FACULDADE DE TECNOLOGIA DE SÃO PAULO – FATEC/SP

PROCESSAMENTO DE DADOS / ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

CARLOS EDUARDO RIBEIRO DA COSTA

ARMAZENAMENTO DE DADOS EM SISTEMAS GERENCIADORES DE BANCO DE DADOS RELACIONAIS (SGDBR’s).

SÃO PAULO – SP
2011

CARLOS EDUARDO RIBEIRO DA COSTA

ARMAZENAMENTO DE DADOS EM SISTEMAS GERENCIADORES DE BANCO DE DADOS RELACIONAIS (SGDBR's).

Monografia apresentada no curso de Tecnologia em Processamento de Dados como requisito parcial para obter o título de Tecnólogo em Processamento de Dados.

Orientador: Professor Mestre Irineu Francisco Aguiar

SÃO PAULO – SP

2011

COSTA, Carlos Eduardo Ribeiro da Costa
Armazenamento de dados em Sistemas Gerenciadores de Banco de Dados Relacionais (SGDBR's)./ Carlos Eduardo Ribeiro da Costa; Orientador: Professor Mestre Irineu Francisco Aguiar – São Paulo, SP – 2011, XX folhas.

Monografia (TCC) – Faculdade de Tecnologia de São Paulo – FATEC/SP, 2011-08-14

1. Modelagem de Banco de Dados – 2. Otimização de Banco de Dados. Título: Armazenamento de dados em Sistemas Gerenciadores de Banco de Dados Relacionais (SGDBR's).

CARLOS EDUARDO RIBEIRO DA COSTA

ARMAZENAMENTO DE DADOS EM SISTEMAS GERENCIADORES DE BANCO DE DADOS RELACIONAIS (SGDBR's).

Monografia apresentada no curso de Tecnologia em Processamento de Dados como requisito parcial para obter o título de Tecnólogo em Processamento de Dados.

Orientador: Professor Mestre Irineu Francisco Aguiar

Banca Examinadora

Professor Mestre Irineu Francisco Aguiar

Instituição: Faculdade de Tecnologia de São Paulo-FATEC/SP.

Julgamento: _____

Assinatura: _____

AGRADECIMENTOS

Agradeço primeiramente ao Grande Criador do Universo, que alguns concebem como DEUS, por estar sempre a meu lado e me dar força e sabedoria para cumprir mais esta etapa de minha jornada.

A minha mãe, que apesar das dificuldades, nunca deixou de trilhar o bom caminho e, a seu modo, sempre tentar passar isto a mim.

A minha adorada esposa, que sempre permaneceu a meu lado e nunca me deixou fraquejar, colocando as primeiras coisas sempre em primeiro lugar.

Aos amigos de curso que compartilharam ao longo destes anos muitos momentos inesquecíveis e conhecimento que nunca se perderão.

Ao grande Mestre Irineu que em nenhum momento me deixou de apoiar e orientar e permitir que pudesse contar com seu valoroso conhecimento na elaboração deste trabalho.

A todos os Professores da FATEC/SP que contribuíram com minha formação.

E, finalmente, mas não menos importante, a todas as pessoas que direta ou indiretamente contribuíram com meu conhecimento, não só para a realização deste trabalho, mas por minha vida inteira.

RESUMO

Este trabalho tem o propósito de elaborar, através de um estudo mais específico, um referencial teórico-prático sobre as características das principais formas e técnicas de Modelagem de Base de Dados Relacionais utilizadas por desenvolvedores de software, Administradores de Dados e Administradores de Banco de Dados.

Qual o objetivo da modelagem de dados? Por que modelar?

- Representar o ambiente observado,
- Documentar e normalizar,
- Fornecer processos de validação,
- Observar processos de relacionamentos entre objetos.

Modelar implica em construir.

Palavras-Chave: Modelagem, Banco de Dados, SGDB.

ABSTRACT

This work aims to develop, through a more specific study, a theoretical and practical knowledge about the characteristics of the main forms and techniques of modelingRelational Database used by software developers, database administrators and databaseadministrators . What is the purpose of data modeling? Why model?
- Represent the environment observed,
- Document and standardize,
- Provide validation processes,
- Observe process of relationships between objects.

Modeling implies building.

Keywords: Modeling, Database, DBMS.

Sumario

Lista de Figuras

Lista de abreviaturas e siglas

SGBDR – Sistema Gerenciador de Banco de Dados Relacionais

DB – Database (Banco de Dados)

SGBD – Sistema Gerenciador de Banco de Dados

GA – Gerenciador de Arquivos

RDBMS – Relational Database Management System

MER – Modelo Entidade-Relacionamento

DER – Diagrama Entidade-Relacionamento

DBA – Database Administrator (Administrador de Banco de Dados)

CEP – Código de Endereçamento Postal

UF – Unidade Federativa

CPF – Cadastro de Pessoas Físicas

CNPF – Cadastro Nacional de Pessoa Jurídica

1:1 – Relacionamento de cardinalidade um para um

1:n - Relacionamento de cardinalidade um para muitos

m:n - Relacionamento de cardinalidade muito para muitos

PK – Primary Key (Chave Primária)

FK – Foreign Key (Chave Estrangeira)

1FN – Primeira Forma Normal

2FN – Segunda Forma Normal

3FN – Terceira Forma Normal

FNBC – Forma Normal de Boyce-Codd

4FN – Quarta Forma Normal

5FN – Quinta Forma Normal

FNPJ – Forma Normal de Projeção de Junção

SQL – Structure Query Language (Linguagem estruturada de consulta)

DDL – Data Definition Language (Linguagem de definição de dados)

DML – Data Manipulation Language (Linguagem de Manipulação de Dados)

DCL – Data Control Language (Linguagem de Controle de Dados)

INTRODUÇÃO

Embasado em bibliografias de estudos empíricos e na experiência na concepção, modelagem, implementação e implantação de base de dados relacionais, este trabalho não tem como objetivo se tornar um norteador para a arte da modelagem de dados, que exige muita dedicação, prática e conhecimento teórico para a compreensão e obtenção da técnica não da melhor, mais da mais apropriada forma de modelar uma base de dados relacionais, visando a correta forma de armazenar o dado, para com isso gerar informação útil, e sim dos benefícios oriundos desta prática, dos benefícios gerados com a mesma, que interferem diretamente no desempenho e otimização do tempo de resposta das consultas realizadas, da qualidade da informação, da facilidade na posterior necessidade de refatoração ou implementação de mais recursos, na qualidade e facilidade de escrita do software no que condiz a extração da informação armazenada na base de dados.

Ao longo deste estudo, acompanharemos um estudo de caso onde poderemos colocar em prática os conceitos aqui apresentados e comprovar a sua utilidade e necessidade.

Em mundo competitivo e globalizado em que vivemos, a informação em muitos caso torna-se um grande diferencial de uma empresa podendo e devendo ser usada sempre em busca de aperfeiçoamento perante a concorrência fazendo pesquisas nas novas tendências e na preferência dos consumidores e/ou clientes, e a qualidade da informação é essencial nesta busca constante.

A tomada de decisão dentro das organizações contemporâneas deriva de análises minuciosas realizadas constantemente aos sistemas de banco de dados.

Na primeira parte deste trabalho falaremos da parte conceitual de dado, informação, banco de dados e os principais conceitos elementos que rodeiam um sistema gerenciador de banco de dados.

Na segunda parte do trabalho falaremos sobre as técnicas de modelagem e como devemos aplicá-las ás nossa necessidades neste trabalho.

Na terceira parte do trabalho, demonstraremos de forma prática, através de um estudo de caso os conceitos estudos.

Na última parte do trabalho, faremos um resumo dos principais tópicos abordados, dando vazão ás considerações finais.

1- Sistema Gerenciador de Banco de Dados Relacionais (SGBDR)

Um Banco de Dados DB (do inglês *DataBase*) são lugares eletrônicos para armazenar dados. Os DB mantêm as informações armazenadas em tabelas. Uma tabela é uma estrutura que consiste em pelo menos uma coluna, mas normalmente possui mais. Um DB portanto, é uma coleção de uma ou mais tabelas de informações relacionadas.

Um **SGBD** é uma coleção de programas que permitem ao usuário definir, construir e manipular Bases de Dados para as mais diversas finalidades.

Um conceito que deverá ficar bastante claro inicialmente é o que envolve a separação clara entre os Gerenciadores de Base de Dados dos Gerenciadores de Arquivo(GA).

Um Sistema Gerenciador de Banco de Dados Relacionais(SGBDR) do inglês Relational database management system (RDBMS) atende aos requisitos de:

1.1 - Atomicidade

Quando uma transação é enviada do usuário para o servidor de banco de dados, a operação é salva completamente ou é revertida. Em outras palavras, se o disco rígido não pudesse armazenar a transação inteira, a transação é revertida como se nunca tivesse acontecido. Desta maneira, os dados sempre podem ser mantidos consistentes. Um exemplo clássico é um saque em um caixa eletrônico, onde o usuário efetua a solicitação de um valor, é feita a verificação se ele possui disponibilidade do valor solicitado, é gravado na base de dados o valor do saque e o novo saldo, mas se por algum problema a operação não for confirmada (queda de energia, falta de dinheiro no caixa eletrônica, etc.), a operação é revertida, e é como se o cliente não tivesse efetuada nenhuma operação.

1.2 - Consistência

As operações que violam uma regra ou um mecanismo semelhante não são salvas no DB. Se um usuário não tem permissão de acesso, tentativas de gravar na tabela são rejeitadas. Os SGDB’s possuem recursos para adicionar quantos usuários forem necessários e pode definir diferentes níveis de privilégios para cada um deles (leitura, escrita, exclusão, etc..), se o usuário que está conectado tiver permissão apenas de leitura, ele não conseguirá efetuar gravações, atualizações e etc.

1.3 - Isolamento

Uma operação no DB está completamente antes da outra operação torna-se ciente dela. Se um usuário estiver adicionando registros a uma tabela enquanto outro estiver lendo as mesmas tabelas, resultados de operações parciais não são mostrados ao leitor. As informações parciais poderiam levar a resultados inconsistentes e confusos para o leitor.

1.4 - Durabilidade

Uma vez que uma transação é salva no DB, é garantido que ela estará lá. Se um evento catastrófico acontecer com o dispositivo, o banco de dados ainda armazenará essa transação (supondo que nenhuma outra falha de hardware cause uma corrupção de disco).

Atende ainda, as seguintes regras básicas:

1.5 - Auto-Contenção

Um SGBD não contém apenas os dados em si, mas armazena completamente toda a descrição dos dados, seus relacionamentos e formas de acesso. Normalmente esta regra é chamada de Meta-Base de Dados. Em um GA, em algum momento ao menos, os programas aplicativos declaram estruturas (algo que ocorre tipicamente em C, COBOL e BASIC), ou geram os relacionamentos entre os arquivos (típicos do ambiente xBase). Por exemplo, quando você é obrigado a definir a forma do registro em seu programa, você não está lidando com um SGBD.

1.6 - Independência dos Dados

Quando as aplicações estiverem realmente imunes a mudanças na estrutura de armazenamento ou na estratégia de acesso aos dados, podemos dizer que esta regra foi atingida. Portanto, nenhuma definição dos dados deverá estar contida nos programas da aplicação. Quando você resolve criar uma nova forma de acesso, um novo índice, se precisar alterar o código de seu aplicativo, você não está lidando com um SGBD.

1.7 - Abstração dos Dados

Em um SGBD real é fornecida ao usuário somente uma representação conceitual dos dados, o que não inclui maiores detalhes sobre sua forma de armazenamento real. O chamado Modelo de Dados é um tipo de abstração utilizada para fornecer esta representação conceitual. Neste modelo, um esquema das tabelas, seus relacionamentos e suas chaves de acesso são exibidas ao usuário, porém nada é afirmado sobre a criação dos índices, ou como serão mantidos, ou

qual a relação existente entre as tabelas que deverá ser mantida íntegra. Assim se você desejar inserir um pedido em um cliente inexistente e esta entrada não for automaticamente rejeitada, você não está lidando com um SGBD.

1.8 - Visões

Um SGBD deve permitir que cada usuário visualize os dados de forma diferente daquela existente previamente no Banco de Dados. Uma visão consiste de um subconjunto de dados do Banco de Dados, necessariamente derivados dos existentes no Banco de Dados, porém estes não deverão estar explicitamente armazenados. Portanto, toda vez que você é obrigado a replicar uma estrutura, para fins de acesso de forma diferenciada por outros aplicativos, você não está lidando com um SGBD.

1.9 - Transações

Um SGBD deve gerenciar completamente a integridade referencial definida em seu esquema, sem precisar em tempo algum, do auxílio do programa aplicativo. Desta forma exige-se que o banco de dados tenha ao menos uma instrução que permita a gravação de uma série modificações simultâneas e uma instrução capaz de cancelar um série de modificações. Por exemplo, imaginemos que estejamos cadastrando um pedido para um cliente, que este deseje reservar 5(cinco) itens de nosso estoque, que estão disponíveis e portanto são reservados, porém existe um bloqueio financeiro (duplicatas em atraso) que impede a venda. A transação deverá
ser desfeita com apenas uma instrução ao Banco de Dados, sem qualquer modificações suplementares nos dados.

1.10 - Acesso Automático:

Em um GA uma situação típica é o chamado Dead-Lock, o abraço mortal. Esta situação indesejável pode ocorrer toda vez que um usuário travou um registro em uma tabela e seu próximo passo será travar um registro em uma tabela relacionada à primeira, porém se este registro estiver previamente travado por outro usuário, o primeiro usuário ficará paralisado, pois, estará esperando o segundo usuário liberar o registro em uso, para que então possa travá-lo e prosseguir sua tarefa. Se por hipótese o segundo usuário necessitar travar o registro travado pelo primeiro usuário, afirmamos que ocorreu um abraço mortal, pois cada usuário travou um registro e precisa travar um outro, justamente o registro anteriormente travado pelo outro! Imaginemos um caso onde o responsável pelos pedidos acabou de travar o Registro

Item de Pedido, e, necessita travar um registro no Cadastro de Produtos, para indicar uma nova reserva. Se concomitantemente estiver sendo realizada uma tarefa de atualização de pendências na Tabela de Itens, e para tanto, previamente este segundo usuário travou a Tabela de Produtos, temos a ocorrência do abraço mortal. Se a responsabilidade de evitar esta ocorrência for responsabilidade da aplicação, você não está lidando com um SGBD.

2 - Projeto de Banco de Dados

O processo de projetar um corretamente DB complexo não é fácil. Requer tempo e esforço. O modelo de dados normalmente é o primeiro passo no processo. Isso inclui definir os processos de negócios e construir um Diagrama de Entidade Relacionamento.

2.1 - O que é modelagem de dados

Modelagem de dados é o ato de explorar estruturas orientadas a dados. Como outros artefatos de modelagem, modelos de dados podem ser usados para uma variedade de propósitos, desde modelos conceituais de alto nível até modelos físicos de dados. Do ponto de vista de um desenvolvedor atuando no paradigma orientado a objetos, modelagem de dados é conceitualmente similar à modelagem de classes. Com a modelagem de dados identificamos tipos de entidades da mesma forma que na modelagem de classes identificamos classes. Atributos de dados são associados a tipos de entidades exatamente como associados atributos e operações às classes.

Modelagem de dados tradicional é diferente da modelagem de classes porque seu foco é totalmente nos dados, modelos de classes permitem explorar os aspectos comportamentais e de dados em um domínio de aplicação, já com o modelo de dados podemos apenas explorar o aspecto dado.

2.2 - Análise de requisitos ou Modelos de dados conceituais

Esse primeiro passo é composto por reuniões aparentemente sem fim que analisam o que é exigido para o futuro DB. Os proprietários e usuários do DB definem seus processos de negócios e documentam as entidades, atributos e relacionamentos que comporão o DB.

Essa análise é a mais importante para o sucesso do projeto de banco de dados. Sem uma visão geral e completa dos processos do negócio - ou completa documentação do que a aplicação deve ter e do que ela deve ser - o projeto falhará. Independentemente de quanto tempo for dedicado ás próximas fases do projeto, o destino será um projeto falho (no melhor dos casos) ou um desastre (no pior) se negligenciarmos um item importante neste passo.

2.3 - Projeto ou Modelo lógico do banco de dados

Tem por objetivo avaliar o esquema conceitual frente às necessidades de uso do banco de dados pelos usuários e aplicações, realizando possíveis refinamentos com a finalidade de melhorar o desempenho das operações.

Nesta fase do projeto, realizamos a Normalização das tabelas.

Um esquema lógico é uma descrição da estrutura do banco de dados que pode ser processada por um SGBD.

Depende do modelo de dados adotado pelo SGBD, mas não especificamente do SGBD. Neste mapeamos o modelo conceitual para o modelo de implementação (físico). O projeto lógico gera o esquema lógico.

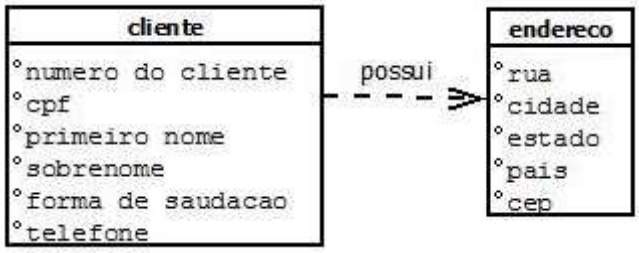


Figura 1 – Exemplo de modelo lógico de dados.

2.4 - Projeto ou Modelo físico de dados

Este toma por base o esquema lógico para gerar o esquema físico e é direcionado para um específico SGBD.

O projeto físico gera o esquema físico.

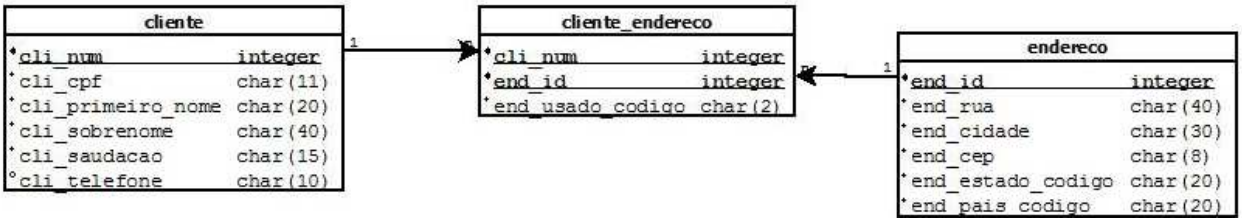


Figura 2 – Exemplo de modelo físico de dados

2.5 - Modelagem de Dados usando o Modelo Entidade-Relacionamento

Apresentaremos os conceitos de modelagem do modelo Entidade-Relacionamento (**MER**), que é um modelo de dados conceitual de alto nível, além de muito popular. Esse modelo e suas variações são normalmente empregados para o projeto conceitual de aplicações de um banco de dados, e muitas ferramentas de projeto de um banco de dados aplicam seus conceitos. Descreveremos os conceitos da estruturação de dados básica e as restrições do MER, e discutiremos seu uso no projeto de esquemas conceituais para aplicações de bancos de dados. Apresentaremos também a notação diagramática associada ao MER, conhecida por diagramas ER (DER).

- Definição: modelo baseado na percepção do mundo real, que consiste em um conjunto de objetos básicos chamados entidades e nos relacionamentos entre esses objetos
- Objetivo: facilitar o projeto de banco de dados, possibilitando a especificação da estrutura lógica geral do banco de dados.

A estrutura lógica geral de um banco de dados pode ser expressa graficamente por um DER.

2.6 - Como modelar dados

É crucial para um desenvolvedor de aplicação ter uma noção dos fundamentos de modelagem de dados não apenas para ler os modelos de dados, mas também para trabalhar efetivamente com os DBA's (Databases Administrator – Administradores de Banco de Dados) responsáveis pelos aspectos relacionados aos dados do projeto.

As seguintes tarefas devem ser realizadas de forma interativa:

- Identificar os tipos de entidade;
- Identificar os atributos;
- Aplicar convenção de nomes;
- Identificar relacionamentos;
- Associar chaves;
- Normalizar para reduzir as redundâncias de dados;
- Desnormalizar para melhorar o desempenho;

2.7 - Componentes do DER (Peter Chen)

Retângulos

Representam conjuntos- entidade

Elipses

Representam atributos

Losangos

Representam conjuntos - relacionamento

Linhas

Ligam atributos a conjuntos- entidade e conjuntos - entidade a conjuntos- relacionamento.

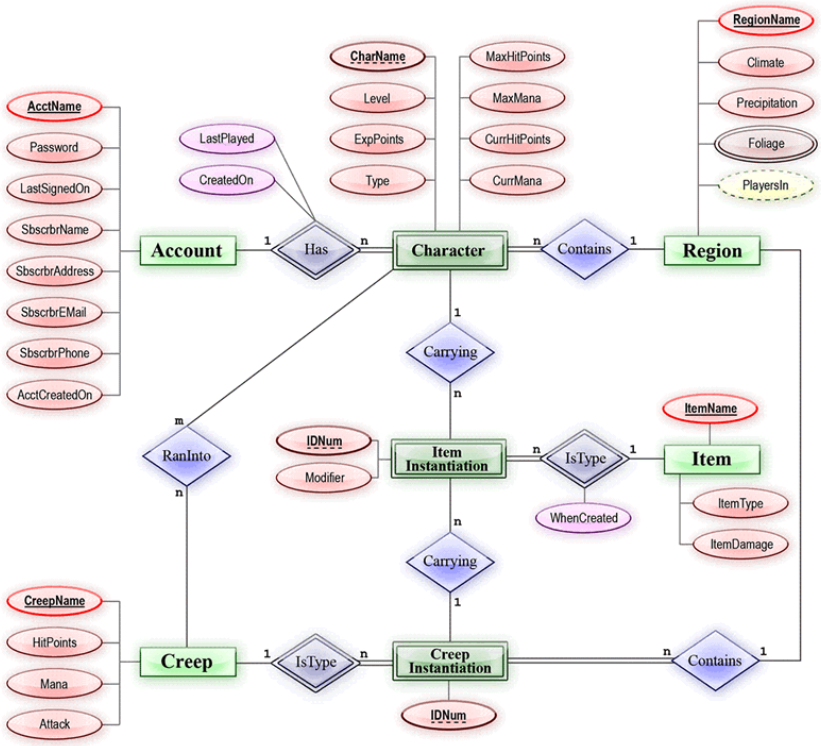


Figura 3 – Modelo de DER usando a notação de Peter Chen

2.8 - Entidades e Conjunto-Entidade

Entidade

É uma representação abstrata de um objeto do mundo real

Ex. : O fornecedor Pedro, com código F1, Pessoa, Carro, etc.

Conjuntos- Entidade

Grupo de entidades que possui características semelhantes

Ex. : Conjunto de todos os estudantes (considerando que estudante é uma entidade.)

2.9 - Identificar os Tipos de entidades

Uma entidade é conceitualmente similar ao conceito de orientação de objeto de uma classe – uma entidade representa uma coleção de objetos similares. Uma entidade pode representar uma coleção de pessoas, lugares, coisas, eventos ou conceitos.

Idealmente, uma entidade deveria ser normal, descrevendo de forma coesa uma informação do mundo real. Uma entidade normal descreve um conceito, tal como uma classe coesa modela um conceito. Por exemplo, cliente e venda são claramente conceitos diferentes e faz sentido modelá-los como entidades diferentes.

2.10 - Atributos(Campos)

Cada entidade terá um ou mais atributos de dados.

Atributos devem ser coesos do ponto de vista do domínio da aplicação.Usar o nível de detalhe correto pode ter um impacto significativo no esforço de desenvolvimento e manutenção.

-O nome dos atributos por convenção, sempre é em caixa baixa (letras minúsculas) e quando é necessário um nome composto, usa-se o caractere _ (underline) para uni-los (cod_forn) ou a primeira letra do segundo nome vem em caixa alta (codForn).

-Não é aconselhável e nem usual usar caracteres especiais (/*,-, etc..) para compor nome de tabelas e atributos;

Atributo

Elemento de dado que contém informação que descreve uma entidade

Atributo Monovalorado

Assume um único valor para cada elemento do conjunto- entidade

Ex. : Nome

Atributo Composto

São atributos que podem ser subdivididos em vários atributos. Por exemplo, um endereço, que pode ser dividido em:

Logradouro(Rua, Avenida, Praça, etc..), Nome do endereço(nome propriamente dito), Número, Complemento, Bairro, CEP, Cidade, UF.

Atributo Multivalorado

São atributos que podem conter mais de um valor para um mesmo registro. Por exemplo, uma pessoa pode possuir mais de um número de telefone, pode ter cadastrado mais de um dependente, neste caso, telefones e dependentes são atributos multivalorados.

Atributo atômico

São atributos que não podem ser subdivididos e não são multivalorados. Logo é indivisível. Por exemplo: CPF, CNPJ, Preço unitário.

Atributo Determinante

Identifica cada entidade de um conjunto-entidade (também conhecido com atributo chave)

Ex. : Cod_ Func (código do funcionário)

Domínio de um Atributo

Conjunto de valores permitidos para o atributo

Ex. : Sexo {M, F}.

Exemplo de duas entidade (Fornecedor e Produto) e seus atributos:

Fornecedor
cod_forn (código do Fornecedor)
fornecedor (Razão Social, Nome)
logradouro (Tipo de endereço)
endereco (Endereço do fornecedor)
num_end (Número)
bairro
cidade
uf

Produto
cod_prod (código do Produto)
tipo (tipo de produto)
descricao
cod_forn

2.11 - Cardinalidade

Número (mínimo, máximo) de ocorrências de entidade associadas a uma ocorrência da entidade em questão através do relacionamento.

Cardinalidade mínima

É o número mínimo de ocorrências de entidade que são associadas a uma ocorrência da mesma (auto-relacionamento) ou de outra(s) entidade(s) através de um relacionamento.

A cardinalidade mínima 1 recebe a denominação de associação obrigatória, já que ela indica que o relacionamento deve obrigatoriamente associar uma ocorrência de entidade a outra. A cardinalidade mínima 0 (zero) recebe a denominação de associação opcional.

Cardinalidade máxima

É o número máximo de ocorrências de entidade que são associadas a uma ocorrência da mesma ou de outra entidade através de um relacionamento. Apenas duas cardinalidades máximas são relevantes: a cardinalidade máxima 1 e a cardinalidade máxima n (muitos).

2.12 - Relacionamento

Estrutura que indica a associação de elementos de duas ou mais entidades.

Ex. :

Figura 4 – Modelo de relacionamento formando um depósito de dados

Na Figura 4 temos um modelo de relacionamento entre duas entidades (Pedido e Produto) formando o depósito de dados Pedido.

Atributos de relacionamento: depende de todos os conjuntos- entidade associados entre si.

Restrições de Mapeamento(Cardinalidade).

Um- para- um (1:1)

Uma entidade em A está associada no máximo a uma entidade em B e uma entidade em B está associada no máximo a uma entidade.

Um exemplo disto seria um o caso de uma empresa que possui o cadastro de seus funcionários, de seus departamentos e encarregados. Um funcionário é lotado em um departamento e que é administrado por um gerente.

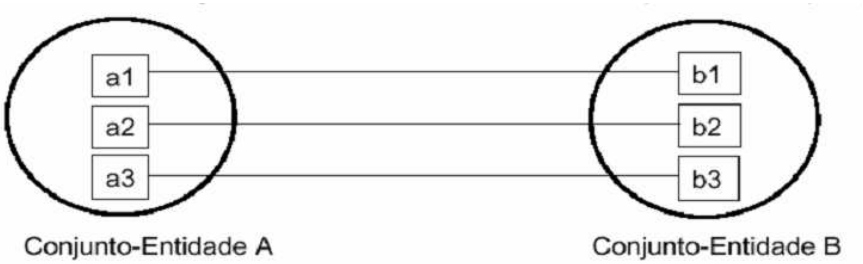


Figura 5 – Exemplo de relacionamento 1:1

Na Figura 5 temos um relacionamento entre Entidades, onde cada registro do Conjunto-Entidade A se identifica com apenas 1 registro do Conjunto-Entidade B

Figura 6 – Modelo de um Relacionamento 1:1

Na Figura 6 temos um exemplo onde um funcionário tem 1 gerente e 1 departamento tem um gerente.

Um- para- muitos (1:n)

Uma entidade em A está associada a qualquer número de entidades em B, enquanto uma entidade em B está associada no máximo a uma entidade em A. Chave estrangeira na entidade de cardinalidade muitos.

No exemplo ilustrado na imagem 6, onde um departamento possui vários funcionários lotados nele, e cada funcionário é lotado em apenas um departamento.

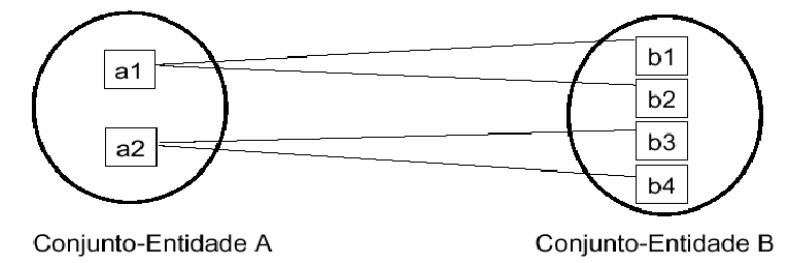


Figura 7 – Exemplo de relacionamento 1:n entre Entidades

Na Figura 7 temos um exemplo de relacionamento 1:n, onde cada registro do Conjunto-Entidade A se identifica com 1 ou mais registros do Conjunto-Entidade B

Figura 8 – Modelo de um relacionamento 1:n

Na Figura 8 temos um exemplo de relacionamento 1:n, onde um Departamento possui vários funcionários mas, 1 funcionário pertence a 1 Departamento.

Muitos- para- muitos (m:n)
Uma entidade em A está associada a qualquer número de entidades em B, e uma entidade em B está associada a qualquer número de entidades em A. Chave estrangeira de ambas entidades tem que estar em uma tabela extra que implementa o relacionamento.
No exemplo ilustrado na imagem 9, um projeto é executado por vários funcionários e um funcionário pode estar envolvido em mais de um projeto concomitantemente.

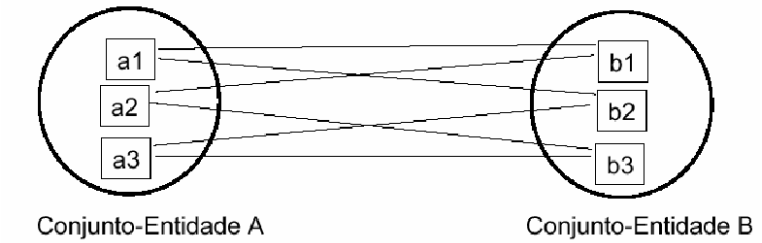


Figura 9 – Exemplo de relacionamento m:n entre Entidades

Na Figura 9 temos um modelo de relacionamento m:n entre entidades, onde cada registro do Conjunto-Entidade A se identifica com 1 ou mais registros do Conjunto-Entidade B e vice-versa.

Figura 10 – Modelo de um Relacionamento m:n

Na Figura 10 temos um modelo de relacionamento m:n onde vários funcionários estão alocados em muitos projetos, sendo que 1 funcionário está em mais de um projeto e um projeto tem muitos funcionários alocados.

2.12 – Chave Primária

Chave: é um conjunto de um ou mais atributos que, tomados coletivamente, permite- nos identificar unicamente uma entidade no conjunto- entidade.

Integridade de Entidade: Nenhum atributo que participe da chave de um conjunto- entidade deve aceitar valores nulos

· Aspectos relevantes das chaves

* A questão fundamental do projeto de chaves é reduzir ao máximo os efeitos de redundância

* A alteração dos valores de campos constituintes da chave primária ou a remoção de uma entidade de um conjunto entidade pode ocasionar problemas de integridade referencial.

Super Chave

É qualquer conjunto de atributos contendo uma chave, seja ela primária ou candidata.

Chave Candidata

Atributo ou grupo de atributos que têm a propriedade de identificar unicamente um registro. Pode vir a ser uma chave Primária. A chave candidata que não é chave Primária também se chama “chave alternativa”. Por exemplo, uma tabela com os campos> código, nome e cpf. Tanto código como CPF podem ser chave primária e ambas são chaves candidatas pois permitem identificar o registro.

Chave Primária (Primary Key (PK))

Uma Chave–Primária (PK), do inglês “Primary Key”, é um identificador único, onde identifica um registro em uma tabela e não pode ter repetições e não pode ter um valor nulo. Imagine um cadastro de funcionários com dois ou mais funcionários com o mesmo código identificador, seria calamitoso pois imagine os registros de outras tabelas que informam ocorrências ou informações para aquele código (dependentes, salários, produção...). Da mesma forma, uma PK não pode ser nula ou em branco*.

*Embora pareça uma coisa redundante, existe uma grande diferença entre valor nulo ou valor em branco. Valor nulo, é sem valor, não há nada registrado; enquanto valor em branco, não existe caracteres visíveis cadastrados, mas há uma valor cadastrado.

Chave Estrangeira (Foreign Key (FK))

Uma FK é um identificador da tabela relacionada á tabela atual. É através dela que sabemos a qual registro na outra tabela este registro está relacionado.

Por exemplo, na tabela “funcionário”, temos o atributo “cod_func” e na tabela telefone, nos números de telefone pertencente a este funcionário teremos o atributo “cod_func” como chave estrangeira, identificando a qual funcionário este número de telefone pertence.

As chaves primárias de cada entidade e relacionamento estão sublinhadas:

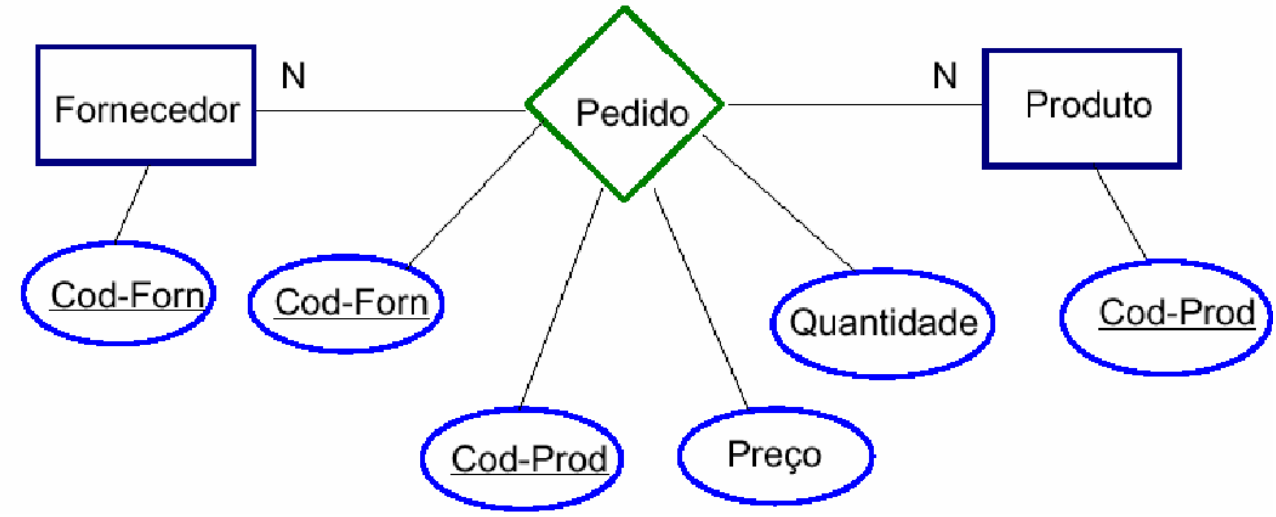


Figura 11 – Modelo Entidade Relacionamento exibindo os atributos chaves.

2.14 - Dependência Funcional

Uma dependência funcional é um relacionamento entre dois ou mais atributos de forma que o valor de um atributo identifique o valor para cada um dos outros atributos, ou seja, um atributo está relacionado a outro.

Por exemplo:

Código do Cliente -> Nome do Cliente

Neste exemplo, para descobrirmos o nome do cliente, primeiramente precisamos saber qual é o código dele. Assim, o campo/atributo nome é dependente do campo/atributo código. Observe que o inverso não é verdadeiro, pois poderá ocorrer de possuirmos dois clientes com o mesmo nome.

Outro fato importante é que em uma tabela podemos ter mais de uma dependência funcional.

Exemplo:

Código do Cliente -> Nome do Cliente

Código do Cliente -> UF do Cliente

ou

Código do Cliente -> [Nome do Cliente, Código do Cliente]

Dependência Funcional Parcial

Ocorre quando os atributos não chave não dependam funcionalmente de toda a chave primária quando esta for composta. Assim, nas tabelas onde a chave primária for composta, todos os atributos devem depender de toda a chave primária. Caso a dependência seja de parte da chave, verificamos a existência de dependência funcional parcial.

Dependência Funcional Transitiva

Na definição dos campos de uma entidade podem ocorrer casos em que um campo não seja dependente diretamente da chave primária ou de parte dela, mas dependente de outro campo não chave da tabela e isto caracteriza a dependência funcional transitiva. É importante destacar a diferença entre dependência funcional transitiva e dependência funcional parcial pois enquanto a primeira não depende de nenhum campo chave e sim a um campo não chave, a segunda depende de parte de uma chave primária composta e não de toda ela.

Dependência Funcional Multivalorada

Uma dependência funcional multivalorada ocorre quando, para cada valor de um atributo **A**, há um conjunto de valores para outros atributos **B** e **C** que estão associados a ele, mas são independentes entre si.

Por exemplo, supondo uma tabela filmes, consideremos que cada filme possui um número de registros de atores e um número de registros de produtores, mas atores e produtores são independentes entre si.

Dependência Funcional Cíclica:

Uma dependência funcional cíclica ocorre quando temos dependências como: A->B, B->C, C->A.

Por exemplo:

Um professor ministra disciplinas: professores->disciplinas;

Um professor escreve apostilas: professores->apostilas;

Cada disciplina pode ter várias apostilas: disciplinas->apostilas;

Desta forma temos a relação cíclica professor->disciplinas, disciplinas->apostilas e professores->apostilas.

2.15 - Normalização

É um processo onde se aplica regras a todas as entidades (tabelas) do banco de dados, afim de evitar falhas no projeto, como redundância de dados, mistura de diferentes assuntos numa mesma entidade, entre outros problemas. As formas normais mais conhecidas, são a primeira, segunda e terceira formas normais. Basicamente, aplicando e respeitando as regras de cada uma dessas formas normais, poderemos garantir um banco de dados mais íntegro, com uma grande possibilidade de sucesso no seu projeto.

A normalização é composta de 5 fases mas apenas as primeiras três são usadas normalmente, são elas:

2.15.1 - Primeira Forma Normal (1FN)

Uma tabela se encontra na 1FN se todos os atributos possuírem apenas valores atômicos (simples e indivisíveis) e os valores de cada atributo no registro também deve ser um valor simples (o atributo não é composto). Desta forma, caso existam atributos compostos, estes devem ser divididos em atributos atômicos. Caso existam atributos multivalorados, estes devem fazer parte de outra tabela, que está relacionada com a tabela original.

Para exemplificar, vejamos a tabela funcionário:

Figura 12 - Tabela funcionário não normalizada (não está na 1FN)

A referida tabela não está na 1FN, pois possui um atributo multivalorado (telefone) e um atributo composto (endereco). Este último deve ser dividido em vários atributos atômicos, como mostrado na próxima imagem, e o telefone deve ser apresentado em uma tabela separada relacionada com a inicial, tendo o atributo cod_func como chave estrangeira. Desta forma, as duas tabelas estão atendendo a 1FN.

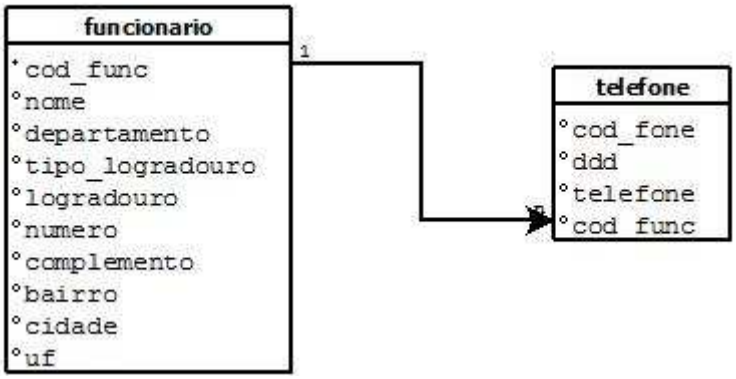


Figura 13 – Tabela funcionario na 1FN dando origem à tabela telefone.

Percebemos aqui que o relacionamento é de cardinalidade 1:n, ou seja, 1 funcionário pode possuir 1 ou mais números de telefone cadastrado.

2.15.2 - Segunda Forma Normal (2FN)

Uma tabela se encontra na 2FN se estiver na 1FN e não possuir dependência funcional parcial. Caso existam atributos que não dependam integralmente da chave primária, devemos retirar da tabela todos eles e dar origem a uma nova tabela.

Para exemplificar, seja a tabela vendas, onde os atributos correspondem a:

- nro: número da venda;
- codProd: código do produto;
- nomeProd: nome do produto.
- vlUnit: valor unitário;
- qtdde: quantidade;
- vlTt: valor total;

Figura 14 - Tabela de vendas

Supondo que a PK desta tabela seja os atributos nro e codProd, logo, trata-se de uma PK composta, assim iremos verificar se esta tabela encontra-se na 2FN.

O primeiro passo é verificar se a tabela vendas encontra-se na 1FN. Podemos verificar que não existem atributos compostos e/ou multivalorados, logo a 1FN é verificada nesta tabela.

Posteriormente, precisamos verificar se existe dependência parcial de chave. Verificamos que codProd->nomeProd,vlUnit são determinados pelo código do produto, desta forma existem atributos que não dependem integralmente da PK, logo a tabela não está na 2FN.

Para adequar a tabela vendas na 2FN teremos que dividi-la em 2 tabelas, vendas e produtos. Os atributos da dependência parcial devem fazer parte da tabela produtos.

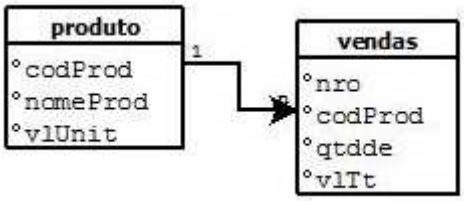


Figura 15 – Tabela de vendas na 2FN, sem dependência funcional parcial.

O relacionamento entre as duas tabelas é de cardinalidade 1:n, onde 1 produto pode estar presente em muitas vendas.

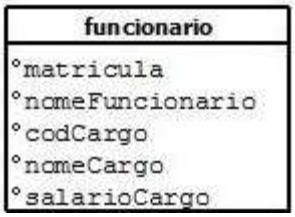
Desta forma, com duas novas tabelas, verificamos a adequação à 2FN. Para verificar a adequação à 2FN, podemos seguir alguns passos:

- se existirem apenas atributos atômicos, as tabelas encontram-se na 1FN;
- caso não existam chaves primárias compostas, não há como existir dependência funcional parcial, e as tabelas encontram-se na 2FN;
- caso existam chaves primárias compostas, deve-se verificar a dependência funcional parcial;

2.15.3 - Terceira Forma Normal (3FN)

Uma tabela está na 3FN se estiver na 2FN e não possuir dependência funcional transitiva.

Para exemplificar, vamos verificar a tabela funcionário:



-matricula: número da matricula do funcionário(PK);
- nomeFuncionario: nome do funcionário;
-codCargo: código do cargo ocupado pelo funcionário;
-nomeCargo: nome do cargo ocupado pelo funcionário;
- salarioCargo: salário do cargo;

Figura 16 – Tabela funcionario.

O primeiro passo é verificar se a tabela funcionário está na 2FN. Verificamos que não existe dependência funcional parcial, pois não há chave composta, e os atributos são todos atômicos, logo ela se encontra na 2FN.

Após a constatação que a tabela está na 2FN, verificamos se há a existência de dependência funcional transitiva em tabela à chave primária. Verificamos então que nesta tabela há a seguinte dependência:

- codCargo->nomeCargo, salarioCargo.

Percebemos que codCargo não é chave primária e os atributos nomeCargo e salarioCargo são dependentes dele, logo, a tabela “funcionario” não está na 3FN.

Para adequar a tabela funcionário a 3FN é necessário separá-la em duas ou mais tabelas de forma a eliminar a dependência funcional transitiva. Neste caso, devemos separar a tabela em duas, funcionario e cargo, da seguinte forma:

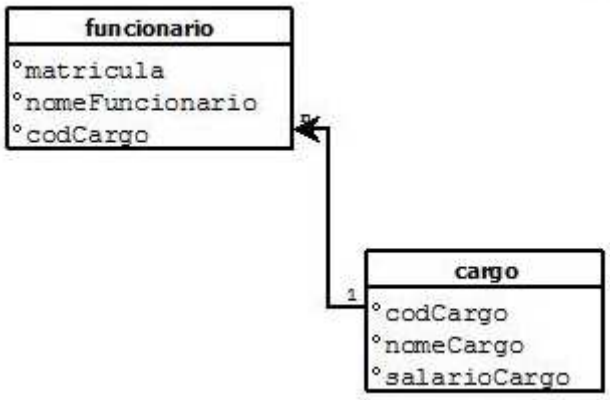


Figura 17 – Tabela funcionario na 3FN dando origem a tabela cargo.

O relacionamento nesta figura é de cardinalidade 1:n, onde muitos funcionários podem ter o mesmo cargo.

Verifica-se agora que na tabela funcionario o atributo codCargo é uma chave estrangeira referenciando a tabela cargo. Efetuada as alterações, ambas se encontram agora na 3FN.

2.15.4 - Forma Normal de Boyce-Codd (FNBC)

Esta forma normal pode ser aplicada às tabelas que se encontram na 3FN, sendo que uma relação está na FNBC se para toda dependência funcional X->Z, X é uma Super Chave.

Para exemplificar, vamos verificar a tabela curso, que se encontra na 3FN porque está na 2FN e não apresenta dependência funcional transitiva.

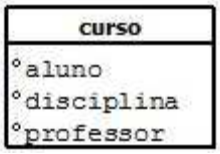


Figura 18 – Tabela curso.

Ao contrário das formas normais que vimos até aqui, a FNBC não exige que a tabela já esteja na 3FN para se aplicada, ou seja, podemos ir direto de uma tabela não normalizada para a FNBC. Ela serve de atalho para atingirmos as formas normais 1FN, 2FN e 3FN.

Verificamos agora se existe dependência funcional e constatamos a seguinte dependência: disciplina->professor

E neste caso, disciplina é uma Super Chave. Esta dependência acontece porque para cada disciplina existe um ou mais professores, logo, os professores dependem de disciplina. Para

resolvermos esta dependência, devemos separar a tabela curso em duas tabelas, tutoria e curso, da seguinte forma:

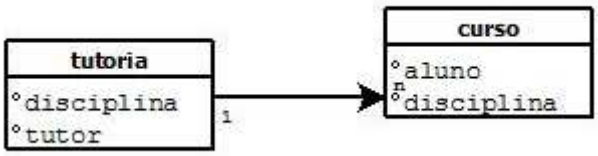


Figura 19 – Tabela curso normalizada segundo a FNBC, dando origem a tabela tutoria.

Relacionamento de cardinalidade 1:n, onde 1 aluno pode estar matriculado em várias disciplinas.
Verifica-se que na nova tabela (tutoria) o atributo disciplina é uma chave estrangeira referenciando a tabela curso. Ambas as tabelas estão na FNBC.

2.15.5 - Quarta Forma Normal (4FN)
Uma tabela está na 4FN se, e somente se, estiver na FNBC e não existirem dependências funcionais multivaloradas.
Vamos supor uma tabela livros:

- codLivro: código do livro;
- autor: autor do livro;
- titulo: título do livro;
- assunto: assunto abordado pelo livro;
- ano: ano de lançamento/edição;

Figura 20 – Tabela livro.

Podemos verificar que esta tabela possui dependência multivalorada em relação ao Autor e ao Assunto (um livro pode ter vários autores e apresentar muitos assuntos, e autores e assuntos não possuem vínculos entre si). Assim, para que esta tabela fique na 4FN devemos separá-la nas seguintes tabelas:
- livro: esta nova tabela foi obtida retirando os atributos multivalorados;
- assunto: esta tabela foi obtida através do atributo multivalorado “assunto” da tabela livro anterior;
- autor: esta tabela foi obtida através do atributo multivalorado “autor” da tabela livro anterior;

- autorLivro: esta tabela foi obtida, após retira o atributo multivalorado autor da tabela livro, criamos a tabela autor que será responsável por cadastrar todos os autores, porém, verificamos que os livros podem possuir mais de um autor. Logo trata-se de um relacionamento “muito para muitos”(m:n). Assim, precisamos de uma nova tabela que será a autorLivro.
- livroAssunto: esta tabela foi obtida, após retira o atributo multivalorado assunto da tabela livro, criamos a tabela assunto que será responsável por cadastrar todos os assuntos nos livros existentes, porém, verificamos que os livros podem possuir mais de um assunto. Logo se trata de um relacionamento “muito para muitos”(m:n). Assim, precisamos de uma nova tabela que será a livroAssunto.

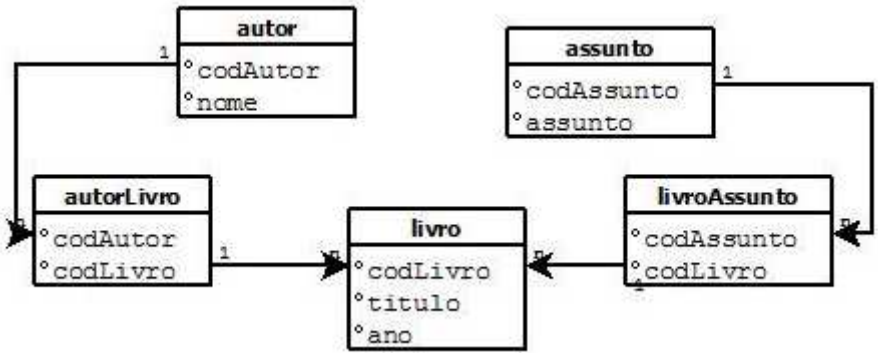


Figura 21 – Tabela livro na 4FN.

Exemplo de vários relacionamentos 1:n onde:
- 1 autor pode participar da elaboração de um ou mais livros;
- 1 assunto pode fazer parte de um ou mais livros e um livro pode tratar de um ou mais assuntos;

Desta forma, as dependências funcionais multivaloradas deixam de existir nestas tabelas. Portanto, as tabelas encontram-se na 4FN.

2.15.6 - Quinta Forma Normal (5FN) ou Forma Normal de Projeção de Junção (FNPJ)

Uma tabela está na 5FN se não existir dependência funcional cíclica. Para resolver este problema da dependência cíclica, devemos separar o ciclo, envolvendo relacionamentos com cardinalidade m:n;

Para exemplificar, vamos usar a tabela professor. Nesta tabela identificamos a seguinte dependência funcional cíclica:

professor-> disciplinas, disciplinas->apostilas, apostilas->professores;

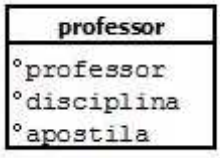


Figura 22 – Tabela professor.

Para resolver esta dependência funcional cíclica, podemos separar a tabela professor em 3 tabelas, professor, disciplina e apostila:

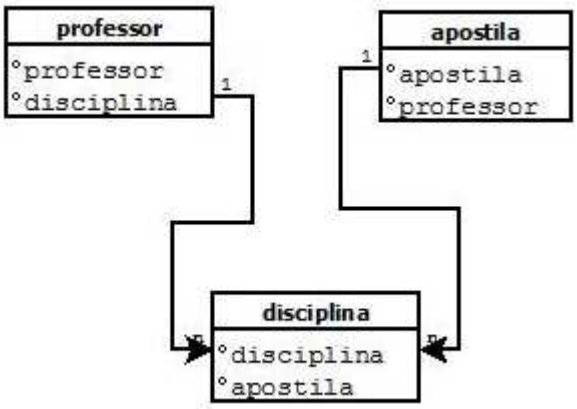


Figura 23 – Tabela professor na 5FN.

Relacionamentos 1:n onde:

-1 professor pode adotar 1 ou mais apostilas para determinada disciplina;

Desta forma, eliminamos a dependência funcional cíclica, logo, as tabelas encontram-se na 5FN.

2.16 - Desnormalizar para melhorar desempenho

Esquema de dados normalizados, quando colocados em produção, normalmente sofrem problemas de desempenho. Isso porque as regras de normalização visam evitar redundância e inconsistência dos dados e não melhorar desempenho no acesso aos dados.

Uma parte importante que deve ser usado com cautela, é a desnormalização de porções do esquema de dados para melhorar o desempenho de acesso aos dados.

Observando sempre que, se o projeto inicial e normalizado dos dados atinge o desempenho necessário, nada mais precisa ser feito. A desnormalização deve ser aplicada apenas quando os testes de desempenho mostram que temos um problema com os objetos, revelando que precisamos melhorar o tempo de acesso aos dados.

3 - Estudo de Caso: Modelagem de um sistema de controle de estoque

No estudo de caso que estaremos analisando, construiremos um sistema para o gerenciamento de estoque de um centro de distribuição de uma rede de lojas. As mercadorias compradas pela rede serão armazenadas nesse estoque central. A partir disso, à medida que as lojas precisarem das mercadorias, o pedido será feito para o estoque central.

Esse sistema poderá ser aplicado para controlar o estoque de uma rede de lojas em qualquer ramo. Para facilitar a exemplificação, iremos usar o exemplo de uma rede de lojas de ferramentas.

Para o gerenciamento do banco de dados, iremos utilizar o MySql como SGDBR. A ferramenta foi escolhida por se tratar de uma licença gratuita e robusta, tendo aplicações em uso até mesmo pela NASA, tendo competitividade com ferramentas pagas e já consolidadas no mercado.

3.1 - Descrição do sistema

Iremos nos basear nas necessidades do cliente e como todo o cliente, ele realizou algumas exigências:

- o sistema deverá ter a capacidade de armazenar os produtos contidos no estoque, para que esses possam ser controlados individualmente e armazenar a quantidade mínima que deverá ter este produto no estoque;
- cada produto terá um fornecedor relacionado a ele, sendo possível controlar os produtos divididos por fornecedores;
- os produtos poderão ser divididos por categorias, ou seja, cada produto terá uma categoria;
- as entradas e saídas dos produtos deverão ser registradas no programa, para futuramente obtermos um histórico completo de todo o trajeto do produto dentro do centro de distribuição;
- na entrada do produto será necessário armazenar a data do pedido e a data de entrega da mercadoria, para depois podermos analisar quanto tempo o pedido demora a chegar ao estoque;
- na saída, obrigatoriamente será informada a loja para qual a mercadoria foi enviada, pois no final do mês devemos fazer o fechamento do faturamento para saber qual a loja que mais obteve vendas;
- calcular o peso total de uma entrada ou de uma saída;
- no programa, deve-se apresentar os produtos nos quais a sua quantidade total em estoque é menor ou igual à quantidade mínima requerida em estoque definida previamente;

- a transportadora será outro item importante na análise, pois devemos saber qual transportadora é mais utilizada para fazer a entrega dos produtos e qual é a mais utilizada para fazer a saída;
- qual categoria possui mais item no local;

3.2 - Definindo as Entidades

Baseado no que o cliente solicitou, definimos as entidades:

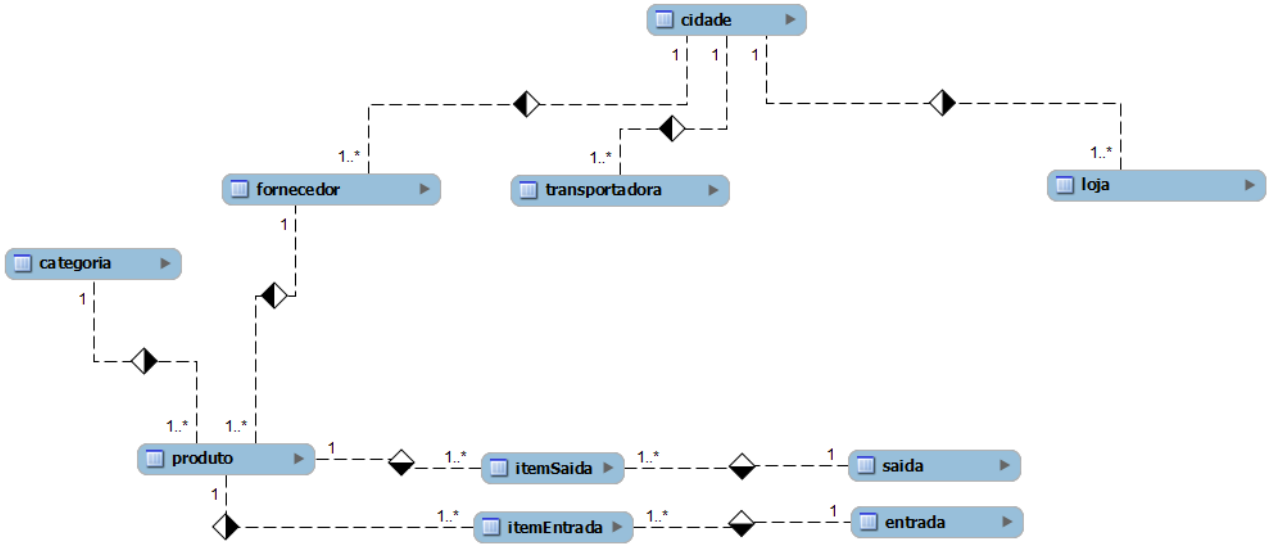


Figura 24 – Modelo Conceitual

O que percebemos neste modelo conceitual é que, embora ele já preencha em muito as exigências do cliente, e algumas normalizações já possam ser verificadas (uma entidade cidade e outra categoria para evitar redundâncias), ainda vemos alguns relacionamentos m:n (muito para muitos) e nosso estudo mostra que sempre temos este tipo de relacionamento, é necessário dividir as tabelas envolvidas com a criação de, pelo menos, mais uma tabela.

Pensando desta forma, temos o nosso modelo mais aproximado do ideal:

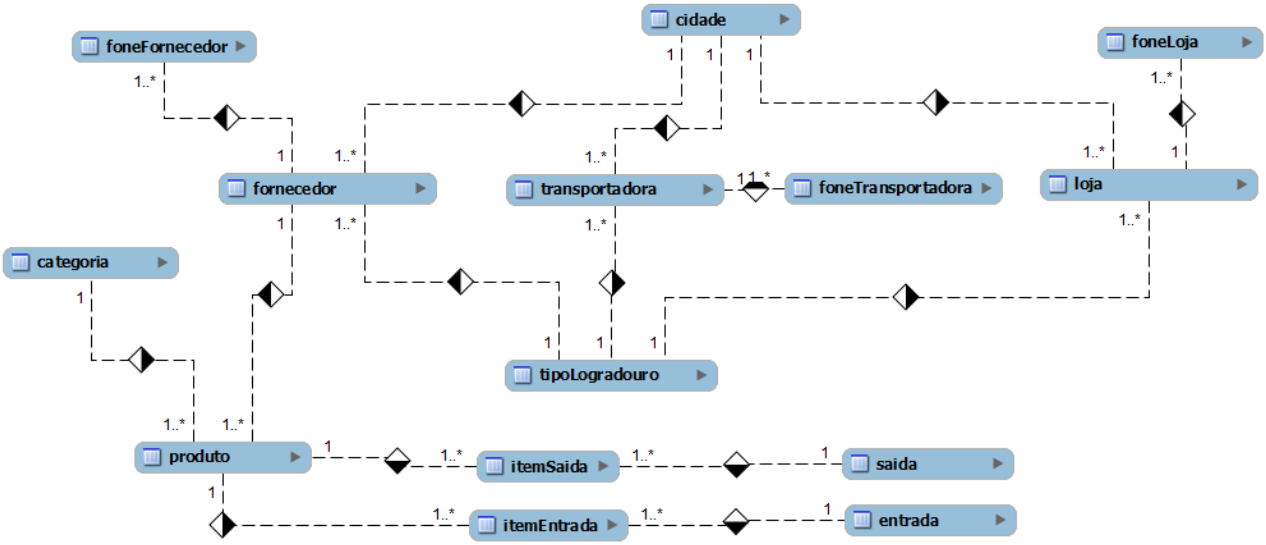


Figura 25 – Modelo Conceitual atualizado e normalizado.

Neste novo modelo temos os relacionamentos m:n eliminados e a criação de mais quatro entidades fracas*, itemEntrada, itemSaida, tipoLogradouro e foneLoja;

*Entidades fracas são entidades dependentes de outras entidades para existirem e sua existência sem elas se torna sem razão. Neste exemplo, não existe razão de existir a tabela itemEntrada se não houver uma tabela de entrada, mesmo fato ocorrendo com itemSaida.

3.3 - Detalhando as tabelas

No total, definimos 12 tabelas para compor o nosso banco de dados. As tabelas são:

- categoria: necessária para separar os produtos por categoria. Por exemplo, na categoria de perfumes, teremos a possibilidade de controlar apenas os produtos que fazem parte desta categoria. Os atributos a serem adicionados são: *codCategoria* e *categoria*;

- cidade: o depósito de dados de cidade terá um relacionamento com as tabelas que necessitam de endereço. Por questões de normalização, o campo cidade e uf foram colocados separados. Neste caso, a cidade estará sendo ligada às tabelas de *loja*, *fornecedor* e *transportadora*. Os campos armazenados serão *cidade* e *uf;*

- loja: o banco terá necessidade de armazenar para onde foram as saídas cadastradas no sistema, por isso a necessidade de criar a tabela de lojas. As lojas terão um relacionamento com as tabelas de *cidade* e *saída*. Os atributos desta entidade são: *codLoja*, *codCidade*, *nome*, *codTipoLogradouro*, *logradouro*, *num*, *bairro*, *insc*, *cnpj*, *cep*.

- transportadora: o estoque receberá os pedidos através de uma transportadora, a partir daí a necessidade de armazenar qual transportadora trouxe o produto e também qual transportadora

levou o produto até a loja. Os atributos desta entidade são: *codTransportadora, codCidade, transportadora, codTipoLogradouro, logradouro, num, bairro, CEP, CNPJ, insc.*

- fornecedor: para conseguirmos controlar os produtos de acordo com seu fornecedor foi criada a tabela de fornecedor. Os atributos desta entidade são: *codFornecedor, codCidade, fornecedor, codTipoLogradouro, logradouro, num, bairro, cep, insc.*

- produto: a tabela de produto terá informações pertinentes ao produtos que serão relacionados com as tabelas saída e entrada, dando a possibilidade de saber quais os produtos estão em uma determinada operação de entrada ou saída. O produto se relaciona com as tabelas *categoria* e *fornecedor*. Os atributos desta entidade são: *codProduto, codCategoria, codFornecedor, descricao, peso, qtdeMin.*

- entrada: nessa tabela serão armazenadas os dados pertinentes às entradas (chegada) de mercadorias até a central de distribuição, relacionando-se diretamente com as tabelas de *fornecedor* e *produto,* que por sua vez gera um relacionamento m:n, dando origem à tabela *itemEntrada*. Os atributos desta entidade são: *codEntrada, codTransportadora, dtPedido, dtEntrada, total, frete, nf, imposto.*

- itemEntrada:é originada pelo relacionamento m:n entre as tabelas *produto* e *entrada*. Ela armazenará cada produto que está contido na tabela de entrada, e suas respectivas informações tais como: *codItemEntrada, codProduto, codEntrada, lote, qtde, valor.*

- saída: na tabela saída teremos todos os dados relativos a saída (despacho) da mercadoria para as lojas. Ela terá relacionamentos com as tabelas *loja, transportadora,*e *produtos*, sendo que esta última é um relacionamento m:n, dando origem à tabela *itemSaida*. A tabela *saida* armazenará os seguintes atributos: *codSaida, codLoja, codTransportadora, total, frete, imposto.*

- *itemSaida*: o depósito de dados *itemSaida* é gerado pelo relacionamento m:n entre a tabela *saida* e *produto*. Essa tabela guardará os dados de cada produto que constar na saída. Os atributos a serem armazenados são: *codItemSaida, codSaida, codProduto, lote, qtde, valor.*

- *tipoLogradouro*: esta tabela irá conter os tipos de logradouro para preenchimento de endereço. Esta tabela é aconselhável por questão de normalização. Os atributos desta tabela são: *codTpLogradouro,tipoLogradouro.*

- *foneLoja:* esta tabela irá armazenar os números de telefone das lojas e também é aconselhável por questão de padronização. Os atributos que irão compor esta tabela são: *codLoja, codTelefone, ddd, numero, ramal, setor, contato.*

- *foneFornecedor:* esta tabela irá armazenar os números de telefone dos fornecedores e também é aconselhável por questão de padronização. Os atributos que irão compor esta tabela são: *codFornecedor, codTelefone, ddd, numero, ramal, setor, contato.*
- *foneTransportadora:* esta tabela irá armazenar os números de telefone dos fornecedores e também é aconselhável por questão de padronização. Os atributos que irão compor esta tabela são: *codTransportadora, codTelefone, ddd, numero, ramal, setor, contato.*

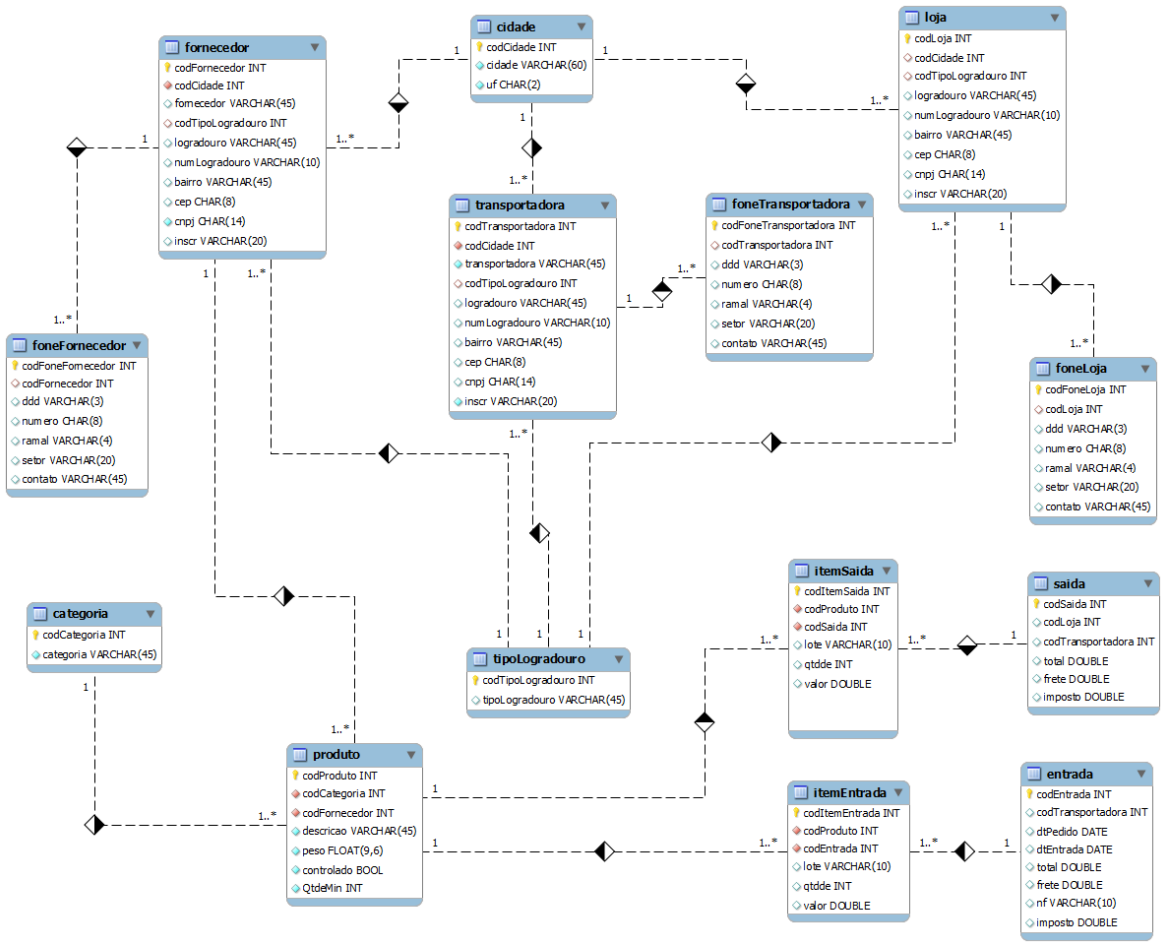


Figura 26 – Modelo Físico de dados.

Ao visualizar o modelo físico, notamos que nas tabelas não existe redundância de informação (dados repetidos em mais de uma tabela que não sejam campos chaves) e ao mesmo tempo, todos os dados necessários para o sistema se encontram armazenados na base de dados, constatando assim que o banco de dados está bem modelado.

Para efetuar a extração dos dados usamos a SQL (Structure Query Language – Linguagem estruturada de consulta).

4 - SQL - Structure Query Language

Para acesso a um banco de dados relacional, é necessário uma linguagem. A SQL é uma linguagem usada pela maioria dos bancos de dados relacionais.

É uma linguagem baseada no inglês, fácil de escrever, ler e entender.

SQL tem como características, a economia de tempo, flexibilidade e segurança na manutenção de bancos de dados.

Existem pequenas diferenças da linguagem, em alguns bancos.

No VisualFox por exemplo, quando o comando ocupa mais de uma linha, devemos colocar ponto e vírgula no final de cada linha, e no final do comando somente um enter.

No MySql, ao contrário, devemos colocar um ponto e vírgula somente no final do comando, mesmo que este ocupe mais de uma linha.

Estou usando a nomenclatura do MySql, e criamos o Banco de Dados conforme o modelo físico, nomeando-o como “estoqueDb”, iremos fazer todas as operações de inserção (clausulas Insert), seleção (select), atualização (update) e exclusão (delete) demonstrando a funcionalidade do exemplo criado.

A SQL divide-se em:

DDL - Linguagem de definição de dados (Data Definition Language)

É um conjunto de comandos dentro da SQL usada para a definição das estruturas de dados, fornecendo as instruções que permitem a criação, modificação e remoção das tabelas, assim como criação de índices. Estas instruções SQL permitem definir a estrutura de uma base de dados, incluindo as linhas, colunas, tabelas, índices, e outros metadados.

Entre os principais comandos DDL estão CREATE (Criar), DROP (deletar) e ALTER (alterar).

DML - Linguagem de Manipulação de Dados

Linguagem de manipulação de dados (ou DML, de Data Manipulation Language) é o grupo de comandos dentro da linguagem SQL utilizado para a recuperação, inclusão, remoção e modificação de informações em bancos de dados.

Os principais comandos DML são SELECT (Seleção de Dados), INSERT (Inserção de Dados), UPDATE (Atualização de Dados) e DELETE (Exclusão de Dados).

DCL - Linguagem de Controle de Dados

Linguagem de controle de dados (ou DCL, de Data Control Language) é o grupo de comandos que permitem ao administrador de banco de dados controlar o acesso aos dados deste banco. Alguns exemplos de comandos DCL são:

GRANT: Permite dar permissões a um ou mais usuários e determinar as regras para tarefas determinadas;

REVOKE: Revoga permissões dadas por um GRANT.

As tarefas básicas que podemos conceder ou barrar permissões são:

CONNECT

SELECT

INSERT

UPDATE

DELETE

USAGE

4.1 – Imagens das tabelas do DB estoqueDb preenchidas para a realização de testes:

codProduto	codCategoria	codFornecedor	descricao	peso	controlado	QtdeMin
1	1	1	MARTELO 2"	1.000000	0	12
2	1	1	ALICATE UNIVERSAL	0.500000	0	12
3	1	1	ALICATE DE CORTE	0.500000	0	12
4	1	1	ALICATE DE BICO 6"	0.500000	0	12
5	1	1	ALICATE DE PRESSÃO	0.700000	0	6
6	1	1	ALICATE DE BOMBA	0.500000	0	6
7	1	1	ALICATE GRIFO 8"	1.500000	0	3
8	1	1	SERROTE CARPINTEIRO 26"	1.000000	0	3
9	1	1	SERROTE CARPINTEIRO 22"	1.000000	0	4
10	1	1	ARCO DE SERRA COMPLETO	1.200000	0	12
11	2	1	FURADEIRA ELÉTRICA 3/8	2.000000	0	10
12	2	1	FURADEIRA ELÉTRICA 1/2	2.000000	0	10
13	2	1	SERRA TICO TICO	2.000000	0	5
14	2	1	LIXADEIRA ELÉTRICA	2.000000	0	3
15	2	1	PLAINA	2.000000	0	2
16	1	1	BROCA DE AÇO RÁPIDO 1/8	0.050000	0	30
17	1	1	BROCA DE AÇO RÁPIDO 5/32	0.050000	0	30
18	1	1	BROCA DE AÇO RÁPIDO 1/4	0.075000	0	30
19	1	1	BROCA DE AÇO RÁPIDO 3/16	0.050000	0	30
20	1	2	MARTELO 2"	1.000000	0	12
21	1	2	ALICATE UNIVERSAL	0.500000	0	12
22	1	2	ALICATE DE CORTE	0.500000	0	12
23	1	2	ALICATE DE BICO 6"	0.500000	0	12
24	1	2	ALICATE DE PRESSÃO	0.700000	0	6
25	1	2	ALICATE DE BOMBA	0.500000	0	6
26	1	2	ALICATE GRIFO 8"	1.500000	0	3
27	1	2	SERROTE CARPINTEIRO 26"	1.000000	0	3
28	1	2	SERROTE CARPINTEIRO 22"	1.000000	0	4
29	1	2	ARCO DE SERRA COMPLETO	1.200000	0	12
30	2	2	FURADEIRA ELÉTRICA 3/8	2.000000	0	10
31	2	2	FURADEIRA ELÉTRICA 1/2	2.000000	0	10
32	2	2	SERRA TICO TICO	2.000000	0	5

Figura 27 – Tabela produto preenchida

	codLoja	codCidade	codTipoLogradouro	logradouro	numLogradouro	bairro	cep	cnpj	inscr
	1	1	1	FLORENCIO DE ABREU	111	CENTRO	01315000	12365478000112	12345678963
	2	2	2	TEXTEIS	112	PIMENTAS	01478030	96385214000162	96325874123
*	(Auto)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Figura 28 – Tabela loja preenchida com dois registros

	codFornecedor	codCidade	fornecedor	codTipoLogradouro	logradouro	numLogradouro	bairro	cep	contato	cnpj	inscr
	1	1	STANLEY	1	DAS FERRAMENTAS	111	CENTRO	01385030	MARCOS	12345678000165	123654789
	2	2	GEDORE	2	DAS FERRAGENS	222	PIMENTAS	01413020	JULIO	32165498000199	321654987
*	(Auto)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Figura 29 – Tabela fornecedor preenchida com dois registros

	codFoneFornecedor	codFornecedor	ddd	numero	ramal	setor	contato
	1	1	11	12345678	123	COMPRAS	DINEI
	2	1	11	23456789	124	FINANCEIR	RONALDO
	3	2	11	98765432	321	COMPRAS	NETO
	4	2	11	87654321	421	FINANCEIR	SOCRATES
*	(Auto)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Figura 30 – Tabela foneFornecedor preenchida

	codFoneLoja	codLoja	ddd	numero	ramal	setor	contato
	1	1	11	36549871	0	VENDAS	DANIEL
	2	1	11	45623897	0	GERENCIA	GABRIEL
	3	2	11	56231478	0	VENDAS	ANDRE
	4	2	11	63254136	0	FINANCEIR	GILBERTO
*	(Auto)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Figura 31 – Tabela foneLoja preenchida

	codFoneTransportadora	codTransportadora	ddd	numero	ramal	setor	contato
	1	1	11	69325874	0	ENTREGAS	SIDNEY
	2	1	11	45632178	0	CONTRATOS	MARCOS
	3	2	11	85236974	0	ENTREGAS	JOSE
	4	2	11	78965412	0	CONTRATOS	MARCIO
*	(Auto)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Figura 32 – Tabela foneTransportadora preenchida

	codTipoLogradouro	tipoLogradouro
	1	RUA
	2	AVENIDA
	3	ALAMEDA
	4	TRAVESSA
	5	VIA
	6	VIADUTO
*	(Auto)	(NULL)

Figura 33 – Tabela tipoLogradouro preenchida

	codCategoria	categoria
	1	Manual
	2	Elétrica
*	(Auto)	(NULL)

Figura 34 – Tabela categoria preenchida

	codCidade	cidade	uf
	1	São Paulo	SP
	2	Campinas	SP
	3	Osasco	SP
	4	Guarulhos	SP
*	(Auto)	(NULL)	(NULL)

Figura 35 – Tabela cidade preenchida

	codEntrada	codTransportadora	dtPedido	dtEntrada	total	frete	nf	imposto
	1	1	2011-09-20	2011-09-22	2300	200	005698	130
	2	1	2011-10-01	2011-10-03	4500	250	005785	150
	3	2	2011-09-25	2011-10-01	9000	350	009871	200
	4	2	2011-09-27	2011-10-01	8000	300	009899	250
	5	2	2011-09-29	2011-10-02	12000	500	009921	380
	6	2	2011-09-29	2011-10-01	13000	450	009922	350
*	(Auto)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Figura 36 – Tabela entrada preenchida

	codItemEntrada	codProduto	codEntrada	lote	qtdde	valor
	1	1	1	1	10	300
	2	2	1	1	10	350
	3	3	1	1	10	350
	4	4	1	1	10	500
	5	5	1	1	10	500
	6	6	1	1	3	300
	7	1	2	1	20	600
	8	2	2	1	20	700
	9	3	2	1	20	700
	10	4	2	1	20	1000
	11	5	2	1	20	1000
	12	6	2	1	5	500
	13	1	3	1	40	1200
	14	2	3	1	40	1400
	15	3	3	1	40	1400
	16	4	3	1	40	2000
	17	5	3	1	40	2000
	18	6	3	1	10	1000
*	(Auto)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Figura 37 – Tabela itemEntrada preenchida

codSaida	codLoja	codTransportadora	total	frete	imposto
1	1	1	2600	100	80
2	1	1	3000	120	100
3	1	1	6000	200	180
4	1	1	10000	350	250
5	1	2	12000	400	350
6	2	1	8000	350	260
7	2	1	6000	300	220
8	2	2	12000	380	350
9	2	2	18000	450	420
(Auto)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Figura 38 – Tabela saída preenchida

codItemSaida	codProduto	codSaida	lote	qtdde	valor
1	1	1	1	10	300
2	2	1	1	10	350
3	3	1	1	10	350
4	4	1	1	10	500
5	5	1	1	10	500
6	6	1	1	3	300
7	7	1	1	3	300
8	1	2	1	10	300
9	2	2	1	10	350
10	3	2	1	10	350
11	4	2	1	10	500
12	5	2	1	10	500
13	6	2	1	5	500
14	7	2	1	5	500
(Auto)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Figura 39 – Tabela itemSaida preenchida

Perceba que na massa de dados armazenados, os campos chaves possuem apenas os códigos das tabelas relacionadas. Em uma consulta de seleção, alteração, atualização ou exclusão nos referimos a estes campos para realizá-la e mais a frente, iremos exibir como.

4.2 – Script de criação do DB estoqueDb para SGDBR MySql

```
CREATE SCHEMA IF NOT EXISTS estoqueDb DEFAULT CHARACTER SET latin1 COLLATE latin1_swedish_ci ;
USE estoqueDb ;

-- -----------------------------------------------------
-- Tabela estoqueDb.categoria
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS estoqueDb.categoria (
  codCategoria INT NOT NULL AUTO_INCREMENT ,
  categoria VARCHAR(45) NOT NULL ,
  PRIMARY KEY (codCategoria) )
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Tabela estoqueDb.cidade
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS estoqueDb.cidade (
  codCidade INT NOT NULL AUTO_INCREMENT ,
  cidade VARCHAR(60) NOT NULL ,
  uf CHAR(2) NOT NULL ,
  PRIMARY KEY (codCidade) )
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Tabela estoqueDb.tipoLogradouro
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS estoqueDb.tipoLogradouro (
  codTipoLogradouro INT NOT NULL AUTO_INCREMENT ,
  tipoLogradouro VARCHAR(45) NULL ,
  PRIMARY KEY (codTipoLogradouro) )
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Tabela estoqueDb.fornecedor
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS estoqueDb.fornecedor (
  codFornecedor INT NOT NULL AUTO_INCREMENT ,
  codCidade INT NOT NULL ,
  fornecedor VARCHAR(45) NULL ,
  codTipoLogradouro INT NULL ,
  logradouro VARCHAR(45) NULL ,
  numLogradouro VARCHAR(10) NULL ,
  bairro VARCHAR(45) NULL ,
  cep CHAR(8) NULL ,
  contato VARCHAR(45) NULL ,
  cnpj CHAR(14) NOT NULL ,
  inscr VARCHAR(20) NULL ,
  PRIMARY KEY (codFornecedor) ,
  INDEX fk_cidade_fornecedor (codCidade ASC) ,
  INDEX fk_tipoLogradouro_fornecedor (codTipoLogradouro ASC) ,
  CONSTRAINT fk_cidade_fornecedor
    FOREIGN KEY (codCidade )
    REFERENCES estoqueDb.cidade (codCidade )
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT fk_tipoLogradouro_fornecedor
    FOREIGN KEY (codTipoLogradouro )
    REFERENCES estoqueDb.tipoLogradouro (codTipoLogradouro )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Tabela estoqueDb.produto
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS estoqueDb.produto (
  codProduto INT NOT NULL AUTO_INCREMENT ,
  codCategoria INT NOT NULL ,
  codFornecedor INT NOT NULL ,
  descricao VARCHAR(45) NOT NULL ,
  peso FLOAT(9,6) NOT NULL ,
  controlado TINYINT(1)  NOT NULL DEFAULT false ,
  QtdeMin INT NOT NULL ,
  PRIMARY KEY (codProduto) ,
  INDEX fk_produto_fornecedor (codFornecedor ASC) ,
  INDEX fk_produto_categoria (codCategoria ASC) ,
  CONSTRAINT fk_produto_fornecedor
    FOREIGN KEY (codFornecedor )
    REFERENCES estoqueDb.fornecedor (codFornecedor )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_produto_categoria
    FOREIGN KEY (codCategoria )
    REFERENCES estoqueDb.categoria (codCategoria )
    ON DELETE RESTRICT
    ON UPDATE CASCADE)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Tabela estoqueDb.entrada
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS estoqueDb.entrada (
  codEntrada INT NOT NULL AUTO_INCREMENT ,
  codTransportadora INT NULL ,
  dtPedido DATE NULL ,
  dtEntrada DATE NULL ,
  total DOUBLE NULL ,
  frete DOUBLE NULL ,
  nf VARCHAR(10) NULL ,
  imposto DOUBLE NULL ,
  PRIMARY KEY (codEntrada) )
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Tabela estoqueDb.transportadora
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS estoqueDb.transportadora (
  codTransportadora INT NOT NULL AUTO_INCREMENT ,
  codCidade INT NOT NULL ,
  transportadora VARCHAR(45) NOT NULL ,
  codTipoLogradouro INT NULL ,
  logradouro VARCHAR(45) NULL ,
  numLogradouro VARCHAR(10) NULL ,
  bairro VARCHAR(45) NULL ,
  cep CHAR(8) NULL ,
  cnpj CHAR(14) NULL ,
  inscr VARCHAR(20) NOT NULL ,
  PRIMARY KEY (codTransportadora) ,
  INDEX fk_cidade_transportador (codCidade ASC) ,
  INDEX fk_transportadora_tipoLogradouro (codTipoLogradouro ASC) ,
  CONSTRAINT fk_cidade_transportador
    FOREIGN KEY (codCidade )
    REFERENCES estoqueDb.cidade (codCidade )
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT fk_transportadora_tipoLogradouro
    FOREIGN KEY (codTipoLogradouro )
    REFERENCES estoqueDb.tipoLogradouro (codTipoLogradouro )
    ON DELETE RESTRICT
    ON UPDATE CASCADE)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Tabela estoqueDb.itemEntrada
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS estoqueDb.itemEntrada (
  codItemEntrada INT NOT NULL AUTO_INCREMENT ,
  codProduto INT NOT NULL ,
  codEntrada INT NOT NULL ,
  lote VARCHAR(10) NULL ,
  qtdde INT NULL ,
  valor DOUBLE NULL ,
  PRIMARY KEY (codItemEntrada) ,
  INDEX fk_itemEntrada_produto (codProduto ASC) ,
  INDEX fk_itemEntrada_entrada (codEntrada ASC) ,
  CONSTRAINT fk_itemEntrada_produto
    FOREIGN KEY (codProduto )
    REFERENCES estoqueDb.produto (codProduto )
    ON DELETE NO ACTION     ON UPDATE CASCADE,
  CONSTRAINT fk_itemEntrada_entrada     FOREIGN KEY (codEntrada )
    REFERENCES estoqueDb.entrada (codEntrada )
    ON DELETE CASCADE     ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Tabela estoqueDb.saida
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS estoqueDb.saida (
  codSaida INT NOT NULL AUTO_INCREMENT ,
  codLoja INT NULL ,
  codTransportadora INT NULL ,
  total DOUBLE NULL ,
  frete DOUBLE NULL ,
  imposto DOUBLE NULL ,
  PRIMARY KEY (codSaida) )
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Tabela estoqueDb.itemSaida
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS estoqueDb.itemSaida (
  codItemSaida INT NOT NULL AUTO_INCREMENT ,
  codProduto INT NOT NULL ,
  codSaida INT NOT NULL ,
  lote VARCHAR(10) NULL ,
  qtdde INT NULL ,
  valor DOUBLE NULL ,
  PRIMARY KEY (codItemSaida) ,
  INDEX fk_itemSaida_produto (codProduto ASC) ,
  INDEX fk_itemSaida_saida (codSaida ASC) ,
  CONSTRAINT fk_itemSaida_produto
    FOREIGN KEY (codProduto )
    REFERENCES estoqueDb.produto (codProduto )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT fk_itemSaida_saida
    FOREIGN KEY (codSaida )
    REFERENCES estoqueDb.saida (codSaida )
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Tabela estoqueDb.loja
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS estoqueDb.loja (
  codLoja INT NOT NULL AUTO_INCREMENT ,
  codCidade INT NULL ,
  codTipoLogradouro INT NULL ,
  logradouro VARCHAR(45) NULL ,
  numLogradouro VARCHAR(10) NULL ,
  bairro VARCHAR(45) NULL ,
  cep CHAR(8) NULL ,
  cnpj CHAR(14) NULL ,
  inscr VARCHAR(20) NULL ,
  PRIMARY KEY (codLoja) ,
  INDEX fk_loja_cidade (codCidade ASC) ,
  INDEX fk_loja_tipoLogradouro (codTipoLogradouro ASC) ,
  CONSTRAINT fk_loja_cidade
    FOREIGN KEY (codCidade )
    REFERENCES estoqueDb.cidade (codCidade )
    ON DELETE RESTRICT
    ON UPDATE CASCADE,
  CONSTRAINT fk_loja_tipoLogradouro
    FOREIGN KEY (codTipoLogradouro )
    REFERENCES estoqueDb.tipoLogradouro (codTipoLogradouro )
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Tabela estoqueDb.foneLoja
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS estoqueDb.foneLoja (
  codFoneLoja INT NOT NULL AUTO_INCREMENT ,
  codLoja INT NULL ,
  ddd VARCHAR(3) NULL ,
  numero CHAR(8) NULL ,
  ramal VARCHAR(4) NULL ,
  setor VARCHAR(20) NULL ,
  contato VARCHAR(45) NULL ,
  PRIMARY KEY (codFoneLoja) ,
  INDEX fk_foneLoja_loja (codLoja ASC) ,
  CONSTRAINT fk_foneLoja_loja
    FOREIGN KEY (codLoja )
    REFERENCES estoqueDb.loja (codLoja )
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Tabela estoqueDb.foneTransportadora
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS estoqueDb.foneTransportadora (
  codFoneTransportadora INT NOT NULL AUTO_INCREMENT ,
  codTransportadora INT NULL ,
  ddd VARCHAR(3) NULL ,
  numero CHAR(8) NULL ,
  ramal VARCHAR(4) NULL ,
  setor VARCHAR(20) NULL ,
  contato VARCHAR(45) NULL ,
  PRIMARY KEY (codFoneTransportadora) ,
  INDEX fk_foneTransportadora_transportadora (codTransportadora ASC) ,
  CONSTRAINT fk_foneTransportadora_transportadora
    FOREIGN KEY (codTransportadora )
    REFERENCES estoqueDb.transportadora (codTransportadora )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Tabela estoqueDb.foneFornecedor
-- -----------------------------------------------------
CREATE  TABLE IF NOT EXISTS estoqueDb.foneFornecedor (
  codFoneFornecedor INT NOT NULL AUTO_INCREMENT ,
  codFornecedor INT NULL ,
  ddd VARCHAR(3) NULL ,
  numero CHAR(8) NULL ,
  ramal VARCHAR(4) NULL ,
  setor VARCHAR(20) NULL ,
  contato VARCHAR(45) NULL ,
  PRIMARY KEY (codFoneFornecedor) ,
  INDEX fk_foneTransportadora_transportadora (codFornecedor ASC) ,
  CONSTRAINT fk_foneFornecedor_fornecedor
    FOREIGN KEY (codFornecedor )
    REFERENCES estoqueDb.fornecedor (codFornecedor )
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

4.3 – Consulta seleção (SELECT)

A estrutura básica de uma expressão SQL consiste em três clausulas: SELECT, FROM e WHERE. Sendo:

SELECT é equivalente a projeção da álgebra relacional, relacionando os atributos resultantes da consulta.

FROM é equivalente ao produto cartesiano da álgebra relacional. Indica a origem de onde será feita a consulta.

WHERE é equivalente a seleção do predicado da álgebra relacional. Tem a função de restringir a uma ou mais condições.

Ex.

SELECT At1 FROM R1 WHERE P1.

O exemplo demonstra uma estrutura básica de uma consulta SQL, onde At1 é um atributo, R1 uma relação e P1 um predicado. Observando que para cada uma das variantes pode ser aplicado n valores.

A clausula WHERE pode ser omitida, sendo assim o predicado sempre se tornará verdadeiro.

Para exemplificar vamos realizar uma consulta para que seja exibido todos os registros de lojas cadastradas, e uma outra com os registros de telefones de uma determinada loja.

Consulta das lojas constantes no DB estoqueDb exibindo os dados de forma interativa com o operador.

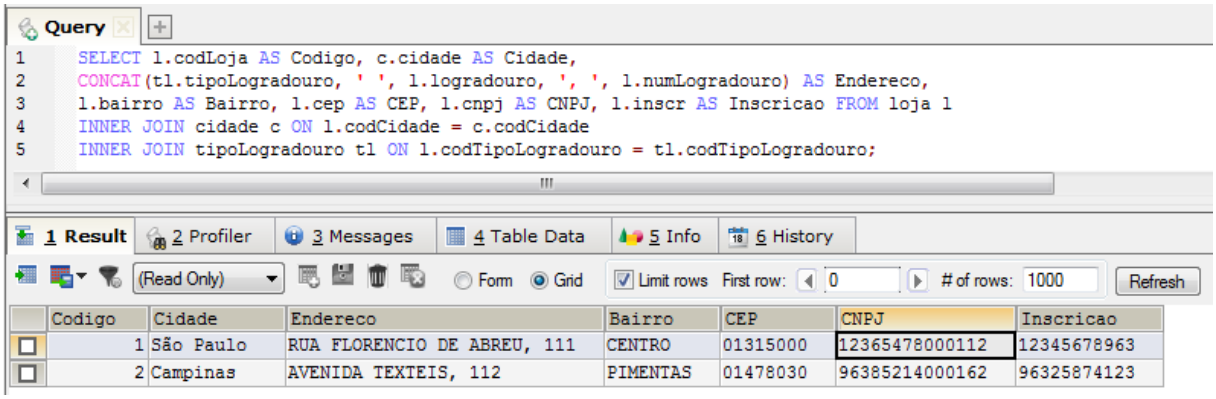


Figura 40 – Consulta usando JOIN para selecionar os campos em tabelas relacionadas.

Percebemos na consulta seleção exibida na Figura 40 que, além da clausula SELECT * FROM, usamos a função CONCAT para exibir o endereço das lojas em um único campo além da função INNER JOIN que é uma junção entre tabelas relacionadas usando os campos chaves como parâmetro.

“FROM loja l INNER JOIN cidade c ON l.codCidade = c.codCidade”

Nesta linha informamos que queremos mostrar os campos da tabela loja com os campos da tabela cidade desde que os campos chaves codCidade de ambas as tabelas sejam iguais.

Percebemos então que a SQL nos fornece recursos de exibição de informação de dados sem que necessitemos repeti-los em várias tabelas. Necessitamos apenas informar os campos nas suas respectivas tabelas

Com isso, termos:

- a informação em um único local, tornando o trabalho mais fácil de administrar;
- não ocupamos espaço em disco com redundância de informação;
- não ocupamos desempenho da máquina em administrar colunas redundantes;
- não corremos o risco de ter que alterar a mesma informação em vários locais e acidentalmente esquecer de alterar algum local tornando a mesma informação conflitante em locais (tabelas) diferentes (inconsistência).

Consulta de seleção usando a clausula WHERE

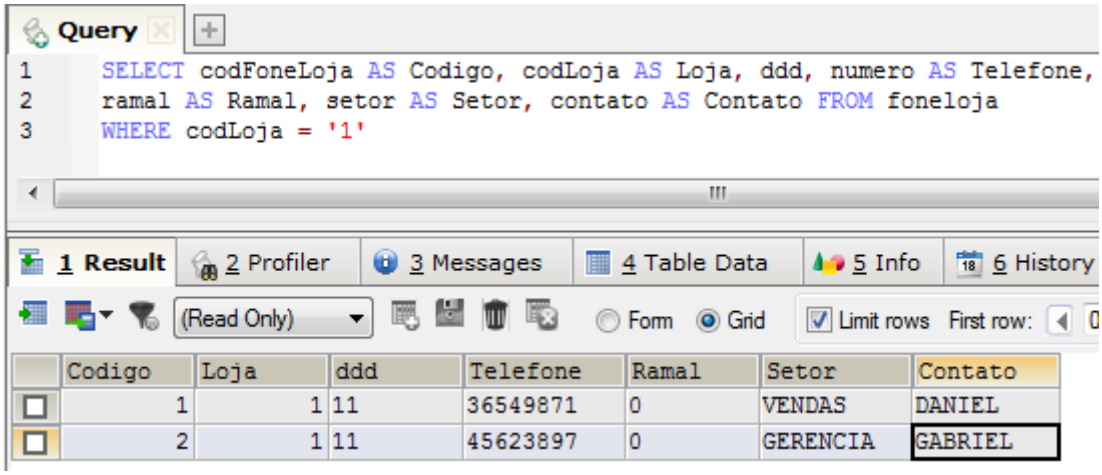


Figura 41 - – Consulta de telefone das lojas cadastradas usando a clausula WHERE

Percebemos na consulta seleção exibida na Figura 41 o uso da clausula WHERE, onde através de um parâmetro escolhemos o tipo de registro que queremos que seja exibido.

No caso em específico, escolhemos exibir os números de telefone da loja de código “1”.

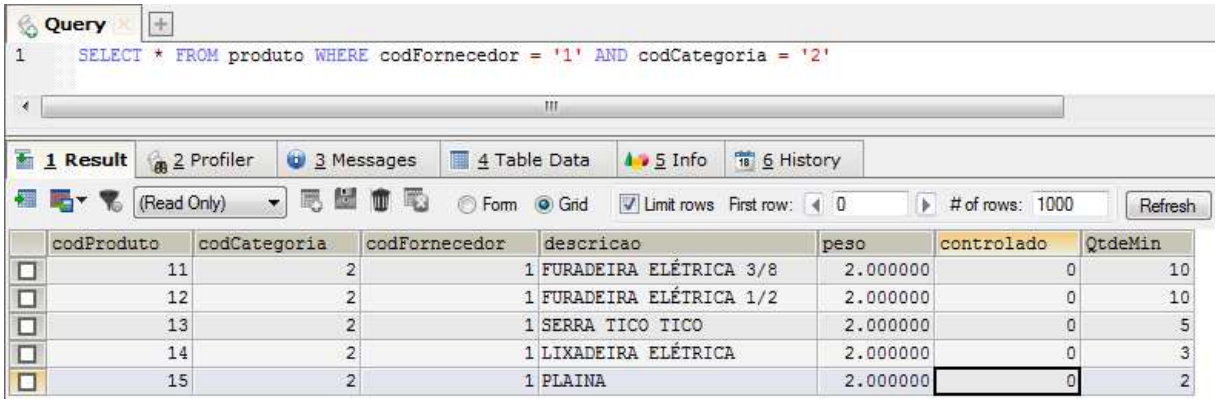


Figura 42 – Consulta de produtos da categoria “2”(Elétrica) do fornecedor “1” (Stanley).

Na consulta seleção da Figura 42 percebemos o uso do caractere “*”(asteristico). Usamos este caractere quando queremos trazer todos os atributos de uma tabela da forma que eles se encontram armazenados na mesma.

Percebemos ainda o uso do operador lógico AND (e) para aprimorar o filtro concatenando mais de um parâmetro.

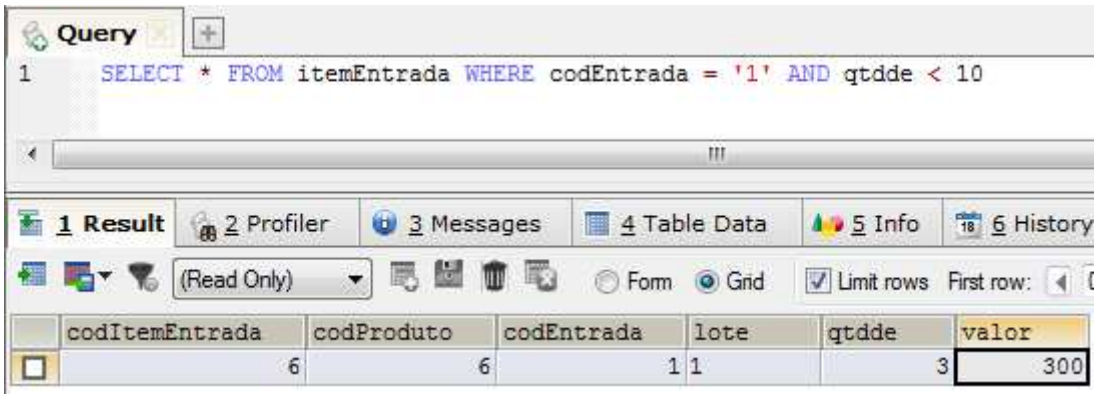


Figura 43 – Consulta seleção exibindo os registros da tabela itemEntrada

Percebemos na consulta seleção exibida na Figura 43 que, além do operador “*” e AND usamos o operador matemático “<” (menor que) .

4.4 – Inserção de dados (INSERT)

O comando para inclusão de dados é o INSERT, que possui a seguinte estrutura:

INSERT INTO nome_tabela (lista-de-campos) VALUES (lista_dados)

OU

INSERT INTO nome_tabela VALUES (lista_dados)

Onde:

Nome_tabela: nome da tabela no qual será inserido os dados.

Lista-de-campos: nome das colunas que receberão os valores.

Lista-dados: valores que serão inseridos na tabela. Estes campos devem estar na mesma ordem descrita em lista-de-campos, todos separados por vírgula.

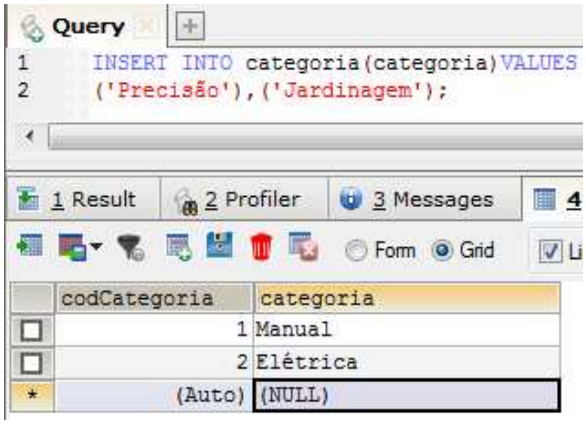


Figura 44 – Tabela categoria antes da inserção de duas novas categorias

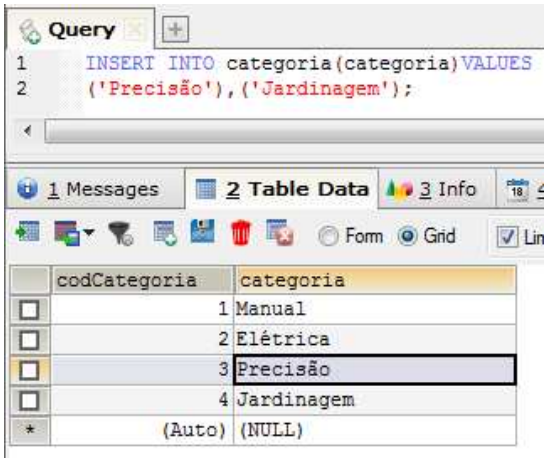


Figura 45 – Tabela categoria após inserção de duas novas categorias

No caso específico dos atributos de tipo INT com Auto-Numeração não precisamos (nem devemos) nos preocupar em preenchê-los, isto é tarefa do SGDB. Logo, precisamos preencher os outros campos. No caso específico da tabela “categoria” das Figuras 44 e 45 precisamos apenas informar o atributo “categoria” já que o atributo codCategoria o SGDB se encarrega.

4.5 – Atualização de dados (UPDATE)

O comando para atualizar registros é UPDATE, que tem a seguinte sintaxe:

UPDATE nome_tabela

SET CAMPO = "novo_valor"

WHERE CONDIÇÃO

Onde:

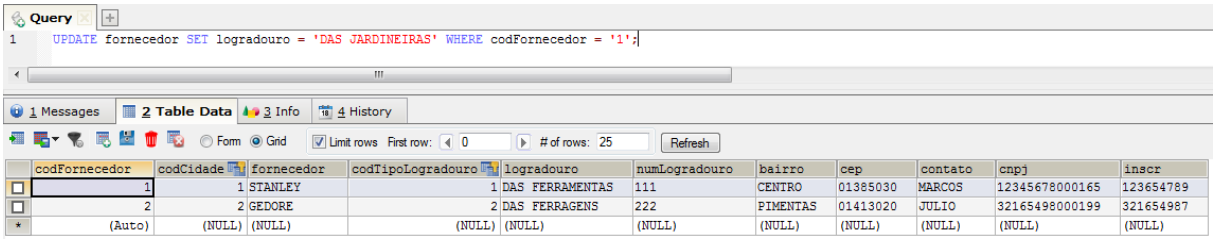
Nome_tabela: nome da tabela que será modificada

Campo: campo que terá seu valor alterado

Novo_valor: valor que substituirá o antigo dado cadastrado em campo

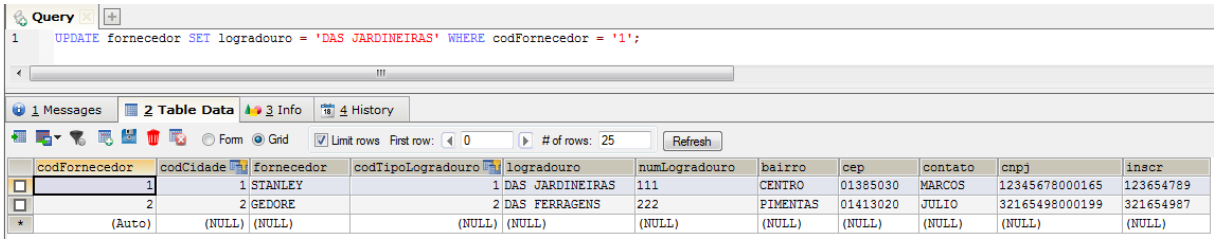
Where: Se não for informado, a tabela intera será atualizada

Condição: regra que impõe condição para execução do comando



codFornecedor	codCidade	fornecedor	codTipoLogradouro	logradouro	numLogradouro	bairro	cep	contato	cnpj	inscr
1	1	STANLEY	1	DAS FERRAMENTAS	111	CENTRO	01385030	MARCOS	12345678000165	123654789
2	2	GEDORE	2	DAS FERRAGENS	222	PIMENTAS	01413020	JULIO	32165498000199	321654987
(Auto)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Figura 46 – Tabela fornecedor antes de atualizar campo logradouro do fornecedor.



codFornecedor	codCidade	fornecedor	codTipoLogradouro	logradouro	numLogradouro	bairro	cep	contato	cnpj	inscr
1	1	STANLEY	1	DAS JARDINEIRAS	111	CENTRO	01385030	MARCOS	12345678000165	123654789
2	2	GEDORE	2	DAS FERRAGENS	222	PIMENTAS	01413020	JULIO	32165498000199	321654987
(Auto)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Figura 47 – Tabela fornecedor após atualizar campo logradouro do fornecedor.

Importante salientar que no caso de UPDATE, quase sempre é necessário o uso da clausula WHERE porque caso contrário, todos os registros da tabela mencionada serão atualizados. No caso exibido nas Figuras 46 e 47 se não houvesse o complemento “WHERE codFornecedor = ‘1’” todos os registros teriam o atributo “logradouro” alterado para “DAS JARDINEIRAS”.

4.6 – Exclusão de dados (DELETE)

O comando utilizado para apagar dados é o DELETE

DELETE FROM nome_tabela

WHERE condição

Onde:

Nome_tabela: nome da tabela que será modificada

Where: cláusula que impõe uma condição sobre a execução do comando.

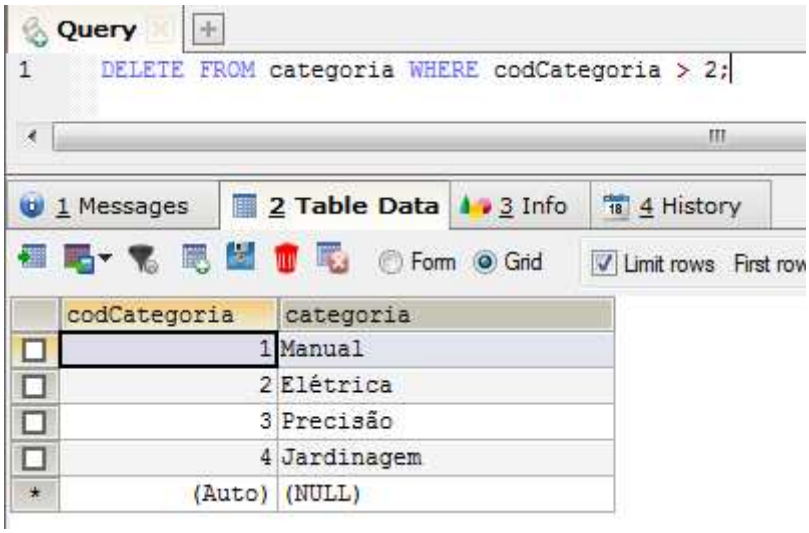


Figura 48 – Tabela categoria antes da exclusão de registros.

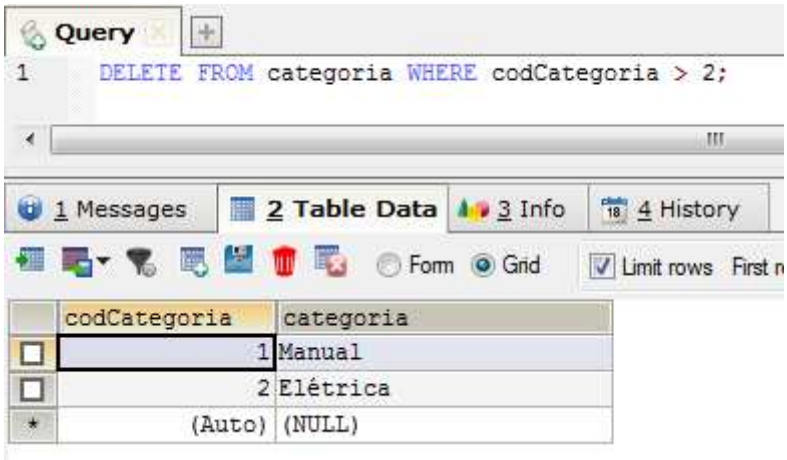


Figura 49 – Tabela categoria após exclusão de registros.

Conclusão

Podemos perceber que a adoção da normalização de dados é importante para que o projeto de um banco de dados não possua redundância e consequentemente inconsistência. Assim, devemos verificar sempre todos os conceitos abordados neste trabalho, com a finalidade de gerar melhores modelos de dados. Consequentemente, os sistemas irão absorver todos estes benefícios, principalmente, gerando melhores informações para tomadas de decisões, por exemplo.

Vale ressaltar que a normalização não pode gerar perdas no poder de extração de informações a partir de banco de dados. Caso isto ocorra, em muitos casos pode ser interessante o processo de desnormalização para melhorar o desempenho das consultas, por exemplo.

Entretanto, este custo pode ser muito alto, comprometendo a garantia de consistência dos dados.

Bibliografia

MACHADO, Felipe. ABREU, Mauricio. Projeto de Banco de dados – Uma visão prática. São Paulo: Érica, 1996

MySql, a Bíblia / Steve Suehring – Rio de Janeiro: Elsevier, 2002 – 2ª Reimpressão

ELMASRI, Ramez. Sistemas de Banco de Dados/Ramez Elmasri e Shamkat B. Navathe - São Paulo: Pearson Addison Wesley, 2005

JUNIOR, Ary.”Normalização de Dados”. SQL Magazine. Brasil: Edição 47 – Ano 4. Páginas: 22 a 35;

ROSSI, Diego.”Modelagem de um sistema de controle de estoque”. SQL Magazine. Brasil: Edição 85 – Ano 7. Páginas: 06 a 13;

NETO, Arilo Claudio Dias .”Introdução à modelagem de dados”. SQL Magazine. Brasil: Edição 86 – Ano 7. Páginas: 06 a 11;

NETO, Arilo Claudio Dias .”Aplicando Normalização a um Modelo de Dados”. SQL Magazine. Brasil: Edição 58 – Ano 5. Páginas: 12 a 21;