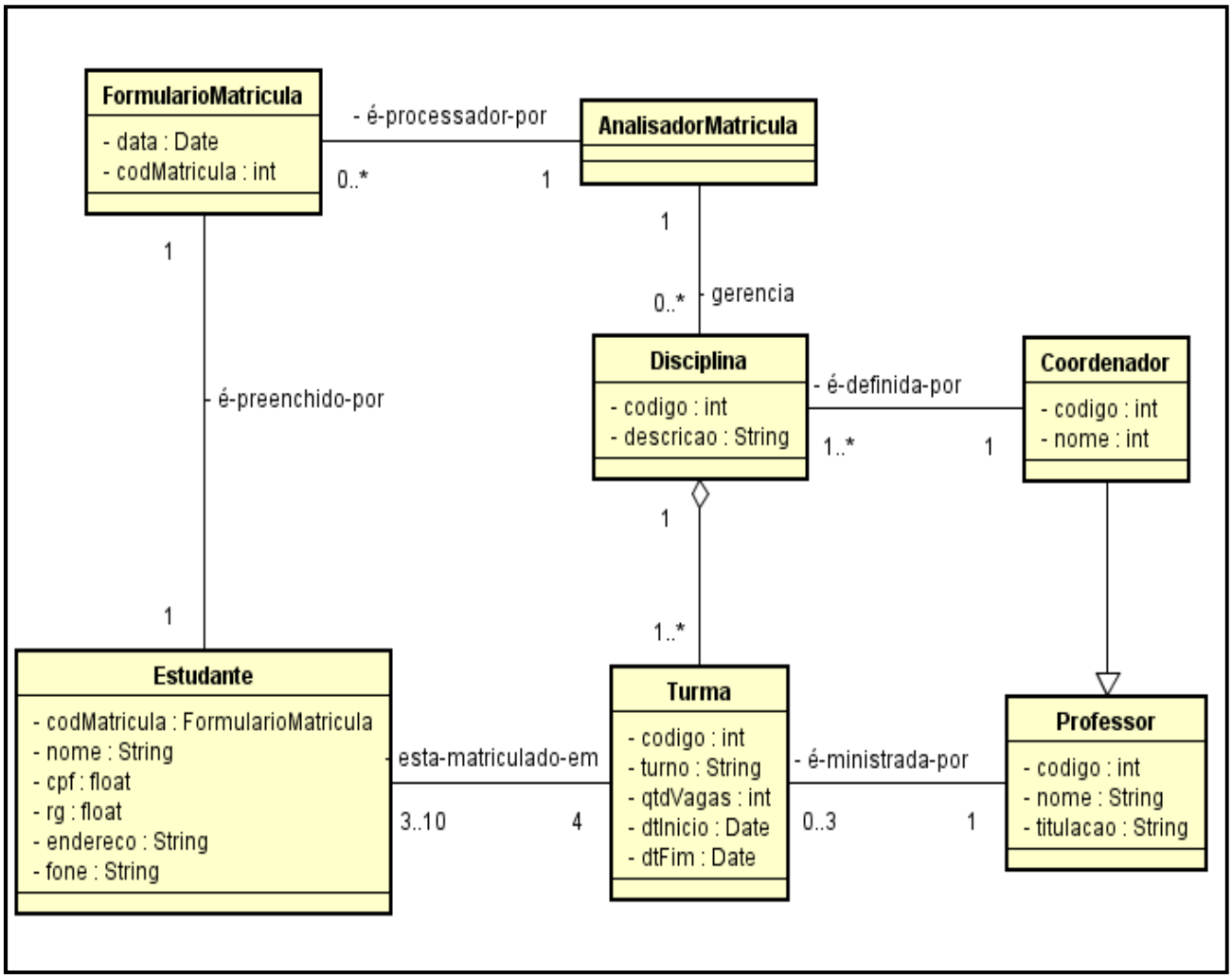

Figura 29 – Diagrama com os Atributos - UML

Para dar continuidade vamos relembrar alguns aspectos importante do diagrama de classe. Cada classe é representado por um retangulo dividido em tres partes, a primeira com o nome, a segunda com os atributos e a terceira são os metodos da classes. O proximo passo é adicionar esses metodos, nosso diagrama fica da seguinte forma:

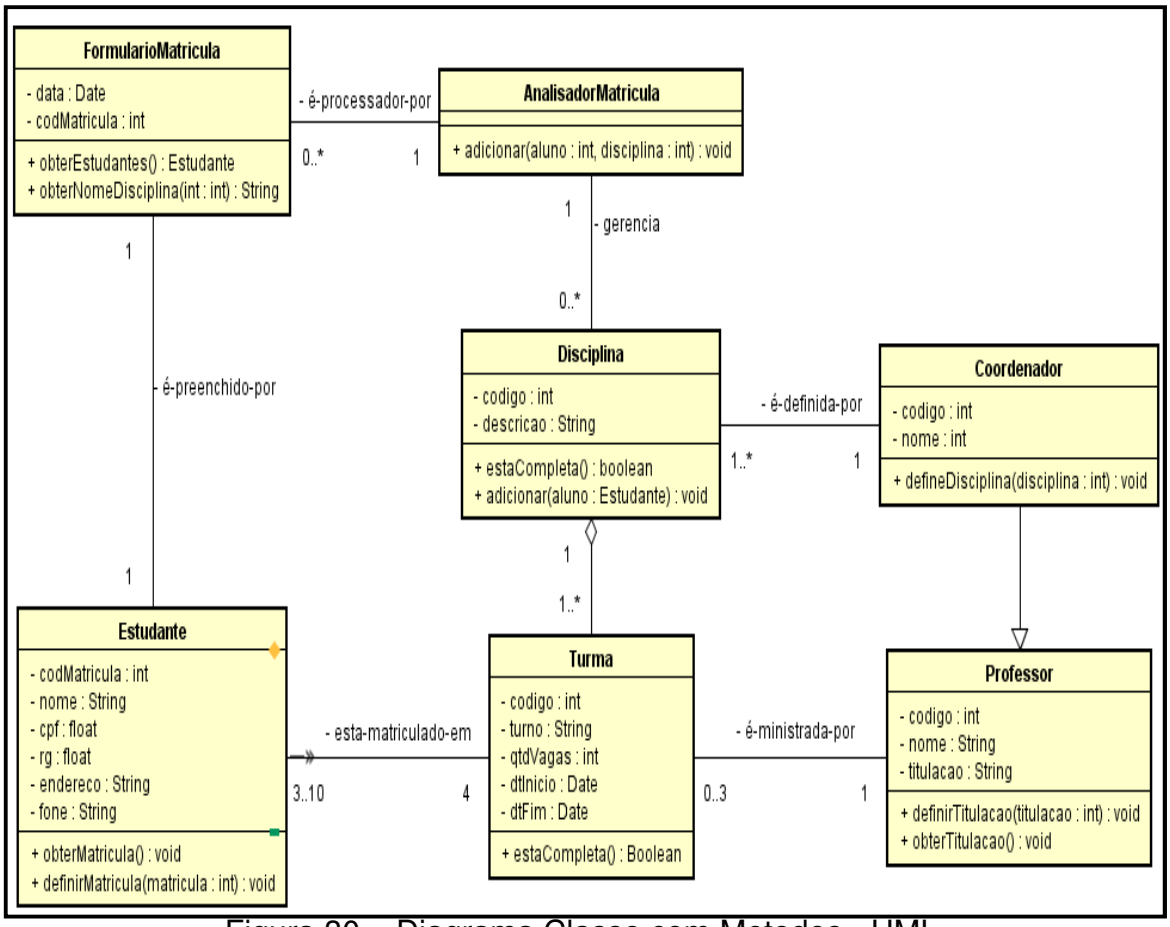

Figura 30 – Diagrama Classe com Metodos - UML

Ainda no diagrama falta o cadastro do usuário e seu relacionamento, com isso vamos acrescentar em nosso diagrama, segue abaixo o modelo aplicando a generalização entre elas.

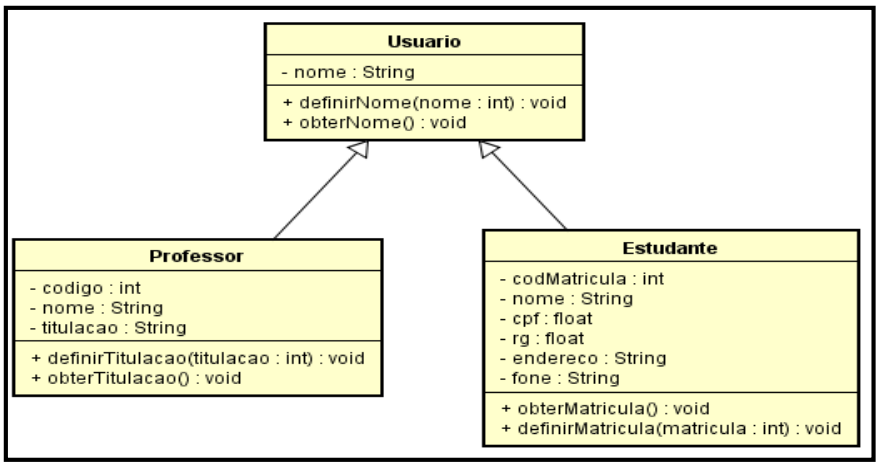

Figura 31 – Relacionamento Generalização – UML

Abaixo segue o diagrama de classe completo, representando assim nosso cenário proposta para realizar as comparações entre as técnicas.

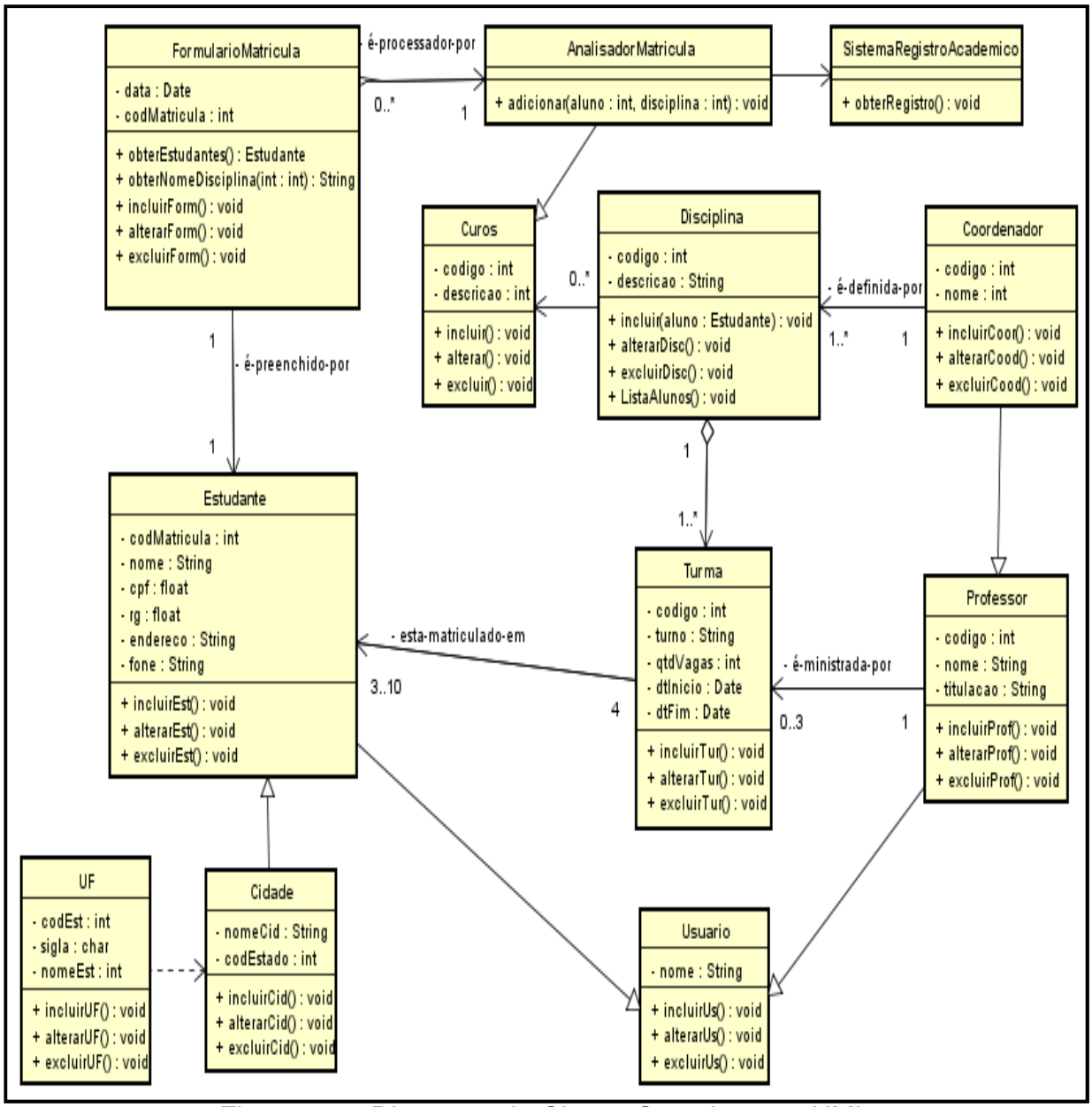

Figura 32 – Diagrama de Classe Completo em UML

Dessa forma terminamos nosso modelo de diagrama de classe em UML, utilizando todos os conceitos vistos ate o momento, abaixo na Figura 33, segue o modelo utilizando as técnicas IDEF1X.

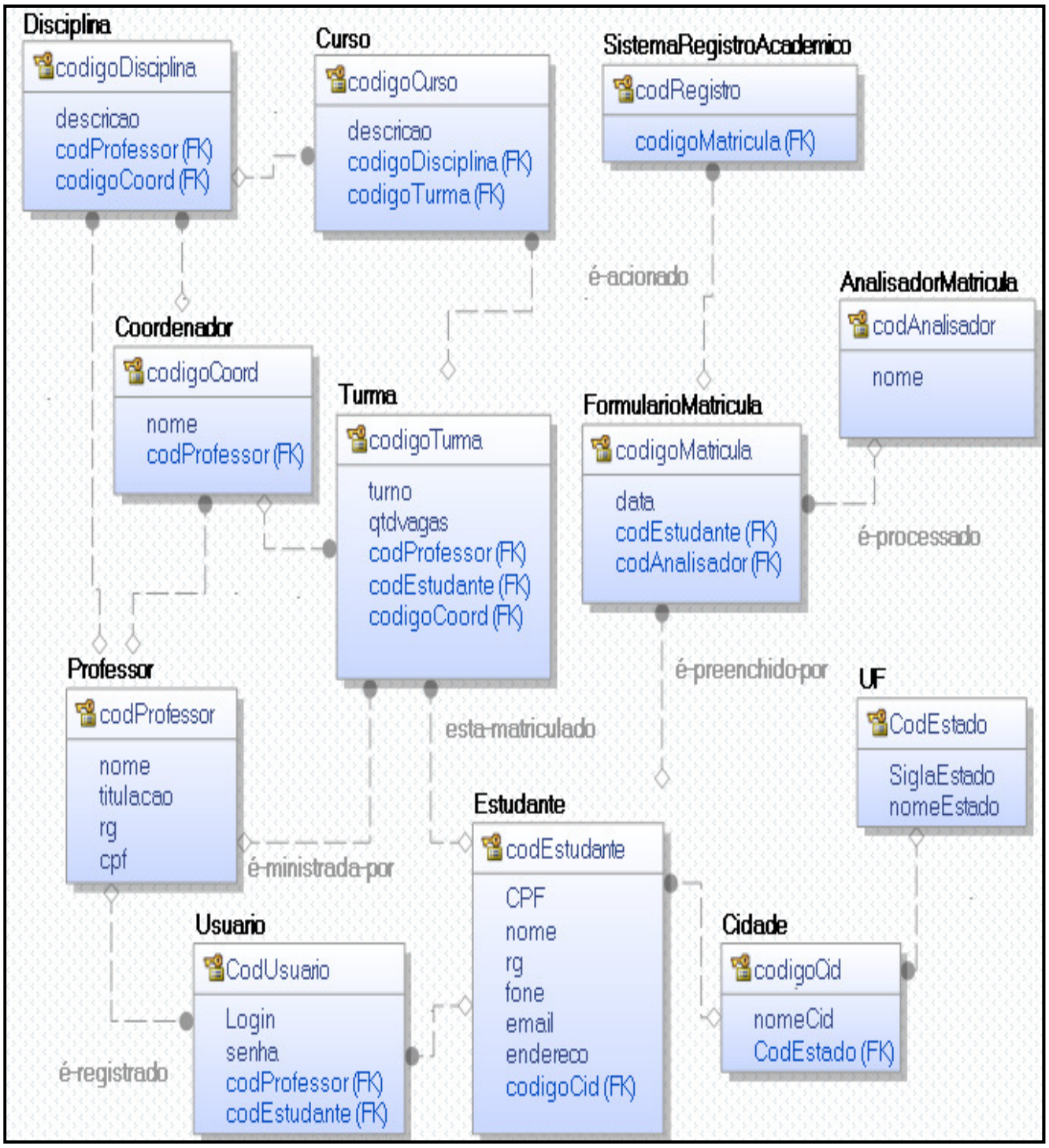

Figura 33 – Diagrama Completo em IDEF1X

Como conclusão geral, ambas as abordagens UML e IDEF1X, pode ser utilizado para modelar quase qualquer ponto de vista útil de um negócio.

7. CONSIDERAÇÕES FINAIS

Este trabalho de conclusão de curso me ajudou a perceber melhor sobre a importância do ensino sobre modelagem de dados, percebendo que é fundamental que professores e alunos, saibam combinar a aprendizagem teórica com a prática, ou seja, tomar consciência de que é necessário pesquisar teorias e novas técnicas para o processo de desenvolvimento de um software, para melhor atender a expectativa dos clientes, e o sucesso do projeto.

Neste trabalho, foram apresentadas as linguagens IDEF1x e UML, como ferramenta de modelagem. Vale ressaltar que essas não são as únicas existentes no mercado, pelo contrario, existe varias opções que podem auxiliar os desenvolvedores nos projetos.

Hay (1995) fez a comparação de varias linguagens de modelagem, incluindo IDEF1X. Conclui declarando que não há linguagem “correta”, e que o sucesso de um projeto de banco de dados depende mais da habilidade do projetista do que da notação escolhida.

Em suma, a necessidade de modelar é uma atividade complexa, pois o sucesso esta ligado a outro fator determinante em qualquer projeto, que é a forma como cada informação é tratada e avaliada pelos participantes do projeto, representando de forma corretas as regras de negócios.

Mesmo assim, os estudos comparativos entre as diversas linguagens para modelagem dos dados têm um mérito, pois os detalhes da sintaxe e semântica de cada linguagem podem favorecer ou prejudicar a qualidade da informação.

Quando vamos começar a desenvolver um projeto, sempre vem a duvida de qual o método mais apropriado. Diante do que estudamos algumas coisas deve ser levado em consideração em sua escolha, ou seja, cabe ao usuário verificar: as tarefas, recursos, padrões... Vai tudo desempenhar um papel na escolha da ferramenta, o desafio não é encontrar a melhor ferramenta, mas a ferramenta certa para o trabalho.

REFERÊNCIAS

ABREU, M.; Machado, F. N. R. **Projeto de Banco de dados:** Uma Visão Pratica. Erica, 1999.

BARKER. **CASE Method Entity Relationship Modelling. Workingham (UK):** Addison-Wesley, 1990.

BEERI, Catriel; BERNSTEIN, Philip A; GOODMAN, Nathan.**Introdução A Teoria Sofisticada para Normalização de Banco de Dados**. VLDB 1978 : 113-124.

BRUCE, T. **Building Quality Databases with IDEF1X Information Models**. Dorset House, 1992.

CHEN, P. P. **"The Entity-Relationship Model - Toward a Unified View of Data".** ACM Transactions on Database Systems.

COAD, Peter; YOURDON, Edward. **Análise Baseada em Objetos**. Rio de Janeiro: Campus, 1992. 225p

CODD, Edgar Frank (1970) **A relational model of data for large shared data banks.** Communications ACM 13(6), 377-387.

COUGO, Paulo Sergio; **Modelagem Conceitual e Projeto de Banco de Dados.**Rio de Janeiro, Editora: Campus, 1997.

DATE, Christopher J. **Introdução a Sistemas de Banco de Dados**. Tradução de Daniel Vieira. 8. Ed. Rio de Janeiro: Editora Elsevier, 2003.

ELMASRI, R. E.; NAVATHE, S.B. **Sistemas de banco de dados**. 4.ed. Rio de Janeiro: Assidon-Weslesy, 2005.

FAGIN, R. "**Multivalued Dependencies and a New Normal Form for Relational Databases",** ACM TODS 2, No 4, December 1977.

GARCIA-Molina, Hector**- Implementação de Sistemas de Banco de Dados**, 2001.
HAY, D.C. **A Comparison of Data Modeling Techniques**. Essential Strategies, 49 p., 1995.

GALANTE, A. C.; MOREIRA, E. L. R.; BRANDAO, F. C., "**Banco de Dados Orientado a Objetos: Uma Realidade**", Revista de Sistemas de Informação da FSMA n.3 (2009) p.55-69

HAY. D. **Princípios de Modelagem de Dados**. São Paulo: Markron Books, 1999.

HEUSER, Carlos Alberto. **Projeto de Banco de Dados**. 3. Ed. Porto Alegre: Editora Sagra Luzzato (Instituto de Informática da UFRGS), 1999.

IDEF Família de Métodos uma abordagem estruturada para o Enterprise Modelagem e Análise. Disponível em: http://www.idef.com/ - Acesso em: 01 de fevereiro de 2015.

IEEE (Institute of Electrical and Electronic Engineers) IDEF1X Standards Working Group. Standard for Conceptual Modeling Language Syntax and Semantics for IDEF1X97 (IDEFobject). IEEE 1320.2 Standards Committee, document P1320.2. Release Draft 0.91 – May 1, 314 p., 1998.

KERN, Vinícius M **Modelagem da informação com IDEF1X: linguagem, método, princípio do consenso**. *Alcance*, 1999, vol. ano VI, n. 3, pp. 99-108. [Journal article (Print/Paginated)]

KROENKE, David M. **Banco de Dados: Fundamentos, Projeto e Implementação**. 6. Ed. Rio de Janeiro: Editora LTC – Livros Técnicos e Científicos S.A, 1999.

LARMAN, Craig. **Utilizando UML e Padrões: Uma Introdução à Análise e ao Projeto Orientados a Objetos e ao Processo Unificado**. 2 ed. Porto Alegre: Bookman, 2004

NIST (National Institute of Standards and Technology). Federal Information Processing Standards Publication 184. Integration definition for information modeling (IDEF1X). Formalization was written by Robert G Brown. Gaithersburg, MD (USA), december, 1993. P.184.

PRESSMAN, R.S. **Software Engeneering – A practitioner's approach**. McGRAW-HILL, 2001.

ROCHA, A. R. C.; MALDONADO, J. C.; WEBER, K. C. **Qualidade de Software: Teoria e Prática**. São Paulo: Prentice Hall, 2001.

SETZER, Valdemar. Bancos de Dados.1.Ed.São Paulo: Editora Edgard Blücher, 2005.

SCOTT, W. A. **"Modelagem Agil: Práticas eficazes para a programação eXtrema e o processo unificado".** Tradução de John Wiley & Song. São Paulo: Editora Artmed, 2002.

SILBERSCHATZ, Abraham; KORTH, Henry F.; SUDARSHAN, S. **Sistemas de Bancos de Dados**. 5. Ed. Tradução de Daniel Vieira. Rio de Janeiro: Editora Elsevier, 2006.

[UML, 2015] Unified Modeling Language. Disponível em: http://www.uml.org Acesso em: 01 de fevereiro de 2015.