GOVERNO DO ESTADO DO CEARÁ

Secretaria da Educação

ESCOLA ESTADUAL DE EDUCAÇÃO PROFISSIONAL - EEEP

ENSINO MÉDIO INTEGRADO À EDUCAÇÃO PROFISSIONAL

CURSO TÉCNICO DE INFORMÁTICA

BANCO DE DADOS

GOVERNO DO
ESTADO DO CEARÁ
Secretaria da Educação

Governador
Cid Ferreira Gomes

Vice Governador
Domingos Gomes de Aguiar Filho

Secretária da Educação
Maria Izolda Cela de Arruda Coelho

Secretário Adjunto
Maurício Holanda Maia

Secretário Executivo
Antônio Idilvan de Lima Alencar

Assessora Institucional do Gabinete da Seduc
Cristiane Carvalho Holanda

Coordenadora da Educação Profissional – SEDUC
Andréa Araújo Rocha

GOVERNO DO ESTADO DO CEARÁ

Secretaria da Educação

Coordenação Técnica Pedagógica
Renanh Gonçalves de Araújo

Equipe de Elaboração
Adriano Gomes da Silva
Cíntia Reis de Oliveira
Fernanda Vieira Ribeiro
Francisco Aislan da Silva Freitas
João Paulo de Oliveira Lima
Liane Coe Girão Cartaxo
Mirna Geyla Lopes Brandão
Moribe Gomes de Alcântara
Niltemberg Oliveira Carvalho
Paulo Ricardo do Nascimento Lima
Renanh Gonçalves de Araújo
Renato William Rodrigues de Souza

Colaboradores
Maria Analice de Araújo Albuquerque
Maria Danielle Araújo Mota
Sara Maria Rodrigues Ferreira Feitosa

FASE I : INTRODUÇÃO A BANCO DE DADOS

1. Conceitos Iniciais

A área de banco de dados é de grande importância no mundo da informática, uma vez que a informação é um bem precioso e deve ser armazenada de forma coerente e adequada, pois é de fundamental importância na tomada de decisão de uma empresa.

Existem vários tipos de banco de dados e eles estão presentes na nossa vida há muito tempo, a lista telefônica, por exemplo, pode ser considerado um banco de dados. Você mesmo já deve ter ligado para algum atendimento eletrônico e ouviu a atendente dizer: “Espere um pouco que estamos consultando nosso banco de dados”. Quando você utiliza o facebook, também está fazendo uso de um banco de dados, pois ele armazena em um sistema de banco de dados suas informações pessoais, mensagens dos amigos, fotos, etc. Se você sair de sua página e entrar novamente, suas informações ainda estarão lá, isso ocorre porque suas informações foram armazenadas em um banco de dados, e, portanto, podem ser recuperadas a qualquer momento, quando o usuário solicitar.

Antigamente as empresas armazenavam informações em arquivos físicos, como fichas de cadastro, mas o surgimento e evolução dos computadores possibilitaram o armazenamento de dados de modo digital. Assim os bancos de dados evoluíram e se tornaram o coração de muitos sistemas de informação.

Atualmente, por mais simples que seja um sistema de informação ele precisará armazenar dados, de forma que possa recuperá-los e/ou alterá-los quando necessário. Por exemplo, se você desenvolver um sistema de informação para a biblioteca da escola, este sistema deverá armazenar dados dos alunos, dos livros, dos empréstimos realizados e devoluções. É para armazenar essas informações e recuperá-las quando necessário de forma rápida e segura que utilizamos um sistema de banco de dados.

1.1. Dado, informação, fato e metadados

Muitos consideram dados e informações como palavras sinônimas, mas na verdade não são. Para entender o que é um banco de dados é muito importante conhecer alguns conceitos básicos.

- **Dado:** é qualquer elemento identificado em sua forma bruta que, por si só, não conduz a uma compreensão de determinado fato ou situação. (Oliveira, 2005).
- **Fato:** é um conjunto de dados relacionados. Registram o mundo real.

- **Informação:** é um agrupamento de dados de forma organizada para fazer sentido, gerar conhecimento, e auxiliar na tomada de decisões de uma empresa.
- **Metadado:** São dados sobre dados. Fornecem uma descrição das características dos dados e do conjunto de relacionamentos que ligam os dados encontrados no banco de dados.

Para os conceitos acima temos o seguinte exemplo: O ano 2013 não faz nenhum sentido se você visualizá-lo sozinho. Se você agora analisar um conjunto de dados relacionados (fato), como: “Brasil”, “2013”, “1º Lugar”, “Copa das Confederações”. Esse fato registra algo do mundo real e a partir dele você chega à informação de que “O Brasil foi campeão da Copa das Confederações do ano de 2013”. Os metadados são algumas informações que você pode ter a respeito de cada dado, por exemplo: “Brasil” (País – nome do dado, texto – tipo do dado), 2013 (Ano – nome do dado, numérico – tipo do dado).

1.2. Histórico dos Bancos de Dados

Como citado anteriormente antigamente as empresas armazenavam dados em fichas de papel que eram organizadas em arquivos físicos através de pastas. Extrair informações e manter esses arquivos organizados era uma tarefa muito custosa. Além disso, o acesso à informação dependia da localização geográfica dos arquivos. Enfim esses arquivos físicos evoluíram para arquivos digitais. No início cada entidade (clientes, funcionários, produtos, etc.) era um arquivo de dados que eram acompanhados de um “software simples” para manipular os dados do arquivo, esses softwares permitiam realizar operações de cadastro, alteração, exclusão e consulta nos arquivos digitais. De fato melhorou bastante, principalmente a tarefa de consulta de informações, porém os arquivos digitais eram ainda uma versão melhorada dos arquivos físicos, mas as entidades precisavam relacionar-se, por exemplo, um produto é fornecido por um fornecedor, e com os arquivos digitais relacioná-las não era uma tarefa muito trivial, os “softwares simples” para manipular os arquivos digitais começaram a ficar “complexos” para permitir os relacionamentos entre entidades.

Na década de 60 a empresa IBM investiu fortemente em pesquisas para solucionar estes problemas dos bancos de dados digitais primitivos. Em 1970, Edgar “Ted” Codd, matemático funcionário da IBM, escreveu um artigo que

viria a mudar tudo isso. Apresentou um modelo relacional onde usuários, sem conhecimento técnico, poderiam armazenar e extrair grandes quantidades de informações de um banco de dados. Na época, ninguém percebeu que as teorias obscuras de Codd desencadeariam uma revolução tecnológica comparável ao desenvolvimento dos computadores pessoais e da internet.

Apesar de ter sido o marco dos bancos de dados relacionais, o artigo de Codd não foi muito explorado no início. Só no final da década de 70 que a IBM desenvolveu um sistema baseado nas ideias do cientista, o "Sistema R". Junto com esse sistema foi criado a linguagem de consulta estruturada (SQL – Structured Query Language) que se tornou a linguagem padrão para bancos de dados relacionais. Embora tenha contribuído para a evolução dos bancos de dados relacionais, o "System R" não foi muito bem sucedido comercialmente, tendo em vista que a IBM voltava-se para o IMS, um sistema de banco de dados confiável, de alta tecnologia, que havia surgido em 1968.

Entre os leitores do artigo de Codd estava Larry Ellison, que havia acabado de fundar uma pequena empresa. Recrutando programadores do Sistema R e da Universidade da Califórnia. Ellison conseguiu colocar no mercado o primeiro banco de dados relacional com base em SQL em 1979, o Oracle 2, bem antes da IBM. Em 1983, a empresa lançou uma versão portátil do banco de dados, teve um faturamento bruto anual de US$ 5.000.000 e mudou seu nome para Oracle. Impelida pela concorrência, a IBM finalmente lançou o SQL/DS, seu primeiro banco de dados relacional, em 1980. Na sequencia vieram SQL Server, MySQL, DBase III, Paradox, etc.

Em 2007, as vendas globais de sistemas de gerenciamento de banco de dados chegaram ao pico de US$ 15 bilhões com a Oracle detendo uma participação de praticamente metade do mercado, seguida pela IBM, com menos de um quarto. A participação do SQL Server da Microsoft cresceu mais rápido do que a de seus competidores, chegando a 14%.

1.3. O que é um Banco de Dados?

A partir do já exposto nas seções anteriores podemos então dizer que banco de dados, ou base de dados, é "uma coleção de dados inter-relacionados, representando informações sobre um domínio específico", ou seja, sempre que for possível agrupar informações que se relacionam e tratam de um mesmo assunto, podemos dizer que temos um banco de dados. Os bancos de dados tem o objetivo de fornecer os dados necessários aos Sistemas de Informação para processamento e geração de informação para os usuários.

Vale lembrar que um banco de dados é projetado, construído e preenchido com dados para um propósito específico. Ele representa algum

aspecto do mundo real, algumas vezes chamado de “mini-mundo”. Mudanças no mini-mundo provocam mudanças no banco de dados.

O usuário pode realizar 4 operações básicas sobre um banco de dados que são:

- **Inserção:** onde ele pode inserir um novo dado no banco;
- **Remoção:** quando ele apaga alguma registro de dados;
- **Atualização:** quando ele edita ou altera algum registro;
- **Consulta:** quando ele quer apenas visualizar os dados contidos no banco de dados.

Essas operações sobre o banco de dados não acontecem diretamente, os usuários as realizam a partir de um sistema de informação. Esse sistema de informação fica conectado a um Sistema Gerenciador de Banco de Dados, que acessa os dados realizando as operações solicitadas pelo usuário.

Podemos dizer que um Sistema de Banco de Dados envolve 4 componentes básicos: Dados, Software, Hardware e Usuário. Sobre o Dado já explicamos que é o componente principal, são os registros que serão armazenados. Hardware é toda a parte física, a máquina em si. Alguns o resumem apenas ao computador, mas é um erro, visto que mesmo um celular pode enviar e receber dados. Software é toda a “parte lógica”, os programas aplicativos, os programas de acesso aos dados, até mesmo o sistema operacional. Sobre os Usuários, alguns livros os dividiram em três tipos: o Administrador de Banco de Dados, responsável por monitorar e gerenciar todas as bases de dados criadas no SGBD; o Programador de Aplicativos, responsável por modelar a base de dados e implementa-la no SGBD, bem como desenvolver a aplicação que se conectarão ao Banco de Dados; e o Usuário Final, que irão trabalhar diretamente com a aplicação desenvolvida, entrando com dados e não precisam ter nenhum conhecimento sobre banco de dados, o que importa pra ele é que os dados fiquem armazenados.

1.4. Abstração de Dados

O grande objetivo de um sistema de banco de dados é prover aos usuários uma visão abstrata dos dados. Isto é, o sistema omite certos detalhes de como os dados são armazenados e mantidos. Entretanto, para que o sistema possa ser utilizado, os dados devem ser buscados de forma eficiente. Este conceito tem direcionado o projeto de estrutura de dados complexas para a representação de dados em um banco de dados. Uma vez que muitos dos usuários de banco de dados não são treinados para computação, a complexidade está escondida deles através de diversos níveis de abstração que simplificam a interação do usuário com o sistema.

- **Nível Interno ou físico:** o nível mais baixo de abstração descreve como os dados estão realmente armazenados. Neste nível se desenham os arquivos que contém a informação, a localização dos mesmos e sua organização, ou seja, criam-se os arquivos de configuração.
- **Nível conceitual:** o próximo nível de abstração descreve quais dados estão armazenados de fato no banco de dados e as relações que existem entre eles. Aqui o banco de dados inteiro é descrito em termos de um pequeno número de estruturas relativamente simples. Embora as implementações de estruturas simples no nível conceitual possa envolver complexas estruturas de nível físico, o usuário do nível conceitual não precisa preocupar-se com isso. O nível conceitual de abstração é usado por administradores de banco de dados, que podem decidir quais informações devem ser mantidas no BD;
- **Nível externo ou de visão**: é o mais próximo ao usuário e descreve apenas parte do banco de dados. Apesar do uso de estruturas mais simples do que no nível conceitual, alguma complexidade perdura devido ao grande tamanho do banco de dados. Muitos usuários do sistema de banco de dados não estarão interessados em todas as informações. Em vez disso precisam de apenas uma parte do banco de dados. O nível de abstração das visões de dados é definido para simplificar esta interação com o sistema, que pode fornecer muitas visões para o mesmo banco de dados.

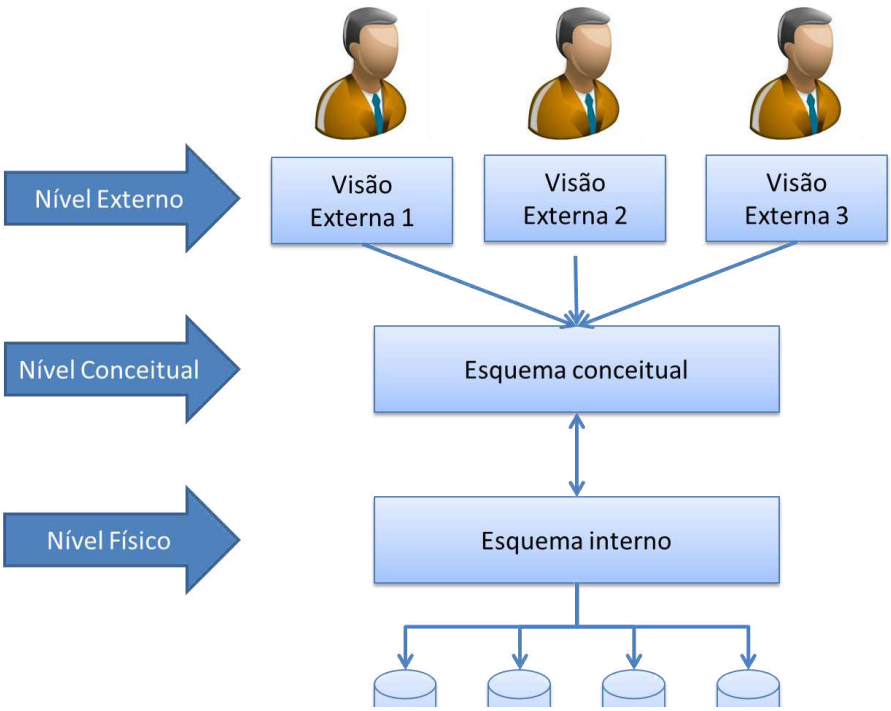


Figura 2.1 – Níveis de Abstração de Dados

EXERCÍCIOS

1. Qual a importância dos bancos de dados para os Sistemas de Informação?

_____
_____

2. O que é um Banco de Dados? Cite dois exemplos de sistemas que você acredita que utiliza banco de dados.

_____
_____

3. Qual a diferença entre dado, fato, informação e metadados?

_____
___

4. Quais os principais componentes de um Sistema de Banco de Dados?

_____
_____
_____
_____

5. Quais as operações básicas que o usuário pode realizar em um banco de dados?

_____
_____
_____
_____

6. Quais os níveis de abstração de um Banco de dados?

_____
_____
_____
_____

2. Sistema Gerenciador de Banco de dados - SGBD

Tudo que fazemos em um banco de dados passa pelo SGBD! Ele é responsável por salvar os dados no HD, manter em memória os dados mais acessados, ligar dados e metadados, disponibilizar uma interface para programas e usuários externos acessem o banco de dados (para banco de dados relacionais, é utilizada a linguagem SQL), encriptar dados, controlar o acesso a informações, manter cópias dos dados para recuperação de uma possível falha, garantir transações no banco de dados, enfim, sem o SGBD o banco de dados não funciona!

Podemos então definir um Sistema Gerenciados de Banco de Dados como um o conjunto de programas de computador (softwares) responsáveis pelo gerenciamento de uma base de dados. Seu principal objetivo é retirar da aplicação cliente a responsabilidade de gerenciar o acesso, a manipulação e a organização dos dados. O SGBD disponibiliza uma interface para que seus usuários possam incluir, alterar ou consultar dados previamente armazenados. O SGBD então é responsável por manipular os dados contidos no banco de dados. Porém suas funcionalidades vão muito além de manipulação de dados, ele também é responsável por definir e construir um banco de dados.

2.1. Características de um SGBD

O SGBD faz todo o gerenciamento de transações dos bancos de dados contidos nele. Uma transação em um banco de dados consiste em um conjunto de operações que são tratadas como uma unidade lógica indivisível. Por exemplo, quando vamos fazer uma transferência bancária, são feitas no mínimo duas operações, a retirada do dinheiro da conta de quem está transferindo e o depósito na conta da pessoa que vai receber o valor transferido, ou seja, a transferência é o conjunto dessas operações.

As transações realizadas pelo SGBD nos bancos de dados devem seguir algumas propriedades fundamentais conhecidas como ACID (Atomicidade, Consistência, Isolamento e Durabilidade).

- **Atomicidade**: Capacidade de uma transação ter todas as suas operações executada ou nenhuma delas. É tudo ou nada. Caso a transação não aconteça totalmente o banco de dados executa um rollback e retorna ao seu estado consistente anterior, caso todas a transação aconteça é executado o commit;

- **Consistência**: A execução de uma transição deve levar o banco de dados de um estado consistente a outro estado consistente.

- **Isolamento:** A propriedade de isolamento garante que a transação não será interferida por nenhuma outra transação concorrente.

- **Durabilidade:** A propriedade de durabilidade garante que o que foi salvo, não será mais perdido.

Além da gerencia de transações o SGBD possui algumas características que permitem controlar e acompanhar melhor os dados armazenados. As características básicas de um SGBD são:

- **Controle de Redundâncias** - A redundância consiste no armazenamento de uma mesma informação em locais diferentes, provocando inconsistências. Se uma mesma informação estiver armazenada em mais de um lugar pode acontecer de você atualizar em um lugar e esquecer-se de atualizar no outro, ficando o banco de dados inconsistente.
- **Controle de concorrência** - O SGBD permite que duas ou mais pessoas acessem a mesma base de dados ao mesmo tempo e o sistema deve controlar para que um acesso não interfira no outro. Um sistema de compras Web por exemplo várias pessoas podem realizar uma compra ao mesmo tempo, e o próprio SGBD controla pra que os dados de todas as compras sejam gravados corretamente.
- **Controle de Acesso** - O SGDB tem mecanismos para criação de regras de segurança, que vão desde a definição de login e senha para os usuários, até a permissão de acesso ao SGBD e acesso aos dados armazenados. É possível definir por exemplo que um usuário tem permissão somente para leitura de dados, e um outro usuário tenha permissão para criar base de dados e manipulá-la, mas não pode criar novos usuários ou fazer backup.
- **Controle de Integridade** – Um SGBD pode definir regras que garantem a integridade dos dados. Essas regras são definidas para garantir que os dados contidos no banco de dados estejam corretos. Por exemplo, podemos definir uma regra em um banco que o campo sexo pode receber somente “M” para masculino e “F” para feminino, não aceitando outro tipo de letra, o que deixaria dados errados no banco.
- **Backups** - O SGBD apresenta facilidade para recuperar falhas de hardware e software, através da existência de arquivos de "pré-imagem" ou de outros recursos automáticos, exigindo minimamente a intervenção de pessoal técnico.

Existem vários SGBD, e cada um deles implementa um banco de dados (ou vários) de maneira diferente, mas para o usuário isso é quase transparente, pois a linguagem de acesso aos dados é a mesma, o SQL.

Os fatores que levam a escolha do SGBD são tempo de resposta, segurança, preço, espaço para armazenamento, quantidade de processos que podem ser realizados por minutos, entre outras características. Abaixo temos os principais SGBD do mercado.

PESQUISA

1. Pesquise na internet um comparativo das principais vantagens e desvantagens dos principais SGBDs existentes no mercado.

Sugestão: Dividir as turma em equipes, onde cada equipe ficará responsável por pesquisar sobre um SGBD. Realizar um debate sobre as vantagens e desvantagens enfocando suas características(tempo de resposta, segurança, preço, espaço para armazenamento, quantidade de processos que podem ser realizados por minutos, etc.)

2.2. Arquitetura do SGBD

Quanto a arquitetura o SGBD pode ser classificados dentre quatro tipos:

2.2.1. Arquitetura Stand-Alone (Sistema de Computador Pessoal)

Figura 3.1 – Arquitetura Stand-Alone

Os computadores pessoais trabalham em sistema stand-alone, ou seja, fazem seus processamentos sozinhos. O SGBD roda na própria máquina. No começo esse processamento era bastante limitado, porém, com a evolução do hardware, tem-se hoje PCs com grande capacidade de processamento. Eles funcionam como hospedeiros e terminais. Desta maneira, possuem um único aplicativo a ser executado na máquina. A principal vantagem desta arquitetura é a simplicidade.

2.2.2. Arquitetura Centralizada

Figura 3.2 – Arquitetura Centralizada

Nessa arquitetura existe um computador com grande capacidade de processamento, o qual é o hospedeiro do SGBD e emuladores para os vários aplicativos. Esta arquitetura tem como principal vantagem a de permitir que muitos usuários manipulem grande volume de dados. Sua principal desvantagem está no seu alto custo, pois exige ambiente especial para mainframes e soluções centralizadas.

2.2.3. Arquitetura Cliente-Servidor

Nesse tipo de arquitetura o cliente (front_end) executa as tarefas do aplicativo, ou seja, fornece a interface do usuário (tela, e processamento de entrada e saída). O servidor (back_end) executa as consultas no SGBD e retorna os resultados ao cliente. Apesar de ser uma arquitetura bastante popular, são necessárias soluções sofisticadas de software que possibilitem: o tratamento de transações, as confirmações de transações (commits), desfazer transações (rollbacks), linguagens de consultas (stored procedures) e gatilhos (triggers).

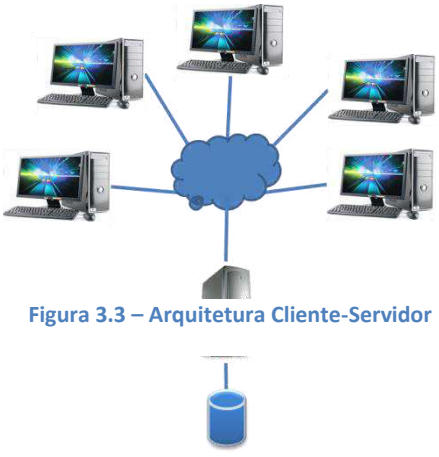


Figura 3.3 – Arquitetura Cliente-Servidor

O servidor de banco de dados faz a verdadeira operação de busca e retorna somente os dados que preencham corretamente a consulta do usuário. O sistema cliente servidor possui algumas vantagens como: menor intensidade de tráfego de dados na rede comparado a

arquitetura distribuída; são rápidos, pois as consultas são feitas em servidores de alta potencia. A maior desvantagem dessa arquitetura é que eles exigem que os dados sejam armazenados num único sistema.

2.2.4. Arquitetura Distribuída

Nesta arquitetura, a informação está distribuída em diversos servidores. Cada servidor atua como no sistema cliente-servidor, porém as consultas oriundas dos aplicativos são feitas para qualquer servidor indistintamente. Caso a informação solicitada seja mantida por outro servidor ou servidores, o sistema encarrega-se de obter a informação necessária, de maneira transparente para o aplicativo, que passa a atuar consultando a rede, independente de conhecer seus servidores, tanto os dados como as funções de processamento são distribuídos em diversos locais.

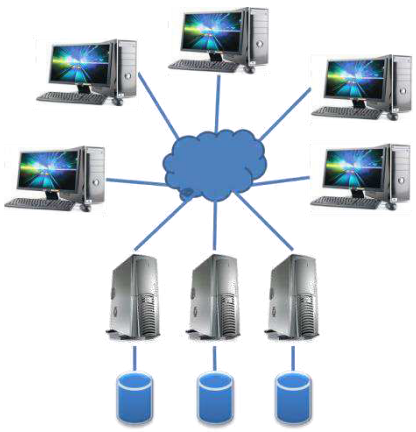


Figura 3.4 – Arquitetura Distribuida

Nos sistemas distribuídos os dados podem estar replicados, ou seja, eles se repetem em cada nó da rede, o que aumenta a disponibilidade do banco, ou os dados podem estar fragmentados, ou seja, divididos por vários nós da rede, aumentando a velocidade pois permite processamento paralelo.

Dentre as vantagens dos sistemas distribuídos estão o menor risco de falhas, pois quando um nó falha, o trabalho é mantido pelos outros nós da rede entre e outras vantagens. Como desvantagem essa arquitetura é mais complexa de ser implementada, e é mais propensa a falhas de segurança tendo em vista os dados estar espalhados em vários locais.

EXERCÍCIOS

1. O que é um SGBD, e qual sua principal função?

_____
_____
_____
_____
_____
_____

2. Umas das características de um SGBD é o gerenciamento de transações. O que é uma transação no contexto de banco de dados?

_____
_____
_____
_____
_____
_____

3. Explique cada uma das propriedades ACID do gerenciamento de transações do SGBD.

a.Atomicidade:

_____
_____

b.Consistência:

_____

_____

c.Isolamento:

_____

_____

d.Durabilidade:_____
_____

_____

4. Quando um SGBD executa um rollback?

_____
_____

5. Explique com suas palavras cada uma das características do SGBD.

_____
_____
_____
_____
_____
_____

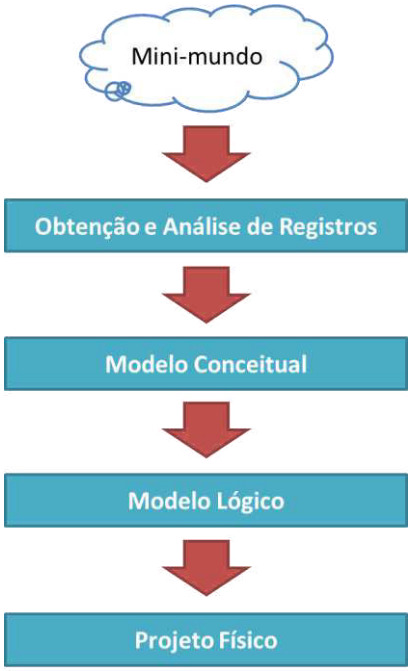
6. Quais as principais arquitetura de SGBD? Aponte vantagens e desvantagens de cada um deles.

_____
_____
_____

FASE I I: MODELAGEM DE BANCO DE DADOS

3. Introdução a Modelagem de Dados

Como vimos na primeira fase de nosso aprendizado um banco de dados é um conjunto de dados devidamente relacionados, e representam algum aspecto no mundo real. Para obtermos um banco de dados que atenda as necessidades de forma eficiente e coerente precisamos fazer a modelagem dos dados que serão armazenados no banco. O objetivo da modelagem de dados é transmitir e mostrar uma representação única, não redundante, e resumida, dos dados de uma aplicação. A fase de modelagem é a principal etapa no projeto de desenvolvimento de um banco de dados. Por isso é muito importante que se dedique tempo e esforço no desenvolvimento de uma boa modelagem de dados.



O projeto de desenvolvimento de um banco de dados compreende as seguintes fases:

- **Modelagem conceitual:** refere-se ao desenvolvimento de um modelo inicial da base de dados que reflitam as necessidades do usuário. Essa modelagem preocupa-se em descrever quais dados serão armazenados na base de dados e quais dados se relacionam. Para fazer o modelo conceitual é necessário entender que dados o usuário final espera que o sistema disponibilize. A modelagem conceitual fornece uma visão mais próxima do modo como os usuários visualizam os dados realmente.

- **Modelagem lógica:** Compreende o processo de desenvolver como os dados serão armazenados no sistema e como irão se relacionar. Isso significa transformar o modelo conceitual obtido na primeira fase num modelo mais próximo da implementação, ou seja, fornece uma visão mais detalhada do modo como os dados são armazenados no computador. Nessa fase também são criados os dicionários de dados, e feita verificação se o modelo está normalizado, veremos os conceitos de normalização de dados mais a frente.

- **Implementação do modelo lógico:** Uma vez que toda a etapa de modelagem esteja concluída, será necessário implementar ou criar a base de dados no SGBD escolhido. Essa fase requer que o desenvolvedor conheça a linguagem SQL e conheça o SGBD selecionado.

Um ponto importante a se destacar aqui é que para ser possível criar uma modelagem da base de dados é necessário grande interação do usuário ou responsável pela análise de requisitos do sistema. Essa interação se faz necessária uma vez que o projetista da base de dados, para desenvolvê-la, precisa ter uma clara compreensão do que o usuário espera do sistema, que tipo de relatórios o usuário espera que este disponibilize, bem como saber quais são os objetivos do sistema. A modelagem de dados tem como base para seu inicio o levantamento de requisitos do sistema.

3.1. Modelo de Dados

Um modelo de dados compreende a descrição de dados que devem ser armazenados pelo sistema e como esses dados devem se relacionar. Para que seja possível fazer essa descrição, é utilizada uma linguagem de modelagem, que pode ser textual ou gráfica. Um modelo de dados deve explicitar os tipos de dados armazenados e as restrições que esses dados possuem. Diversos modelos de dados foram propostos e estão divididos em três diferentes grupos: Modelos baseados em registros, baseados em objetos e modelos físicos.

3.1.1. Modelos baseados em Registros

São usados na descrição de dados nos níveis conceitual e visão, especificam tanto a estrutura global, como uma descrição em auto nível da implementação, dividem-se em:

- **Modelo Hierárquico** - O modelo hierárquico foi o primeiro a ser reconhecido como um modelo de dados. Ele organiza os dados de cima para baixo, como uma árvore e é definido como uma coleção de registros conectados por meio de ligações, onde cada registro é uma coleção de campos e cada campo contém um único valor. O registro da hierarquia que precede a outros é o registro-pai, os outros são chamados de registros-filho. Os dados são classificados hierarquicamente, em formato de árvore descendente.

- **Modelo em Rede** - O modelo em redes surgiu como uma extensão ao modelo hierárquico, eliminando o conceito de hierarquia e permitindo que um mesmo registro estivesse envolvido em várias associações, criando conexões bastante complexas e são bastante utilizados em sistemas para computadores de grande porte.

- **Modelo Relacional** - O modelo relacional apareceu devido à necessidade aumentar a independência de dados nos SGBDs e prover um conjunto de funções apoiadas em álgebra relacional para armazenamento e recuperação de dados. É o modelo mais utilizado hoje no mercado, pois se revelou ser o mais flexível e adequado ao solucionar os vários problemas que se colocam no nível da concepção e implementação da base de dados.

3.1.2. Modelos baseados em Objetos

Usados na descrição de dados no nível conceitual e visão, proporcionam ampla e flexível capacidade de estruturação e permitem a especificação de restrições de dados de forma explicita. Entre os modelos mais conhecidos estão:

- **Modelo Entidade-Relacionamento** - É o modelo que está sendo largamente utilizado na prática, ele se baseia numa percepção do mundo real e consiste numa coleção de objetos básicos chamados de entidades e de relacionamento entre estes objetos.

- **Modelo Orientados a Objetos** – No modelo OO toda e qualquer entidade do mundo real é representada por um modelo conceitual, o objeto. Um objeto sempre estará associado a um estado e um comportamento. A motivação para seu surgimento está em função dos limites de armazenamento e representação semântica impostas no modelo relacional. São muito utilizados em sistemas de informações geográficas (SIG), os sistemas CAD e CAM, que são mais facilmente construídos usando tipos complexos de dados. Possui algumas desvantagens como falta de padronização das linguagens de manipulação dos dados, alto custo de aquisição das novas tecnologias;

3.1.3. Modelos Físicos

Usados para descrever os dados em seu nível mais baixo. Capturam os aspectos de implementação do SGBD.

EXERCÍCIOS

1. Qual o objetivo da modelagem de um banco de dados?

_____
_____
_____

_____
_____
_____

2. Quais as etapas de um Projeto de Desenvolvimento de um Sistema de Banco de Dados? O que é feito em cada uma dessas etapas?

_____
_____
_____
_____
_____
_____

3. O que são modelos de dados?

_____
_____
_____
_____
_____
_____

4. Qual a diferença entre os modelos orientados a registros e os modelos orientados a objetos? Cite exemplos de cada um deles.

_____
_____
_____
_____
_____
_____

4. Modelo Entidade-Relacionamento (MER)

O Modelo Entidade-Relacionamento é um modelo conceitual, e deve estar o mais próximo possível da visão que o usuário tem dos dados. Tem por objetivo descrever quais dados deverão ser armazenados pela aplicação e como esses dados se relacionam. Nele estão representadas todas as entidades de interesse do domínio da aplicação, com seus respetivos atributos e o relacionamento existente entre as entidades. Este modelo é representado de forma gráfica pelo Diagrama de Entidade-Relacionamento (DER).

Existem algumas ferramentas que auxiliam na criação deste modelo. Nesta apostila utilizaremos o brModelo, uma ferramenta de código aberto extremamente compacto e leve, podendo rodar diretamente de pendrives ou

qualquer outro dispositivo portátil. Além disso, não é necessário instalá-lo, basta fazer o download no site: http://sis4.com/brModelo.

4.1. Entidades

Uma entidade representa um conjunto de objetos de um mesmo tipo do mundo real e sobre os quais se pretende armazenar dados. Geralmente as entidades são reconhecidas por serem substantivos e uma forma simples de identifica-las em um domínio de aplicação é fazer as seguintes perguntas:

- Sobre que objetos ou coisas precisamos guardar informação?
- Há mais de um objeto deste tipo?
- Existe uma chave capaz de identificar cada um dos objetos unicamente?

Para ser entidade as três perguntas acimas precisar ter respostas afirmativas.

Vamos ver como funciona:

PEDIDO

Vendedor:	Zé			**Nº Pedido**
Data Venda:	00/00/0000			100002434
Cliente:	XPTO INFORMATICA			
Endereço:	Rua da Frente nº10			
CNPJ	1233322222/0001-44	**I.E.**	122.112.222.112.33	
CODIGO	**PRODUTO**	**QTDE.**	**R$ Unitário**	**R$ Total**
555	Pen-drive Ching-Ling	100	15	1500

Figura 5.1 – Nota de compra de produtos

A imagem acima é uma nota de compra de uma empresa. Para esta aplicação o pedido é uma provável entidade. Vamos verificar se ele atende os requisitos para que seja uma entidade.

PERGUNTAS	RESPOSTAS
Há necessidade de guardarmos a informação dos pedidos?	Sim, temos que guardas as informações de todos os pedidos.
Pode haver mais de um pedido?	Sim, existem diversos pedidos e não apenas um.
Existe uma chave capaz de identificar cada um dos pedidos unicamente?	Sim, cada pedido realizado possui um número único para controle de pedidos.

Verifique agora as perguntas para a quantidade da imagem acima

PERGUNTAS	RESPOSTAS
Há necessidade de guardarmos a informação da quantidade?	
Pode haver mais de uma quantidade?	
Existe uma chave capaz de identificar cada quantidade unicamente?	

A partir deste ponto vamos tomar como exemplo para um melhor entendimento do modelo Entidade-Relacionamento, um pequeno sistema de banco de dados de uma escola, de acordo com o levantamento de requisitos abaixo:

Uma escola precisa de um sistema para guardar os registros de dados de alunos, professores, disciplinas e turmas. Nesse sistema é preciso saber do aluno: matricula, nome, data de nascimento; dos professores é importante guardar informações como: matricula, nome, cpf, telefone, endereço e data de nascimento. O professor pode possui telefone celular e residencial. Cada turma tem um nome e um código, assim como as disciplinas também tem nome e código. Um professor pode dar aula de várias disciplinas em várias turmas. Uma disciplina pode ser ministrada por mais de um professor. Um aluno só pode fazer parte de uma única turma.

Podemos citar então como entidades desse sistema: professor, aluno, disciplina, turma, curso. Cada uma dessas entidades armazenará um conjunto de objetos do mesmo tipo. Uma entidade é representada graficamente por um retângulo com o nome da entidade dentro do retângulo. Por exemplo:

Figura 5.2 – Exemplo de Entidade

4.2. Atributos

Cada entidade possui algumas propriedades que definem suas características. Essas características das entidades são chamadas de **atributos**. Por exemplo, para a entidade "*Professor*", é necessário armazenar dados como: *CPF, nome, telefone, endereço, data de nascimento, matricula*. Esses dados são atributos da entidade "*Professor*" e são eles que caracterizam um objeto do tipo professor.

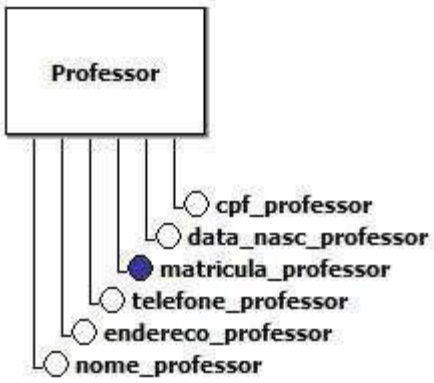


Figura 5.3 – Exemplo de Entidade com atributos

Um atributo pode ser representado graficamente por uma bolinha ligando a entidade com o nome do atributo ao lado, conforme figura 5.3.

Alguns atributos podem ser divididos em subpartes com significados independentes. Por exemplo, o atributo "*endereço_professor*" da entidade "*Professor*" acima, pode ser dividido em: *rua, numero, bairro, cidade, estado e CEP*. Um atributo que é composto de outros atributos mais básicos é chamado **atributo composto**. Já, atributos que não são divisíveis são chamados **atributos simples**. Os atributos compostos podem ser representados como na Figura 5.4.

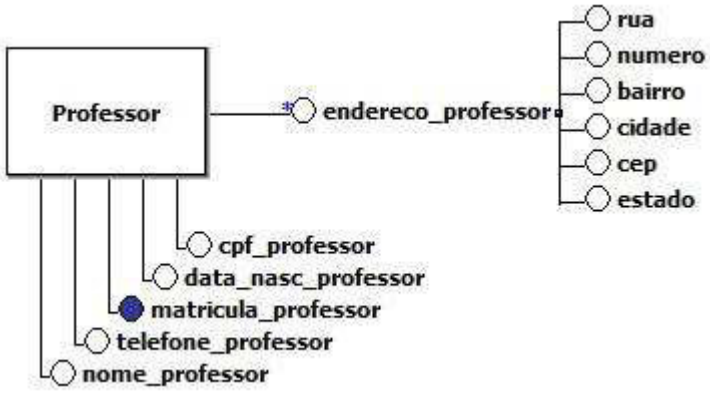


Figura 5.4 – Exemplo de atributo composto

Muitos atributos têm apenas um único valor. Esses atributos são chamados **atributos monovalorados**, por exemplo, o atributo "*data_de_nasc_prof*" da entidade "*Professor*". Em outros casos, um atributo pode ter um conjunto de valores, tais atributos são chamados de **atributos multivalorados**, por exemplo, o professor pode possuir mais de um telefone, um residencial e um celular. Um atributo multivalorado é representado na figura 5.5.

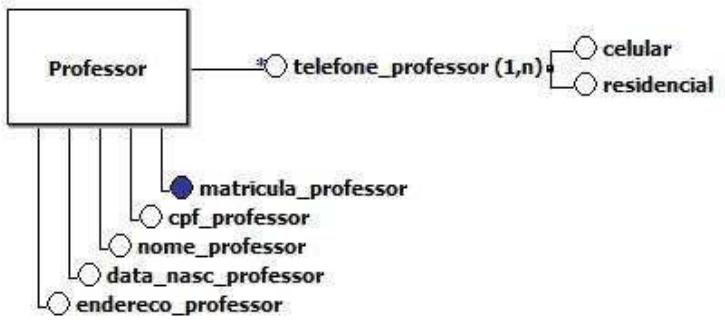


Figura 5.5 – Exemplo de atributo multivalorado

Um **atributo derivado** é aquele cujo valor deriva de outro(s) atributo(s). Por exemplo, podemos acrescentar ao professor do exemplo acima, o atributo idade que é calculado automaticamente a partir da data de nascimento e data atual pela própria aplicação ou SGBD.

Um atributo tem um **valor nulo** quando uma entidade não possui um valor para ele. O valor nulo representa a inexistência de um valor, ou seja, significa que o usuário não precisa cadastrar um valor para o atributo e pode deixa-lo vazio. Em algumas situações é inevitável que permitamos valores nulos para os atributos, por exemplo, digamos que seja acrescentado à entidade professor o atributo email. Alguns professores possuem email, outros não. Aqueles que não possuírem email terão nesse campo um valor nulo, ou vazio.

Um conjunto de atributos que tem a propriedade de identificar univocamente uma ocorrência (instância) de uma entidade é chamado de **identificador** desta entidade. Toda entidade deve possuir um identificador. O identificador também é conhecido como **Chave Primária (Primary Key – PK)**. Você deve ter reparado que na entidade **Professor**, o atributo *matricula_professor* é definido com uma bolinha pintada, esta é a forma de representar a chave primária da entidade professor.

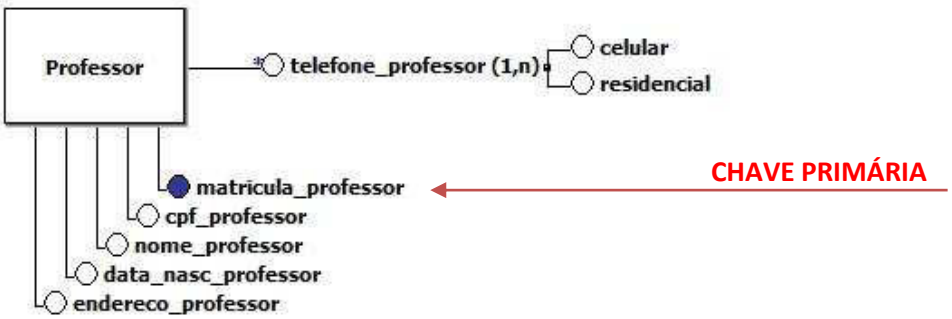
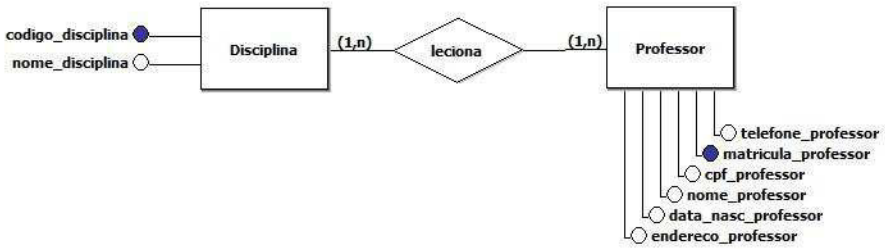


Figura 5.6 – Exemplo de chave primária em uma entidade

4.3. Relacionamento

Um relacionamento é uma associação entre as entidades. Como vimos no inicio da apostila os dados devem ser armazenados e estarem relacionados na base de dados para que possamos utilizá-los eficientemente. O relacionamento entre os dados é o que nos permite descobrir, dadas duas entidades como "**Professor**" e "**Disciplina**", qual a disciplina que o professor



leciona, conforme mostra a figura 5.7.

Representação de um relacionamento

Figura 5.7 – Exemplo de Relacionamento

Um relacionamento é representado por um losango como o nome do relacionamento no centro. O nome do relacionamento representa a relação que existe entre as entidades. Esse nome pode ser um verbo, como por exemplo: pertence, leciona, estuda, possui, etc.; ou também pode ser uma composição dos nomes das entidades: "Aluno_Turma" ao invés de pertence.

Um relacionamento pode ter atributos. Esses atributos são denominados de atributos descritivos. Imagine que seja necessário armazenar a ano em que um professor lecionou determinada disciplina. O atributo ano, não pode pertencer nem a entidade "*Professor*" e nem a entidade "*Disciplina*". Esse atributo pertence ao relacionamento "*leciona*", ou seja, é um atributo do relacionamento. E ele só deve ser preenchido com um valor, quando for feita a relação entre professor e disciplina.

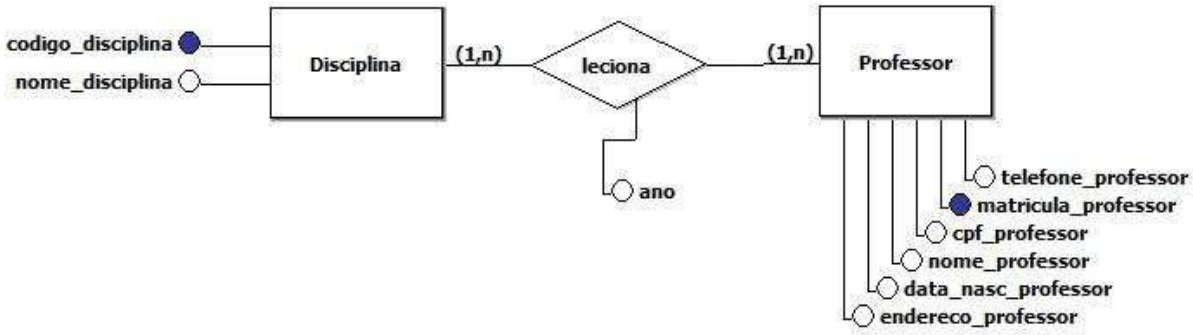


Figura 5.8 – Exemplo de atributo descritivo

EXERCÍCIOS

1. Para que serve o Diagrama Entidade Relacionamento?

_____
_____
_____
_____
_____
_____

2. Defina o que é entidade e dê pelo menos três exemplos de entidades (diferente dos apresentados na apostila).

_____
_____
_____
_____
_____
_____

3. Defina atributo e cite 4 atributos para cada entidade que você citou na questão 2.

_____
_____
_____
_____
_____
_____

4. Quais os tipos de atributos? Explique cada um deles.

_____
_____

5. Explique o que é chave primária e para que ela serve. Apresente 3 exemplos de atributos que poderiam ser chave primária e explique o porque.

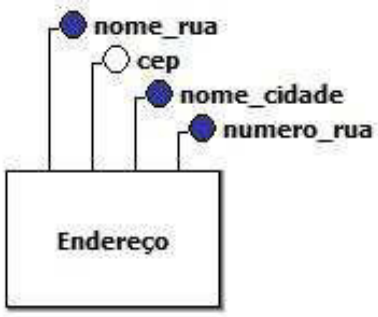
_____
_____
_____
_____
_____
_____

6. Uma chave primária pode assumir valor nulo? Explique sua resposta.

_____
_____
_____
_____

7. Dado o DER abaixo, coloque os atributos para cada entidade e marque as chaves primárias para cada entidade.

8. Dado diagrama abaixo, pode-se afirmar que a entidade “Endereço” possui três chaves primárias? Explique sua resposta.



_____
_____
_____
_____
_____
_____
_____
_____

___

4.4. Cardinalidade de um relacionamento

A cardinalidade indica quantos objetos (instancias) de uma entidade, podem se relacionar com outra entidade através de um relacionamento.

Na hora de fazer a cardinalidade de um relacionamento deve ser considerado as cardinalidades mínimas e máximas.

- **Cardinalidade mínima:** é o número mínimo de instâncias de uma entidade que devem se relacionar com uma instância de outra entidade. A cardinalidade mínima é usada para indicar o tipo de participação da entidade em um relacionamento e sempre ocupa a primeira posição dentro do parêntese que representa a cardinalidade. Esta participação no relacionamento pode ser:

Figura 5.9 – Exemplo de Cardinalidade em um relacionamento

  - **Parcial ou Opcional**: quando uma ocorrência da entidade pode ou não participar de determinado relacionamento; é indicado pela cardinalidade mínima = 0 (zero). Por exemplo, na figura 5.10, pode existir algum professor que é diretor da escola. Observe que ser diretor na escola não é obrigatório, por isso atribuímos o valor 0 (zero), ao relacionamento dirige, no lado escola.
  - **Total ou Obrigatória:** quando todas as ocorrências de uma entidade devem participar de determinado relacionamento; é indicado pela cardinalidade mínima = 1. Na figura 5.10, uma escola obrigatoriamente é dirigida por um professor.

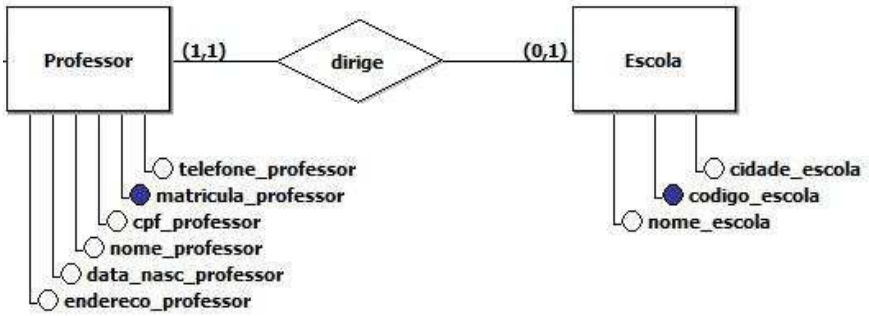


Figura 5.10 – Exemplo de cardinalidade Mínima

- **Cardinalidade máxima:** é o número máximo de instâncias de uma entidade que podem se relacionar com uma instância de outra entidade. A cardinalidade máxima pode assumir valores 1 e N, e ocupa a segunda posição dentro do parêntese que denota a cardinalidade de uma entidade em relação a outra.

- **Cardinalidade 1:1 (Um para Um) –** Ocorre quando uma instancia de uma entidade pode se relacionar a apenas um objeto de outra entidade e vice-versa. Por exemplo, na figura 5.10, um professor é diretor de uma escola, e uma escola só pode ser dirigida por um professor.

- **Cardinalidade 1:N (Um para muitos)** – Ocorre quando uma instancia de uma entidade pode se relacionar com mais de um objeto de outra entidade, mas a recíproca não é verdadeira. Por exemplo, na figura 5.11, em uma escola podem lecionar vários professores, porém um professor só pode lecionar em uma escola.

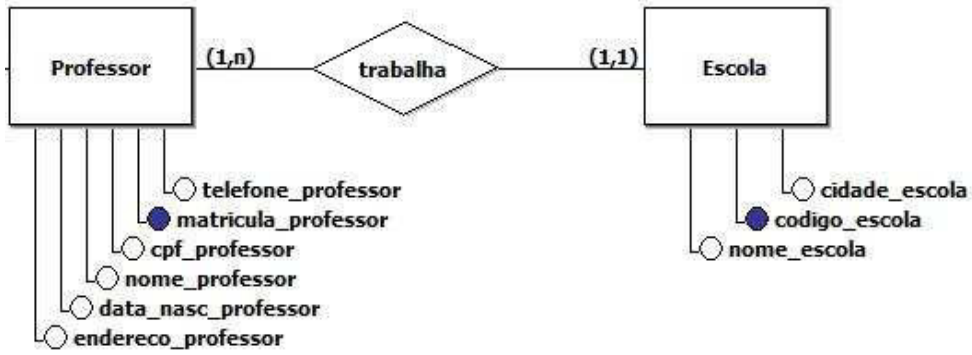


Figura 5.11 – Exemplo de Relacionamento Um para Muitos

- **Cardinalidade N:N (Muitos para muitos)** – Ocorre quando uma ocorrência de uma entidade pode se relacionar com mais de um objeto de outra entidade e vice versa. Por exemplo, um professor pode lecionar mais de uma disciplina, assim como uma disciplina pode ser lecionada por mais de um professor.

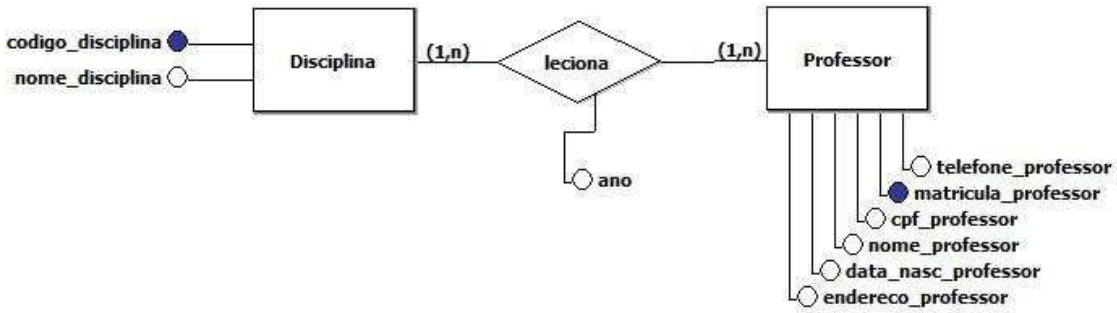


Figura 5.12 – Exemplo de Relacionamento Muitos para Muitos

Quando queremos descobrir a cardinalidade de um relacionamento fazemos a seguinte pergunta: “Um objeto da minha entidade X, pode estar

relacionamento no mínimo com quantos objetos da minha entidade Y? E no máximo?".

Vamos verificar o relacionamento casamento de uma entidade homem, e mulher. Fazemos então a pergunta acima:

- Um homem pode ser casado no mínimo com quantas mulheres na outra entidade? E no máximo?

- Uma mulher pode ser casada no mínimo com quantos homens na outra entidade? E no máximo?

4.5. Grau de um Relacionamento

O grau de um relacionamento indica quantas entidades estão envolvidas em um relacionamento. E pode ser classificado em binário e ternário.

- **Relacionamento binário** - é aquele em que duas entidades estão ligadas por um relacionamento. O relacionamento da figura 5.12, é um exemplo de relacionamento binário, pois um professor está relacionado a uma disciplina.

- **Relacionamento ternário** - é quando existem três entidades envolvidas em um relacionamento. Digamos que queremos acrescentar na nossa modelagem que um professor leciona uma disciplina para uma turma. Teríamos então que associar mais uma entidade ao relacionamento leciona. Ficaríamos então com um relacionamento ternário, mostrado na figura

- 5.13.

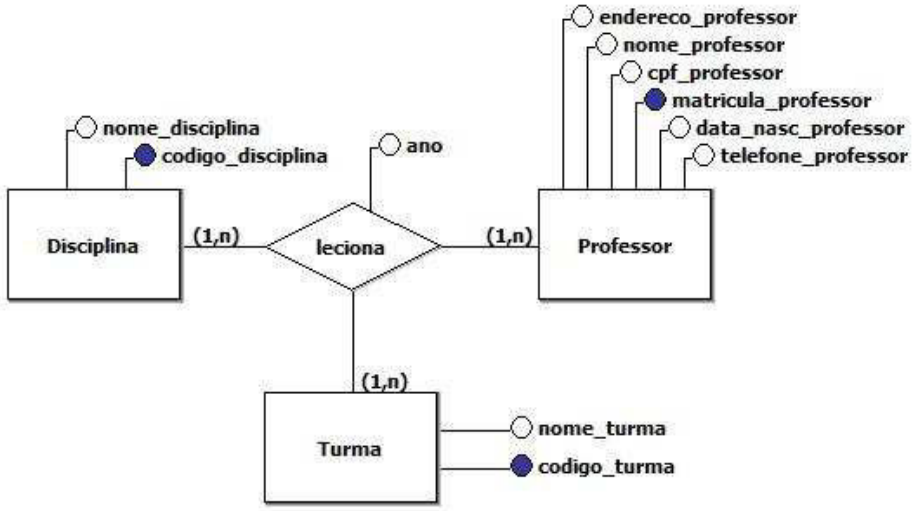


Figura 5.13 – Exemplo de relacionamento ternário

Relacionamentos maiores que ternários devem ser evitados, pois são difíceis de serem entendidos e de serem implementados tornando a relação bastante complexa.

Quando não é possível definir uma chave primária, nem simples e nem composta, para uma entidade, temos uma **entidade fraca**. A entidade fraca é dependente de outra entidade, e o relacionamento entre ela e outra entidade é normalmente 1:N. Representamos uma entidade fraca, com a linha que liga seu relacionamento em negrito, como mostra o exemplo na figura 5.14.

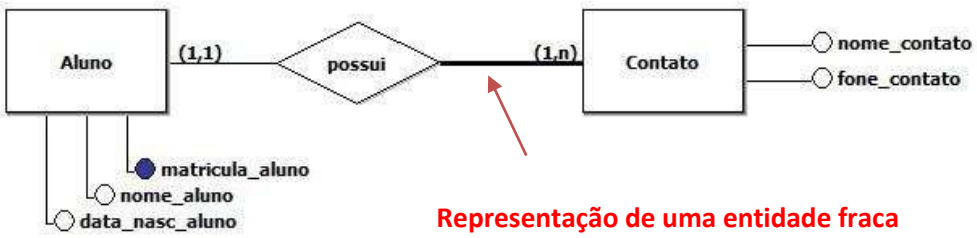


Figura 5.14 – Exemplo de entidade Fraca

EXERCÍCIOS

1. O que é cardinalidade? Qual a diferença entre cardinalidade mínima e máxima?

_____

_____

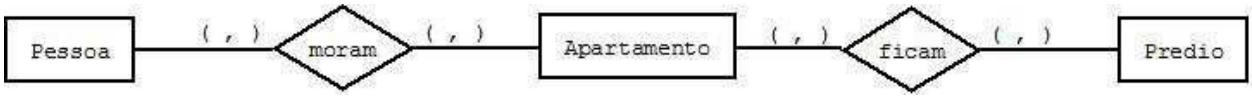
2. O que determina o grau de um relacionamento?

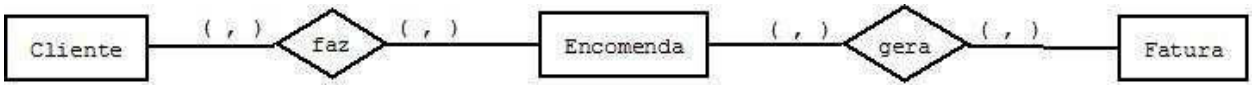
_____
_____
_____
_____
_____
_____

3. O que é uma entidade fraca?

_____
_____
_____
_____
_____
_____

3. Nos diagramas abaixo, para cada entidade defina no mínimo 3 atributos, sua chave primária e a cardinalidade dos relacionamentos:





4. Desenhe o diagrama das situações abaixo, definindo no mínimo 3 atributos para cada entidade, a chave primária e a cardinalidade do relacionamento.

a) Uma universidade tem muitos estudantes e um estudante pode se dedicar a no máximo uma universidade.
b) Uma aeronave pode ter muitos passageiros, mas um passageiro só pode estar em um vôo de cada vez.
c) Um paciente pode ter muitos médicos e um médico muitos pacientes.
d) Uma nação possui vários estados, e um estado, muitas cidades. Um estado só poderá estar vinculado a uma nação e uma cidade só poderá estar vinculado a um estado.

4.6. Auto-relacionamento

Dizemos que existe um auto-relacionamento quando uma entidade se relaciona com ela mesma. Por exemplo, digamos que no exemplo que estamos trabalhando da escola alguns poucos alunos representam grupos de outros alunos em reuniões e assuntos estudantis. Poderíamos criar uma nova entidade representante, porém a entidade representante teria os mesmos atributos do aluno, então ao invés de criar uma nova entidade, podemos simplesmente criar uma relação “representante” e auto-relacionar a entidade aluno conforme a figura 5.15.

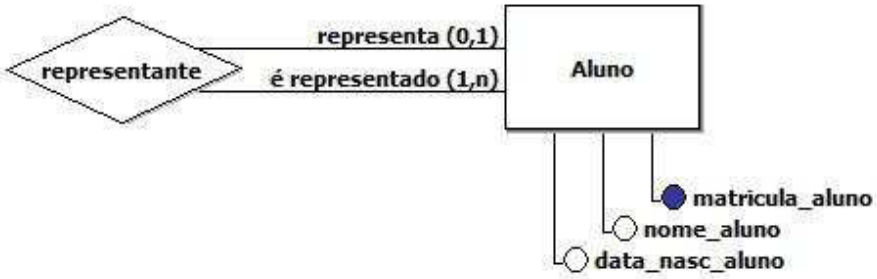
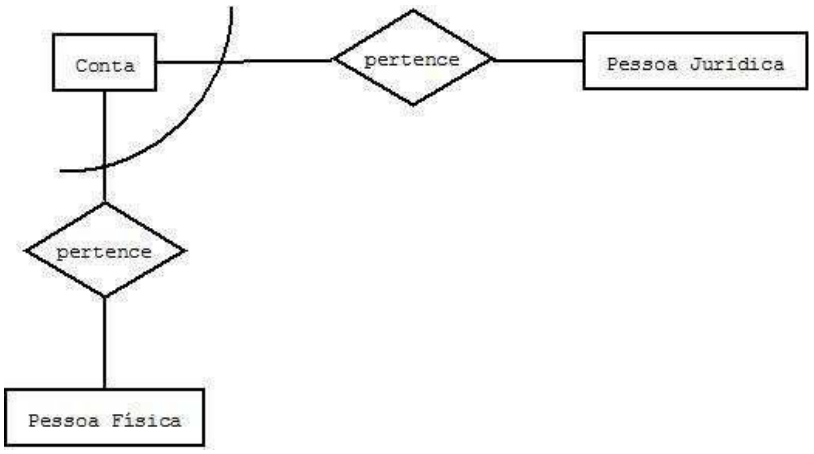


Figura 5.15 – exemplo de auto-relacionamento

4.7. Relacionamento mutuamente exclusivo

Um relacionamento mutuamente exclusivo é aquele em que uma ocorrência de uma entidade pode estar associada com ocorrências de outras entidades, mas não simultaneamente. Por exemplo uma conta pertence uma pessoa jurídica ou pessoa física, nunca aos dois ao mesmo tempo.



4.8. Especialização e Generalização

Especialização consiste na subdivisão de uma entidade mais genérica (ou entidade pai) em um conjunto de entidades especializadas (ou entidades filhas). Isso ocorre quando um conjunto de entidades pode conter subgrupos de entidades com atributos específicos para cada subgrupo. A figura 5.16 apresenta um exemplo de especialização entre as entidades “Pessoa”, “Professor” e “Aluno”.

A especialização é indicada por um triangulo, e as entidades filhas estão relacionadas com a entidade pai por meio deste triangulo.

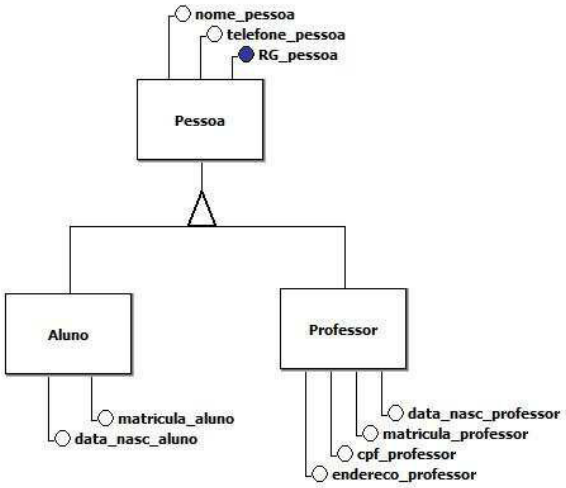


Figura 5.16 – exemplo de especialização

As entidades filhas herdam todos os atributos da entidade pai e, portanto, não se devem repetir os atributos da entidade pai nas entidades filhas. Isso significa que os atributos que aparecem na entidade pai são os atributos que existem em comum entras as entidade filhas. No diagrama da figura 5.16, os atributos da entidade Pessoa (rg_pessoa, telefone_pessoa e nome_pessoa), serão herdados pelas entidades filhas “Professor” e “Aluno”.

Também não é necessário indicar uma chave primária para as entidades filhas. As chaves primárias para as entidades filhas serão definidas no modelo relacional.

Para utilizar uma especialização, deve-se analisar antes se as entidades filhas possuem atributos específicos ou relacionamentos específicos ou ainda outra especialização. Se as entidades filhas não possuírem nem atributos específicos, nem relacionamento específico, ou outra especialização, como mostra a figura 5.18, então ela não deve ser especializada. Neste caso,

dizemos que o modelo deve ser generalizado, ou seja, deve passar pelo processo de generalização.

A generalização é o processo inverso da especialização. Em vez de subdividir a entidade cria-se uma entidade mais genérica e adiciona-se um atributo denominado “tipo”, que identifica o tipo de objeto, como mostra a figura 5.17. O atributo tipo identificará se o telefone é do tipo “celular” ou “residencial”.

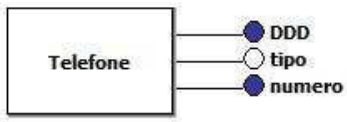


Figura 5.17 – Exemplo de Generalização

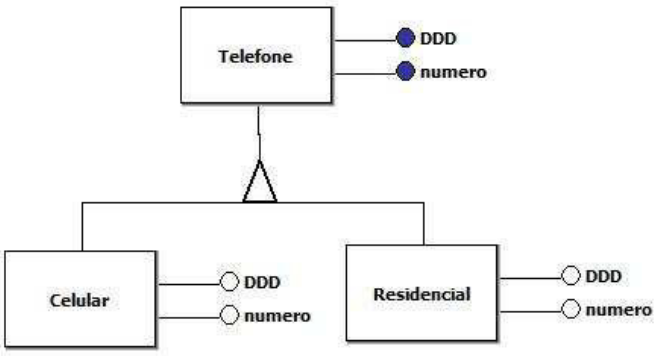


Figura 5.18 – Exemplo errado de especialização

4.9. Agregação

Imagine que tenhamos duas entidades “Cliente” e “Produto” ligadas pelo relacionamento “Compra”. Agora, suponha que tenhamos que modificar esse modelo de modo que seja necessário saber quantas prestações serão pagas em uma compra. Relacionar a entidade “Prestação” com “Cliente” ou com “Produto” não faz sentido, uma vez que as prestações serão referentes a compra efetuada. Sendo assim, a entidade “Prestação” deve se relacionar à entidade “Compra”, como mostra a figura 5.19. O retângulo desenhado em volta do relacionamento indica a agregação.

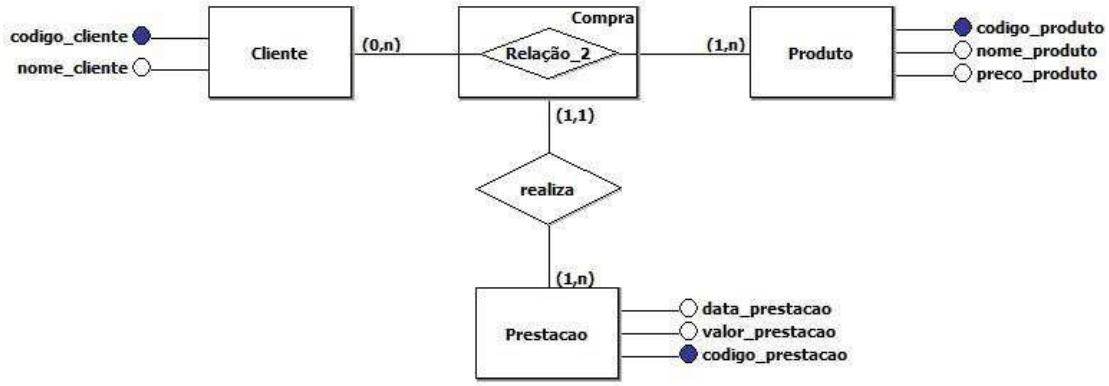


Figura 5.19 – Exemplo de agregação

Podemos também reescrever o modelo sem utilizar agregação. Nesse caso, o relacionamento “Compra” seria transformado em uma entidade que poderia ser relacionada à “Prestação”, como mostra a figura 5.20.

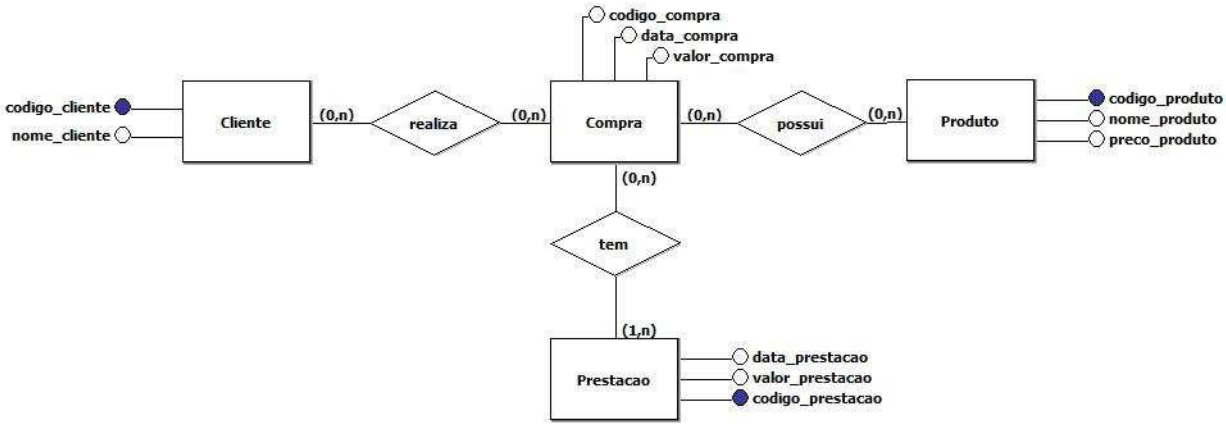
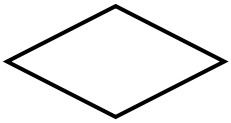


Figura 5.20 – Outra forma de modelar agregação

4.10. Notação do MER

Na apostila, foi apresentada uma notação utilizando a ferramenta brModelo, porém alguns projetistas utilizam outras notações conforme apresentado abaixo:

Atributo Simples ou Monovalorad0

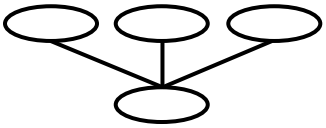


Relacionamento

Atributo Multivalorado

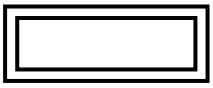
Entidade

Atributo Derivado



Atributo Composto

Atributo Chave



Entidade Fraca

EXERCÍCIOS

1. O que é um auto-relacionamento? Cite um exemplo diferente da apostila que caracteriza este relacionamento.

_____
_____
_____
_____

2. O que é especialização e generalização? Cite um exemplo.

_____
_____
_____
_____
_____
_____

3. O que caracteriza a agregação?

_____
_____
_____
_____

4. O que é um relacionamento mutuamente exclusivo?

_____
_____
_____
_____

5. Faça o DER dos sistemas abaixo:

"Uma concessionária que trabalha com venda de veículos deseja criar uma base de dados para seu negócio. Para qualquer veículo, sabemos o número do chassi, número da placa, cor, ano de fabricação, quilometragem, código da marca, nome da marca, código do modelo, e nome do modelo. Todo carro pertence a um modelo, e este modelo pertence a uma marca. Como a concessionária vende veículos usados de diferentes marcas, é interessante haver um cadastro para as marcas e um cadastro para os modelos. Uma pessoa pode assumir um dos seguintes papeis em relação a concessionária: corretor ou comprador. Sobre o comprador do veículo, tem-se CPF, nome, estado civil, e se for casado, os dados do cônjuge(como nome e CPF). Sobre os corretores, tem-se numero da matricula, nome e data de admissão. Um corretor negocia com um comprador a venda de um veículo. Sobre a venda, são necessárias as seguintes informações: data da venda, valor da venda e valor da comissão do corretor."

5. Modelo Relacional

O modelo Relacional é um modelo lógico, utilizado em bancos de dados relacionais. Ele tem por finalidade representar os dados como uma coleção de tabelas e cada linha (tupla) de uma tabela representa uma coleção de dados relacionados. Neste modelo, começamos a nos preocupar em como os dados devem ser armazenados e em como criaremos os relacionamentos do modelo conceitual. É também nessa etapa que definimos o SGBD que será utilizado, bem como os tipos de dados para cada atributo. O modelo Relacional é definido usando como base o MER.

O Modelo Relacional foi introduzido por Edgar Frank Codd (1970) e tornou-se um padrão para aplicações comerciais, devido a sua simplicidade e

desempenho. É um modelo formal, bastante representativo e ao mesmo tempo bastante simples, foi o primeiro modelo de dados descrito teoricamente.

O Modelo Relacional representa os dados num Banco de Dados como uma **coleção de tabelas** e seus **relacionamentos**. Cada tabela contém um nome e um conjunto de atributos com seus respectivos nomes. Por exemplo, podemos ter a tabela tbAluno, para guardar todos os dados do aluno como: matricula, nome, data de nascimento. Esses dados que são armazenados são os **atributos** do aluno como já visto anteriormente.

No modelo relacional os atributos precisam ter um **domínio** definido, ou seja, precisamos especificar todos os valores possíveis que um atributo pode receber. O domínio de um atributo define qual o tipo de dado e o formato que o dado pode ser armazenado por aquele atributo, por exemplo, no atributo matrícula podem ser guardados valores inteiros, no atributo nome podem ser guardados até 100 caracteres, no atributo data pode ser guardado uma data no formato “dd/mm/aaaa”.

As **tuplas** representam os valores de uma tabela. A Figura abaixo mostra a tabela tbAluno preenchida com valores hipotéticos. Note que as colunas da tabela representam os atributos, enquanto as linhas representam os registros. Se uma tabela não tiver tuplas, ela estará vazia, ou seja, sem dados. Informalmente, as tuplas são também chamadas de registros pelos desenvolvedores.

matricula_aluno	nome_aluno	data_nasc_aluno
100	Ana	12/05/1997
101	João	15/04/1996
102	Maria	22/06/1998
103	José	03/01/1997
104	Marcos	19/03/1996

Para descrever uma tabela no modelo relacional, usamos o nome da tabela seguida dos atributos entre parênteses. Para identificar a chave primária, devem-se sublinhar o(s) atributo(s) correspondente(s) a ela. O tipo de cada atributo também deve aparecer no modelo relacional, como mostra o exemplo abaixo:

```
tbAluno (matricula_aluno: inteiro, nome_aluno: caracter(100),
data_nasc_aluno: data)
```

Podemos também criar um diagrama lógico para representar este modelo. Nesta apostila para a criação do modelo lógico do banco de dados utilizaremos a ferramenta MySQL Workbench, uma ferramenta disponibilizada pela Oracle para modelagem lógica de Banco de Dados e que você pode fazer o download no site: http://dev.mysql.com/downloads/tools/workbench/.

Antes de criarmos nossa representação da tabela aluno utilizando o workbench, vamos primeiramente vamos abrir o programa. A figura abaixo mostra a tela inicial do MySQL Workbench.

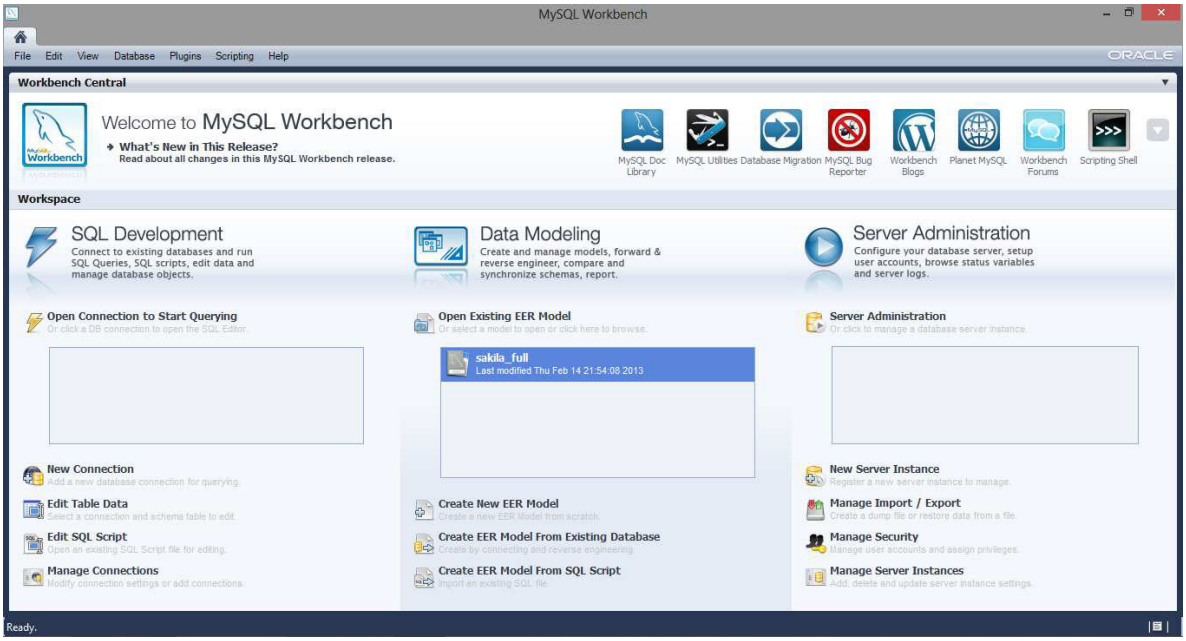


Figura 5.1 – Tela inicial do MySQL Workbench

Para começar, vamos escolher a opção 'Create New EER Model'. Escolhendo essa opção, avançamos para uma próxima tela, na qual temos a opção 'Add Diagram'. Essa tela pode ser visualizada pela Figura 5.2.

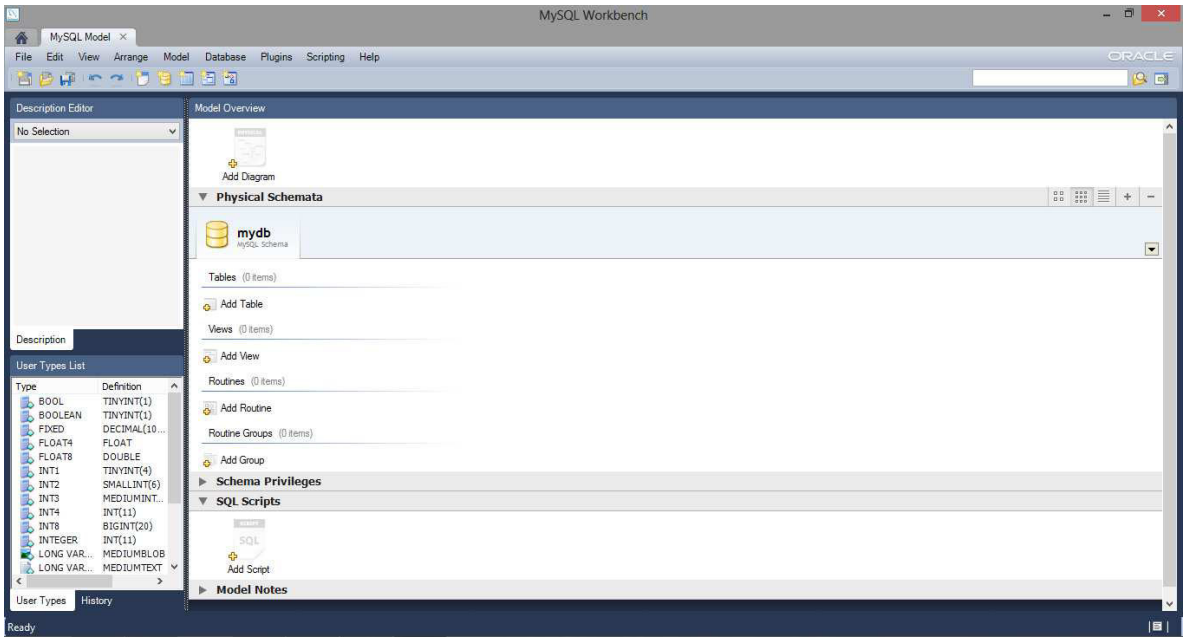


Figura 5.2

Clicando duas vezes em 'Add Diagram', abrimos nossa área para criarmos o Modelo de Relacional.

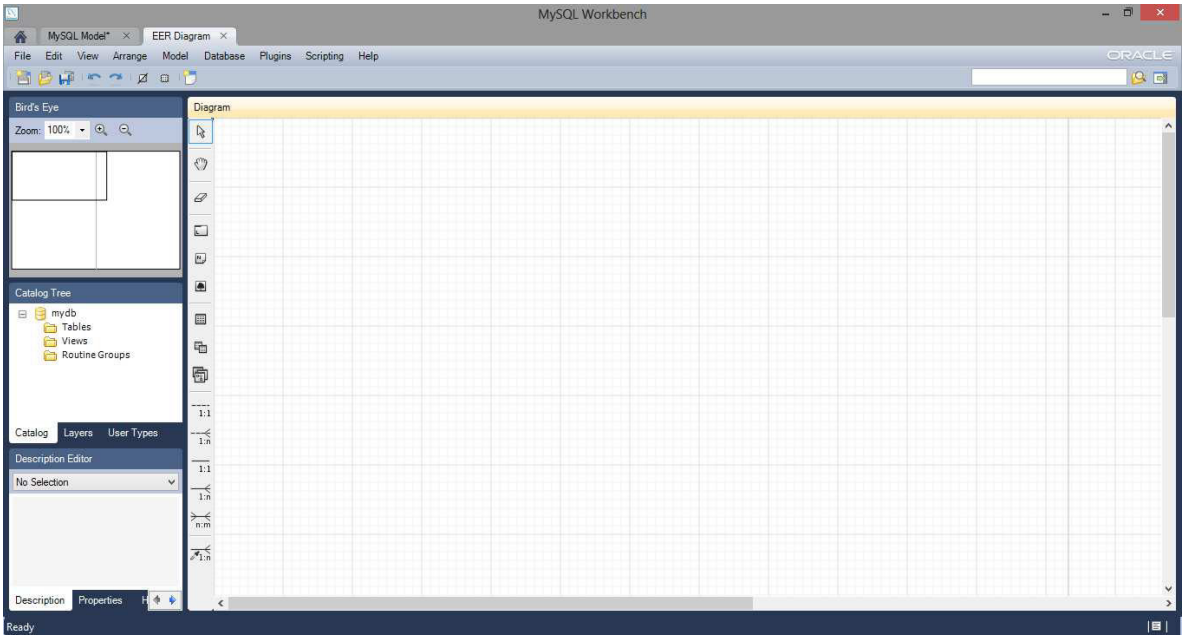


Figura 5.3

Pronto, agora você já pode criar o seu diagrama lógico utilizando o 'MySQL Workbench'. Para exemplificar, vamos mostrar como criar uma tabela e definir seus atributos. Para começar, vamos definir a tabela Aluno usando a ferramenta 'MySQL Workbench'. Na Figura 5.3, temos o ambiente da

ferramenta onde iremos criar nosso Modelo Relacional. O modelo é criado na parte branca que ocupa a maior parte do lado inferior direito da ferramenta.

Na barra mostrada pela Figura 5.4, podemos ver vários ícones. Para criar uma tabela, você deve clicar no ícone “place a new table” e depois clicar na parte branca situada logo a direita da barra mostrada na Figura 5.4.

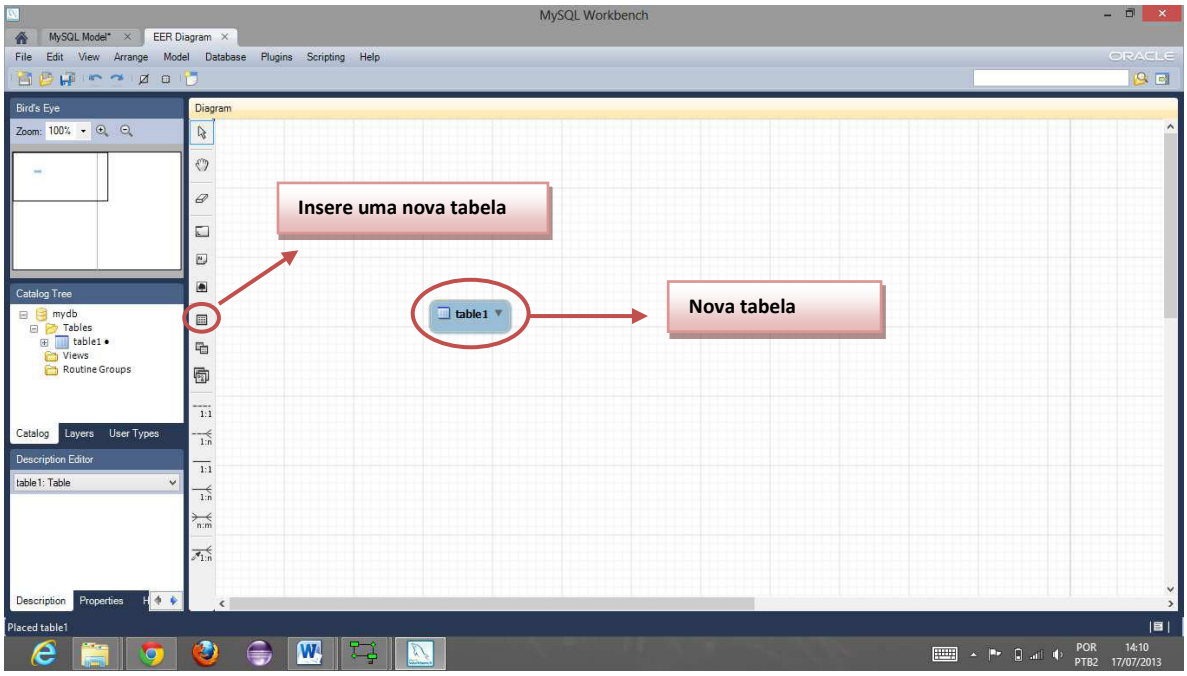


Figura 5.4

Ao clicar com o botão direito do mouse em cima da tabela criada, você irá ver a opção ‘Edit Table’. Quando clicar nessa opção, vai aparecer, na parte inferior da tela, uma aba onde existe a opção ‘Table’. Nessa aba, você irá ver um campo de texto com nome “Name” onde colocamos o nome da tabela. Note que “Table” em inglês equivale à tabela em português. Para o nosso exemplo, você deve fornecer o nome tbAluno. Ao final dessa operação, você deve obter algo parecido com a Figura 5.5.

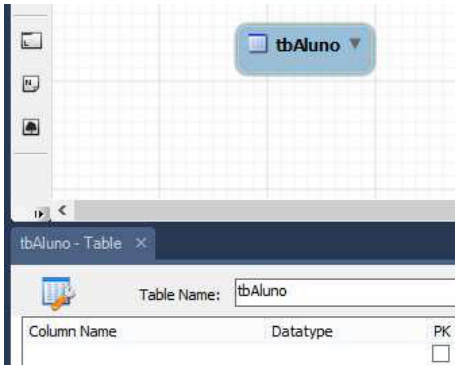


Figura 5.5

Agora, devemos definir os atributos da tabela. Para isso, devemos clicar na aba inferior chamada 'Columns'. Você deve informar o nome do atributo e o tipo em cada linha da tabela. Note que ao clicar na aba 'Columns' o MySQL Workbench cria automaticamente um atributo com nome "idtbAluno" com tipo INT.

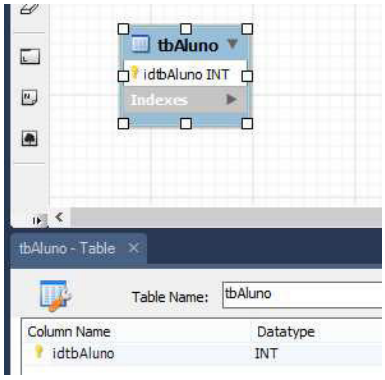


Figura 5.6

Para criar um atributo com nome e tipo desejado, você deve clicar duas vezes em cima da linha onde está o nome do atributo "idtbAluno". Quando o cursor ficar como o mostrado na Figura 5.7, você pode digitar o nome do atributo desejado.

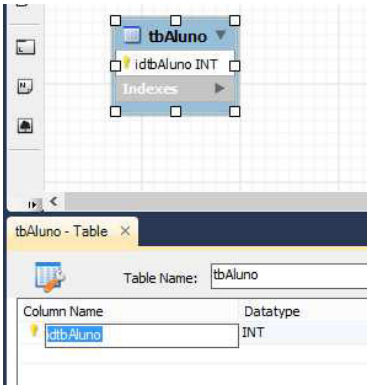


Figura 5.7

Para o nosso exemplo, você deve digitar "matricula_aluno". Para criar outro atributo, você precisa apenas clicar duas vezes na linha abaixo do atributo criado. Tente repetir essa operação para criar os atributos: nome_aluno, data_nasc_aluno.

Depois de definir o nome dos atributos, você precisa definir os tipos de cada um. Para definir o tipo de cada atributo, você deve clicar na coluna "Datatype" de cada atributo. Ao clicar, irá aparecer uma lista com várias opções de tipos suportadas pelo MySQL.

Você deve definir os tipos de modo que o seu modelo fique parecido como o mostrado na Figura 5.8. Note que os tipos dos atributos suportados variam entre os SGBDs. No caso do MySQL, os principais tipos suportados são:

Tipo	Descrição
VARCHAR	Valores no campo VARCHAR são strings de tamanho variável. Você pode declarar um campo VARCHAR para ter qualquer tamanho entre 1 e 255, assim como para campo CHAR. No entanto, diferente de CHAR, valores VARCHAR são armazenados usando apenas quantos caracteres forem necessários, mais 1 byte para gravar o tamanho.
INT	Valores inteiros de -2147483648 a 2147483647.
DECIMAL	O tipo DECIMAL é usado por valores para os quais é importante preservar a exatidão como, por exemplo, dados monetários.
DATA	O tipo DATA é usado quando se necessita apenas do valor da data, sem a parte da hora. MySQL recupera e mostra valores do tipo DATA no formato 'ano-mm-dd'

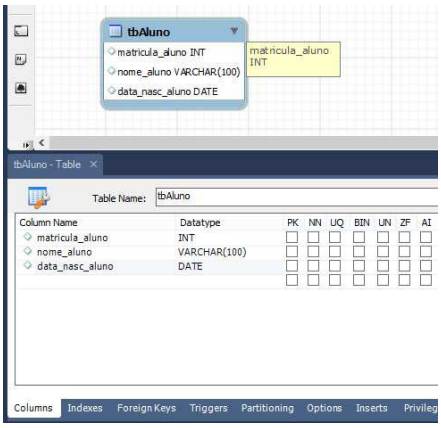


Figura 5.8

Chave Primária

Para inserirmos a chave primária, temos que marcá-la, na Figura 14 há vários quadradinhos ao lado de cada atributo com algumas opções, vamos marcar a opção PK, que é a nossa chave primária. O termo PK vem do inglês “Primary Key”, que em português quer dizer chave primaria.

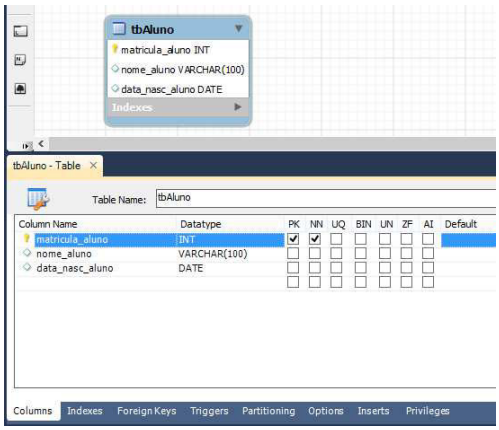


Figura 5.9

Chave Estrangeira

Um conceito muito importante quando se fala em modelo relacional é o conceito de Chave Estrangeira (Foreign Key - FK).

Uma chave estrangeira é um atributo da tabela, que faz referencia a uma chave primaria de outra tabela ou da própria tabela. Suponha que tenhamos as tabelas "Turma" e "Aluno", representadas no modelo relacional a seguir. Observe que a tabela tbAluno possui um atributo código_turma. Esse atributo é chave primária na tabela tbTurma e, portanto, é uma chave estrangeira na tabela tbAluno. O atributo que é chave estrangeira deve ser do mesmo tipo e do mesmo tamanho que sua primária correspondente. É importante deixar explicito a qual tabela a chave estrangeira está fazendo referência.

```
tbTurma (código_turma: inteiro, nome_turma: caracter(5))


tbAluno (matricula_aluno: inteiro, nome_aluno: caracter(100),
data_nasc_aluno: data, código_turma: inteiro)

código_turma referencia tbTurma
```

Uma chave estrangeira **sempre** faz referencia a uma chave primária. A chave estrangeira **nunca** fará referencia a um atributo que não seja uma chave primária.

No Modelo Relacional é a chave estrangeira que especifica o relacionamento entre as tabelas. É através da chave estrangeira que conseguimos descobrir, por exemplo, que o aluno João pertence a turma do 2º ano do Curso Técnico em Informática, como mostrado abaixo.

matricula_aluno	nome_aluno	data_nasc_aluno	código_turma
100	Ana	12/05/1997	1
101	João	15/04/1996	3
102	Maria	22/06/1998	2
103	José	03/01/1997	1
104	Marcos	19/03/1996	3

codigo_turma	nome_turma
1	1º Informática
2	2º Informática
3	3º Informática
4	1º Enfermagem

O valor da chave estrangeira deve ser um valor que já tenha sido cadastrado na chave primária da tabela correspondente, ou um valor nulo. No exemplo acima, não poderia cadastrar uma aluna Maria pertencente à turma de código 9, uma vez que não existe nenhum código 9 cadastrado na tbTurma.

A chave estrangeira garante a **integridade referencial** do modelo relacional, ou seja, ela garante que não se faça referencia a valores que não existam na base de dados, evitando assim inconsistências.

Uma chave estrangeira pode também fazer referencia a uma chave primária dentro da mesma tabela. Isso acontece quando temos um autorelacionamento. Por exemplo:

```
tbAluno (matricula_aluno: inteiro, nome_aluno: caracter(100),
data_nasc_aluno: data, matricula_aluno_representante: inteiro)

matricula_aluno_representante referencia tbAluno
```

Bom, vamos aprender agora como inserimos nossa chave estrangeira utilizando a ferramenta 'MySql Workbench'. Primeiro devemos criar nossa tabela tbTurma. Depois inserir na tabela tbAluno, o atributo que vai fazer referencia a tabela tbTurma. Inicialmente ficaremos como o diagrama como mostrado na figura 5.10

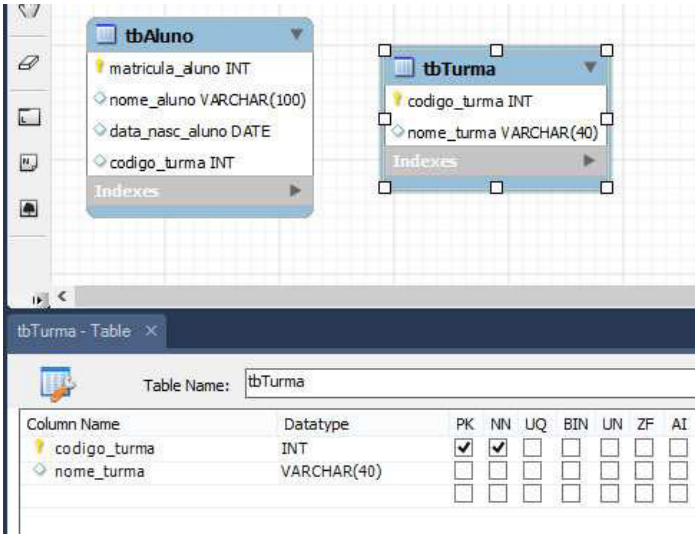


Figura 5.10

Agora com a tabela tbAluno selecionada, clicamos na aba "Foreign Keys" e definir o nome da chave estrangeira no campo "Foreign Key name. Depois disso, você deve informar qual tabela ("Referenced Table") essa chave estrangeira irá se referenciar. No nosso caso, a tabela referenciada é a tabela

tbTurma. Para finalizar, você deve informar qual atributo da tabela tbTurma será utilizada para controlar a integridade da tabela tbAluno. Nesse caso, você deve marcar o atributo código_turma.

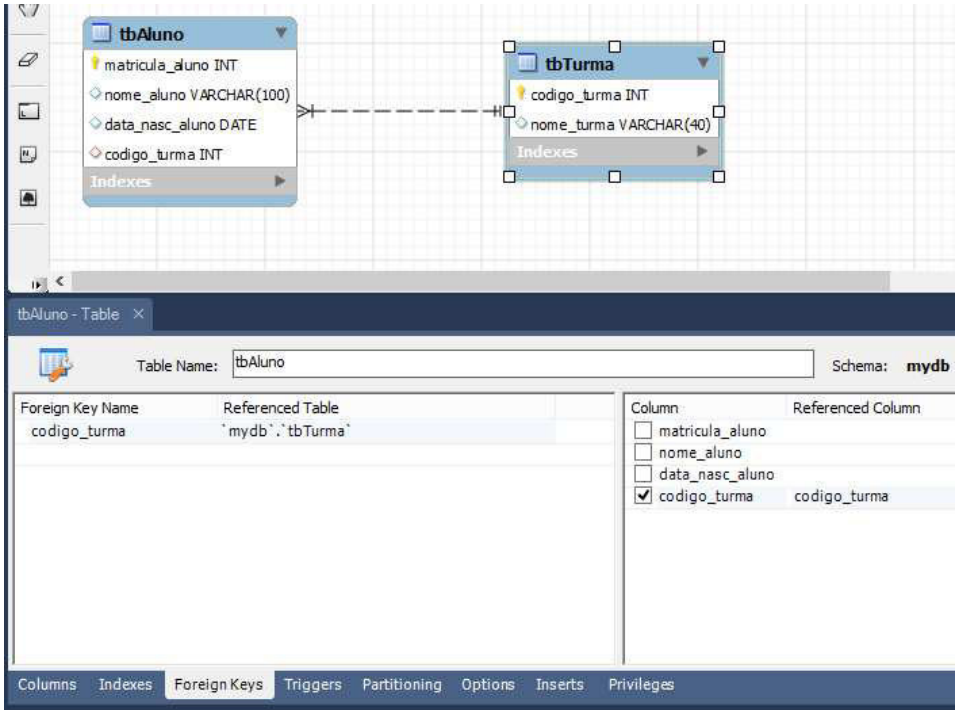


Figura 5.11

Observe que automaticamente o Workbench faz a ligação de uma tabela à outra, caracterizando assim o relacionamento entre elas pela chave estrangeira.

EXERCÍCIOS

1. O que é o Modelo Relacional?

_____
_____
_____
_____
_____

2. O que são as tabelas no modelo relacional? O que representam as colunas e as tuplas?

_____
_____
_____
_____
_____

3. O que você entende por Domínio de um atributo?

_____
_____
_____
_____
_____

4. Para que servem as chaves primárias e estrangeiras?

_____
_____
_____
_____
_____

5. Qual a relação do modelo Entidade-Relacionamento com o Modelo Relacional?

_____
_____
_____
_____
_____
_____

6. Crie as tabelas abaixo, seguindo o modelo relacional.

a. Departamento com atributos: Nome, Número, Localização.
b. Empregado com atributos: Matricula, Nome, Sexo, Endereço, DataNascimento
c. Dependente com atributos: Nome, sexo, DataNascimento e Parentesco.

7. Relacione as tabelas da questão anterior utilizando chave estrangeira.

6. Mapeamento do Modelo Entidade-Relacionamento para o Modelo Relacional

Para transformar do modelo Entidade Relacionamento (ER) para o Modelo Relacional (MR), é necessária a realização de alguns passos. Para mostrar tais passos, iremos utilizar inicialmente a Figura baixo, que descreve um modelo ER parecido com o apresentado nas aulas passadas.

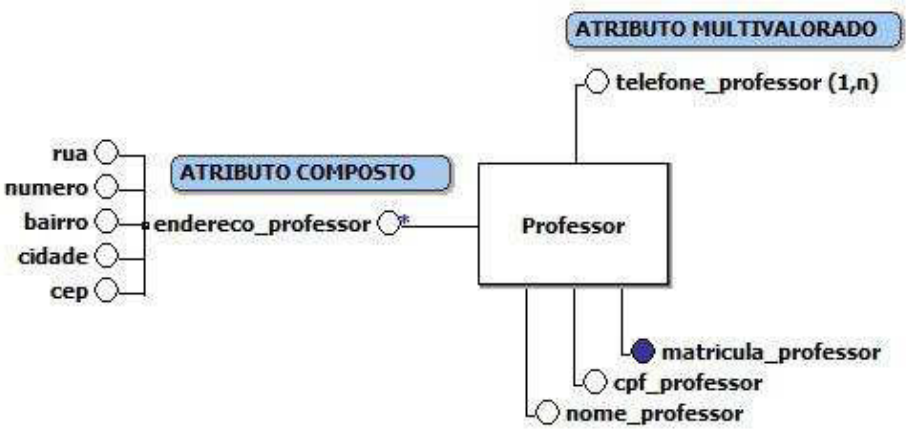


Figura 6.1

6.1. Mapear entidades

O primeiro conceito que devemos mapear do modelo ER para o Modelo Relacional é o conceito de entidade. No Modelo Relacional, entidades são mapeadas para tabelas. Ou seja, cada entidade do modelo ER será mapeado para uma tabela no Modelo Relacional.

Figura 6.2

6.2. Mapear atributos simples

Depois de mapear as entidades, você deve mapear os atributos de uma entidade do modelo ER para os atributos em tabelas no Modelo Relacional. É importante lembrar que no Modelo Relacional os atributos possuem um domínio definido. Ou seja, para definir um atributo você deve indicar no MySql Workbench o tipo de dado que o atributo irá armazenar. Como você sabe, os tipos de dados suportados dependem do SGDB utilizado.

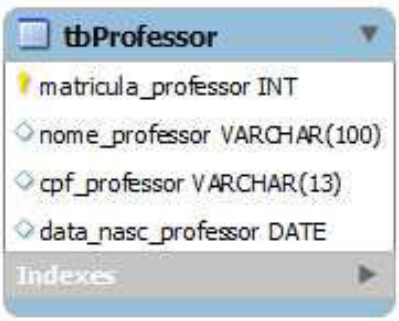


Figura 6.3

6.3. Mapear atributos compostos

Como você deve lembrar, um atributo composto é aquele onde seu conteúdo é formado por vários itens menores. Para mapear atributos compostos no Modelo Relacional, você deve definir na tabela apenas os atributos simples do atributo composto. Ou seja, no nosso exemplo da tabela professor o atributo composto endereço_professor possui cinco atributos simples: Rua, Número, Bairro, Cidade e Cep. Assim, você deve inserir esses atributos na tabela tbProfessor.

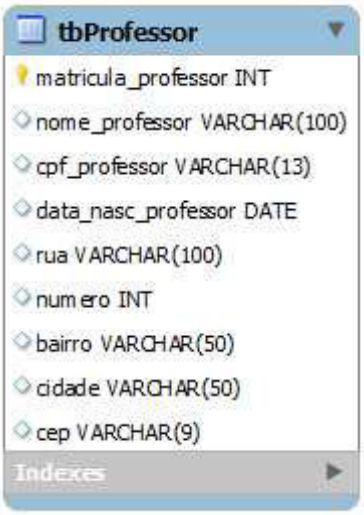


Figura 6.4

6.4. Mapear atributos Chave

No modelo Entidade-Relacionamento da figura 6.1, os atributos chaves são aqueles onde as bolinhas estão pintadas. Atributos chaves no modelo ER são mapeados para chaves primárias no Modelo Relacional. Já vimos anteriormente como criar chave primária no Modelo Relacional.

6.5. Mapear os atributos multivalorados

Os atributos multivalorados no modelo ER são aqueles onde seu conteúdo é formado por mais de um valor. No exemplo da Figura 6.1, o atributo telefone_professor é um exemplo de atributo multivalorado. Nesse caso, um empregado pode ter mais de um telefone. Para fazer o mapeamento do atributo

multivalorado para o Modelo Relacional, é necessário criar uma nova tabela para armazenar os valores do atributo. Desse modo, o mapeamento de atributos multivalorados envolve os subpassos apresentados a seguir.

1. Crie uma tabela no MySql Workbench com o nome do atributo. Por exemplo, a tabela criada poderia se chamar tbTelefone.

Figura 6.5

2. Crie uma chave estrangeira na tabela criada para referenciar a entidade origem do atributo multivalorado. No exemplo da Figura 6.1, a entidade origem do atributo multivalorado é a entidade Professor. Portanto, você deve criar um atributo matrícula_professor na tabela tbTelefone e definir uma chave estrangeira para esse atributo. Ao final deste subpasso, você deve obter um diagrama parecido com a figura abaixo.

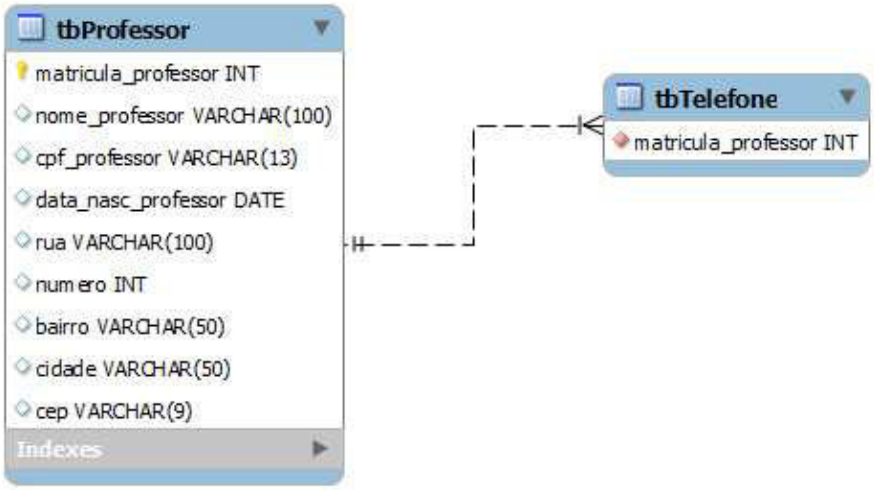


Figura 6.6

3. O próximo subpasso é criar um atributo na tabela recém criada para armazenar os valores do atributo multivalorado. No caso do nosso exemplo, você deve criar um atributo telefone na tabela tbTelefone. Ao final deste subpasso, você deve obter algo parecido com a Figura 6.7.

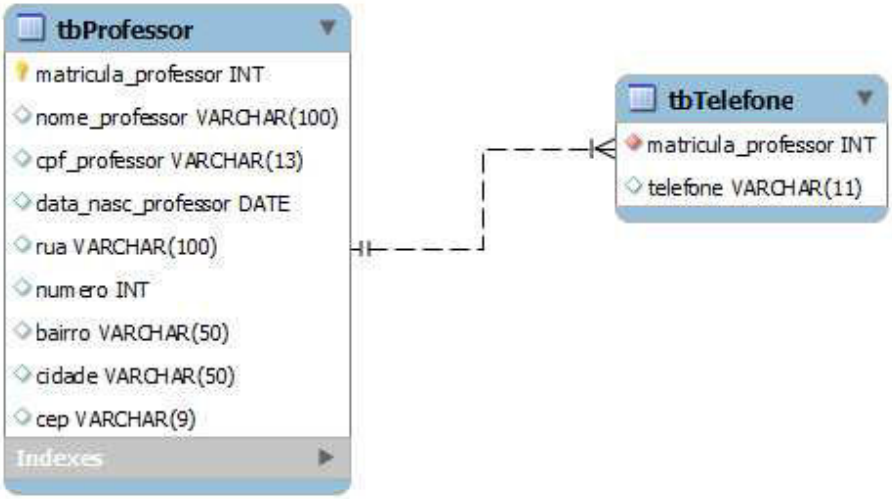


Figura 6.7

4. Note que a tabela tbTelefone não possui chave primária. Como você sabe, todas as tabelas devem ter chaves primárias para garantir a integridade dos dados. A ausência de chave primária iria permitir que um mesmo telefone fosse cadastrado mais de uma vez. Para evitar tal problema, você deve definir os dois atributos da tabela como chave primária. No caso do exemplo, os atributos Matrícula e Telefone serão chave primária da tabela tbTelefone. Ao final deste subpasso, você deve obter algo parecido com a Figura 6.8

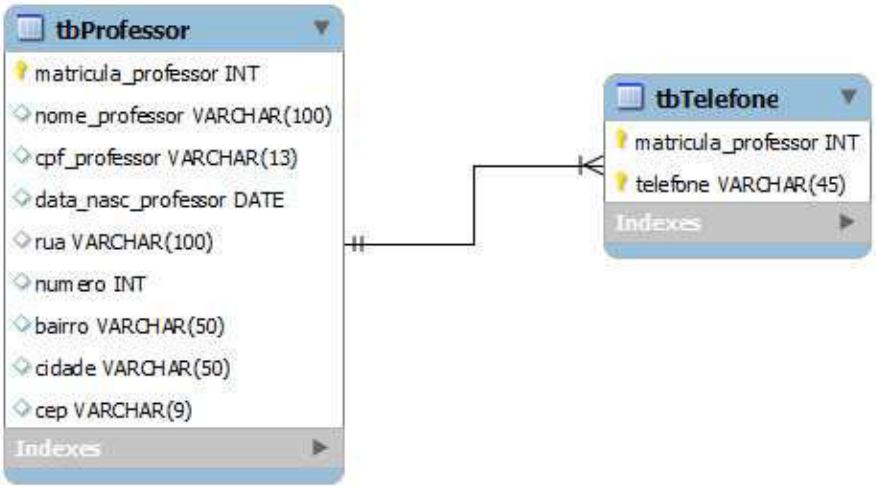
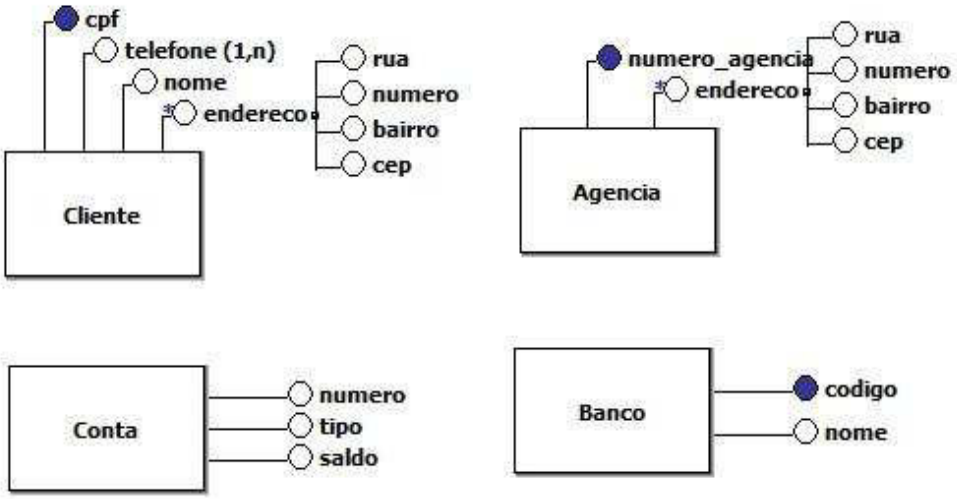


Figura 6.8

EXERCÍCIOS

1. Crie o diagrama relacional das entidades abaixo:



6.6. Mapear Relacionamentos Um para UM

Depois de mapear os atributos do modelo ER, iremos agora mapear os relacionamentos. O primeiro tipo de relacionamento é o um-para-um. A Figura 6.9 mostra um exemplo de relacionamento um-para-um no modelo ER.

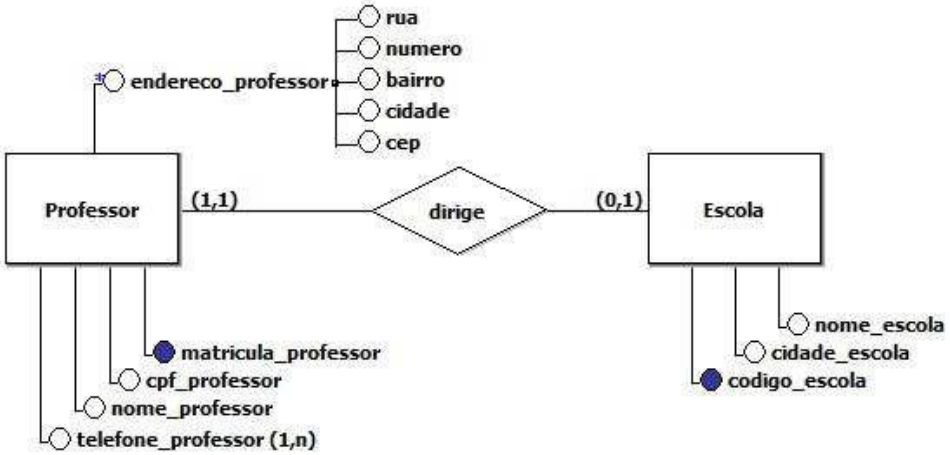


Figura 6.9

Para mapear este relacionamento para o modelo relacional devemos inserir em uma tabela uma chave estrangeira que referencia a chave primária da outra entidade. Para o caso do exemplo, devemos definir na tabela tbEscola uma chave estrangeira que referencia a chave primária da tabela tbProfessor.

Por exemplo, a Figura 6.10 mostra as duas tabelas sem o relacionamento um-para-um.

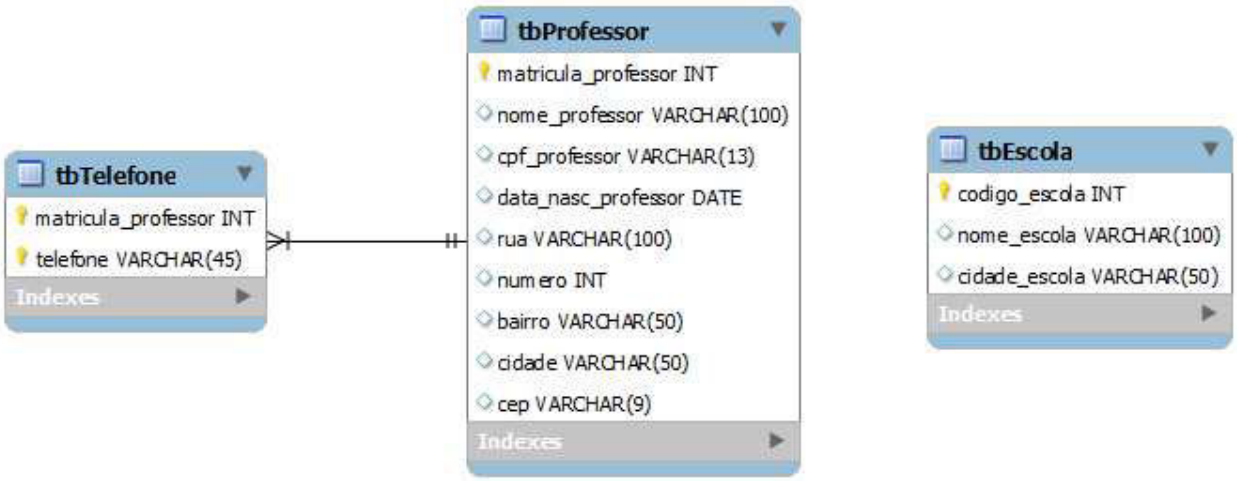


Figura 6.10

Agora, para definir o relacionamento no MySql Workbench, você deve fazer os seguintes subpassos:

1. inserir um atributo na tabela tbEscola com o mesmo nome e tipo do atributo chave primária da tabela tbProfessor, neste caso matricula_professor. Ao final deste subpasso, você deve obter algo parecido com a Figura 6.11.

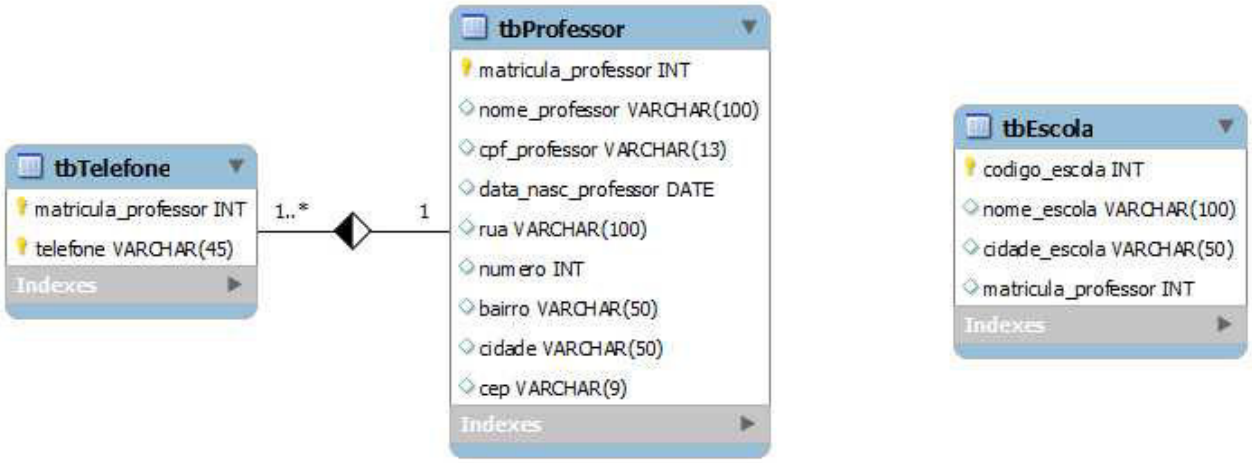


Figura 6.11

2. Definir uma chave estrangeira com o atributo matricula_professor para referenciar a chave primária da tabela Escola.

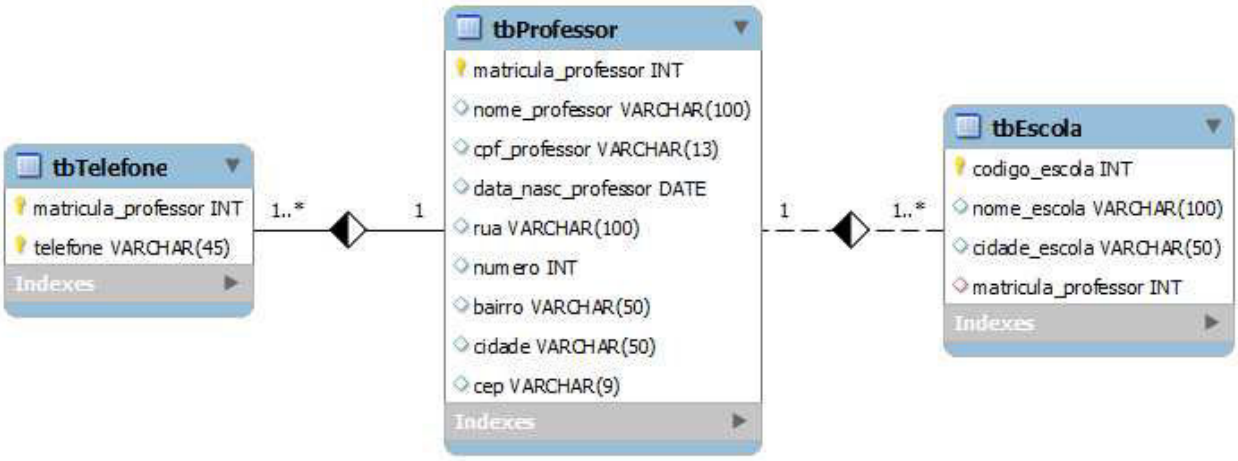


Figura 6.12

3. Note que na Figura 6.12 aparece a cardinalidade de 1..* e 1. Ou seja, o MySql Workbench cria por padrão um relacionamento um-para-muitos. Portanto, devemos ajustar o relacionamento criado através de um clique com o botão direito do mouse em cima das linhas que ligam as duas tabelas. Escolha a opção “Edit Relationship”. Na aba “Foreign key” marque a opção “Cardinality” para “One-to-one 1:1”. Ao final deste subpasso, você deve obter algo parecido com a Figura 6.13.

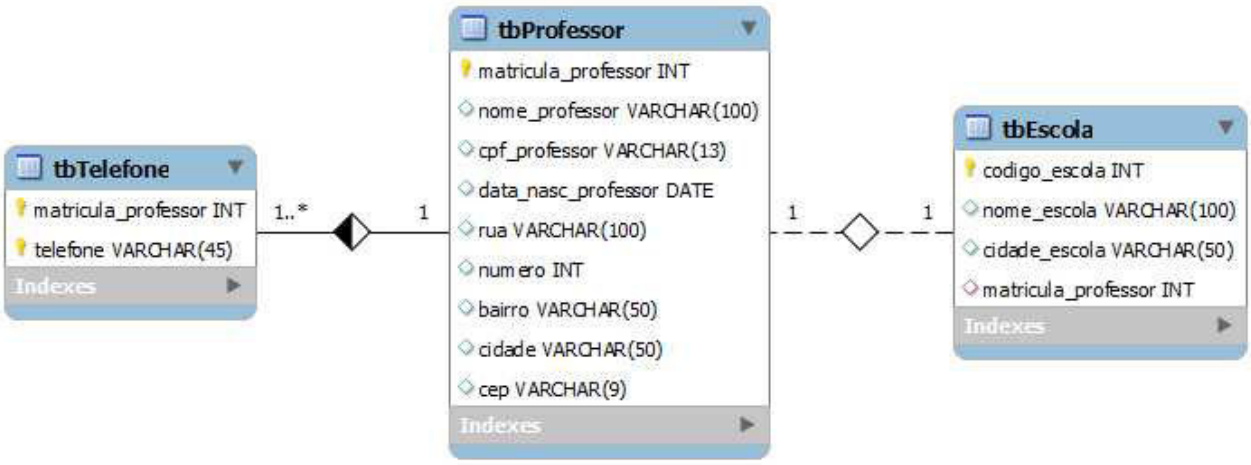
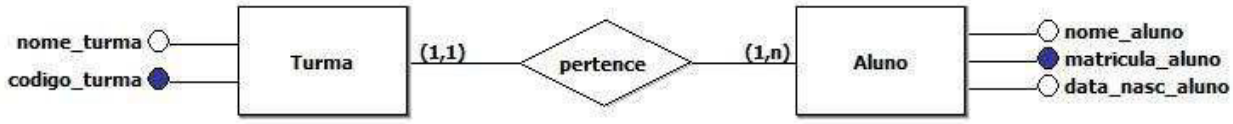


Figura 6.13

6.7. Mapear Relacionamentos Um para Muitos

Como já vimos no capitulo anterior o relacionamento um-para-muitos é usado quando uma entidade A pode se relacionar com uma ou mais entidades



B. A Figura 6.14 mostra um relacionamento um-para-muitos no modelo ER. Agora veremos como mapear tal relacionamento para o modelo Relacional.

Para definir o relacionamento no MySql Workbench você deve primeiro mapear as entidades Aluno e Turma para o modelo Relacional, como já

Figura 6.14

mostramos anteriormente. Depois você deve fazer os seguintes subpassos:

1. inserir um atributo na tabela tbAluno com o mesmo nome e tipo do atributo chave primária da tabela tbTurma, neste caso o atributo código_turma. Uma dúvida que você pode ter é como saber onde criar (qual tabela) o novo atributo. A resposta para esta dúvida é: sempre que você tiver uma relação um-para-muitos ou 1:N, a entidade que estiver do lado N deverá receber o novo atributo com a chave estrangeira. No caso do exemplo, a entidade que está do lado do N é a entidade Aluno.

Figura 6.15

2. Definir uma chave estrangeira com o atributo código_turma para referenciar a chave primária da tabela tbTurma.

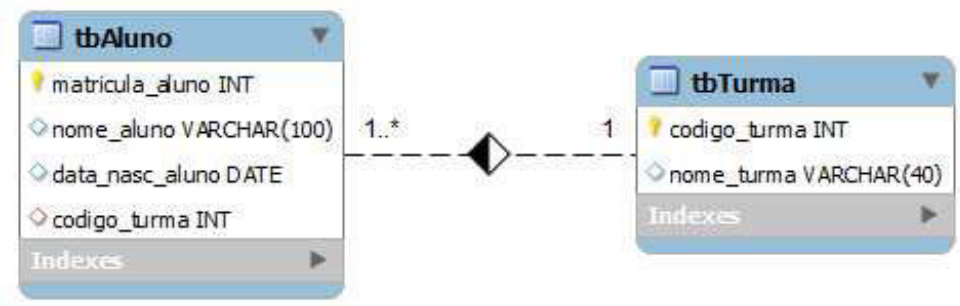


Figura 6.16

6.8. Mapear Relacionamento Muitos para Muitos

Neste passo você deve primeiro lembrar que o relacionamento muitos-para-muitos é usado quando várias entidades A se relacionam com várias entidades B. No exemplo da Figura 6.17 temos o relacionamento muitos-para-muitos entre Professor e Turma. Nela, a entidade Professor pode lecionar várias Turmas. Pelo outro lado, cada Turma pode possuir mais de um

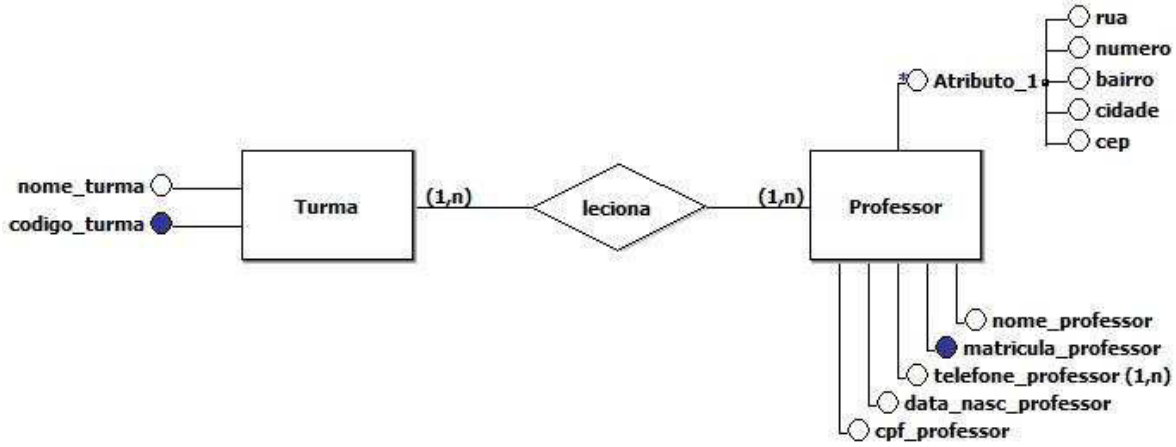
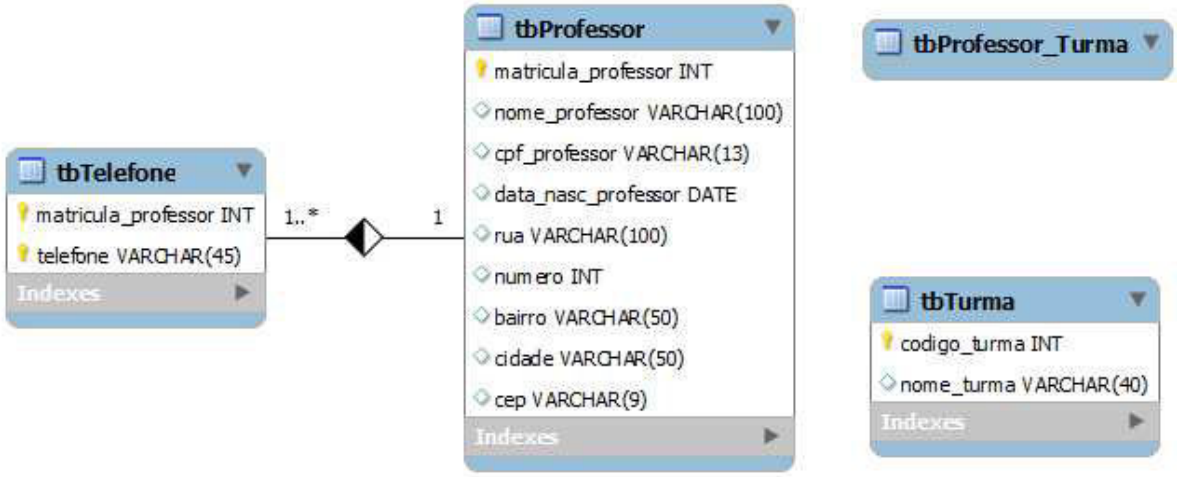


Figura 6.17

Professor.

Para definir o relacionamento muitos-para-muitos no modelo Relacional, usando o MySql Workbench, você deve primeiro mapear as entidades Professor e Turma para o modelo Relacional, como já mostramos anteriormente. Depois você deve fazer os seguintes subpassos:

1. criar uma nova tabela para representar o relacionamento muitos-para-muitos. No nosso exemplo, você poderia criar uma tabela com o nome tbProfessor_Turma, ou tbleciona, para representar o relacionamento no modelo ER. Ao final deste subpasso, você deve obter algo parecido com



a Figura 6.18.

2. inserir como chave estrangeira na tabela recém-criada as chaves primárias das entidades participantes. No caso do exemplo, você deve inserir como chave estrangeira os atributos matricula_professor e código_turma, que são as chaves primárias das tabelas tbProfessor e tbTurma, respectivamente. Ao final deste subpasso, você deve obter algo parecido com a Figura abaixo.

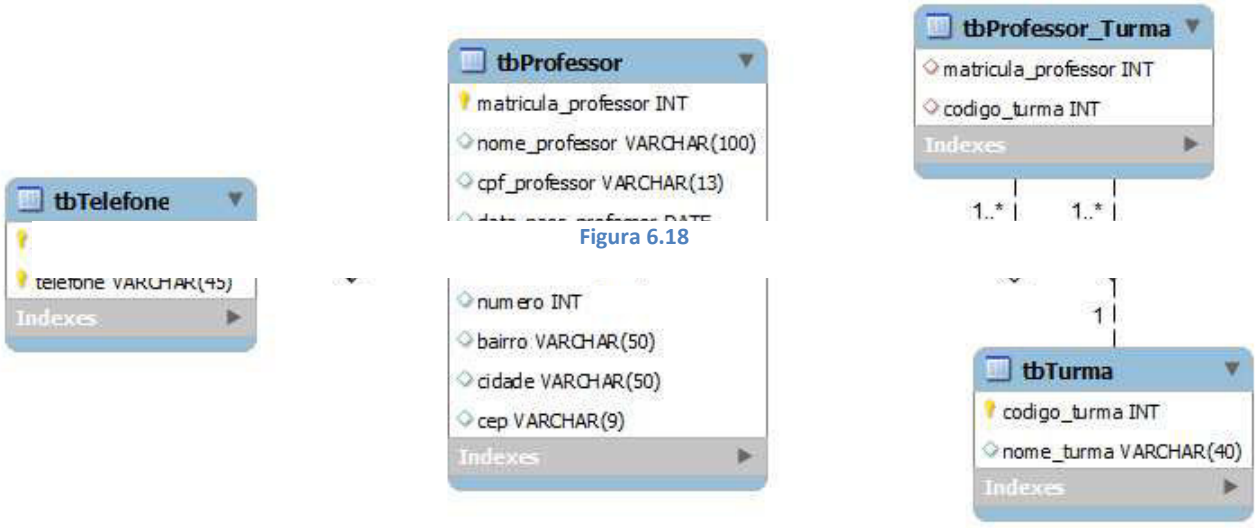
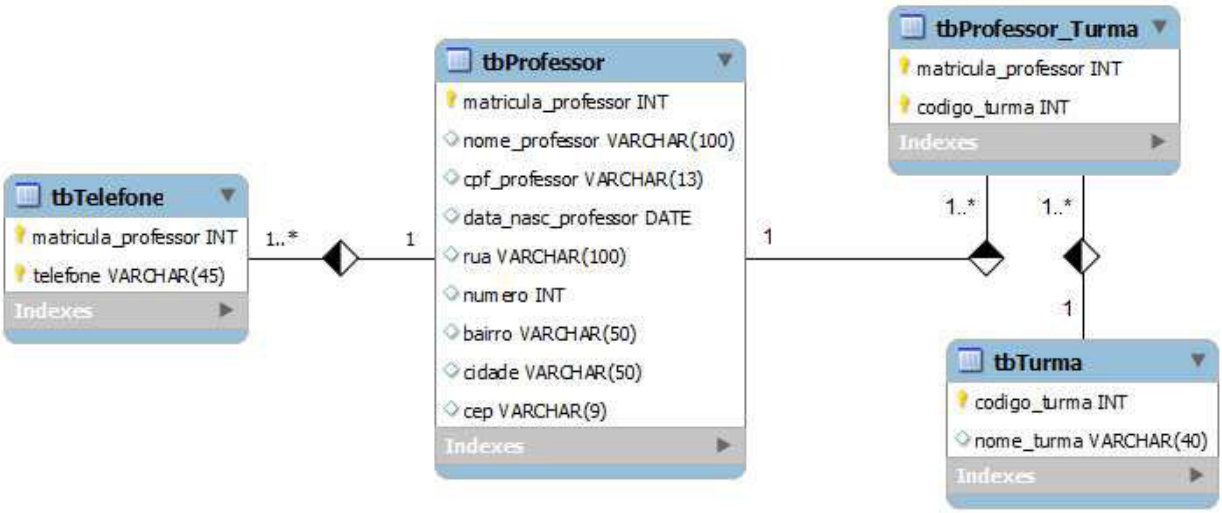


Figura 6.18

Figura 6.19

3. O último passo consiste em definir a chave primária da tabela criada para representar o relacionamento muitos-para-muitos. A chave primária da tabela criada será a composição das chaves primárias das tabelas participantes da relação. No caso do exemplo, você deve definir como chave primária da tabela tbProfessor_Turma os campos matricula_professor e código_turma. Ao final deste subpasso, você deve



obter algo parecido com a Figura abaixo.

EXERCÍCIOS

1. Passe os diagramas entidade-relacionamento abaixo para o modelo

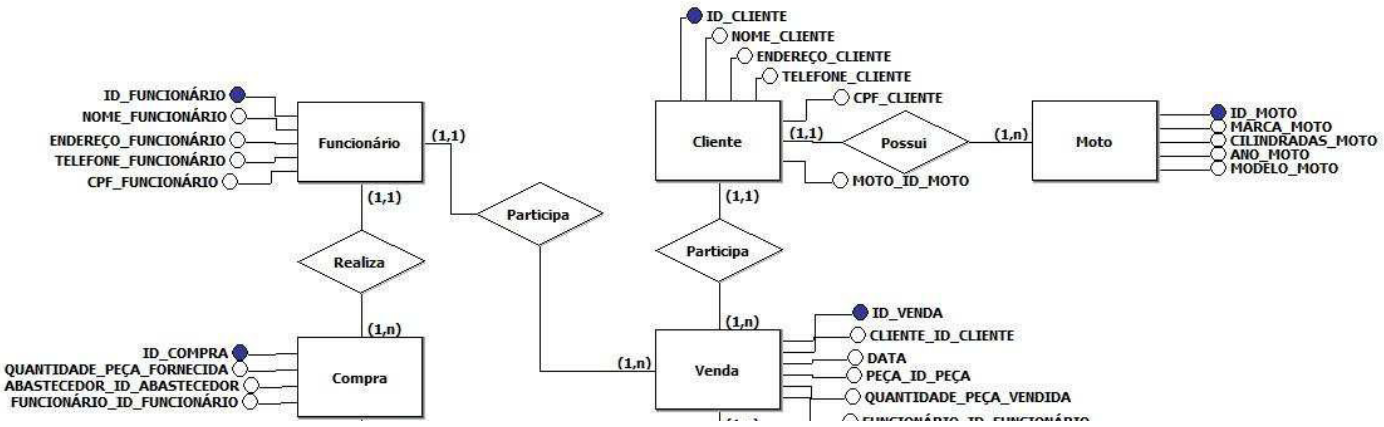
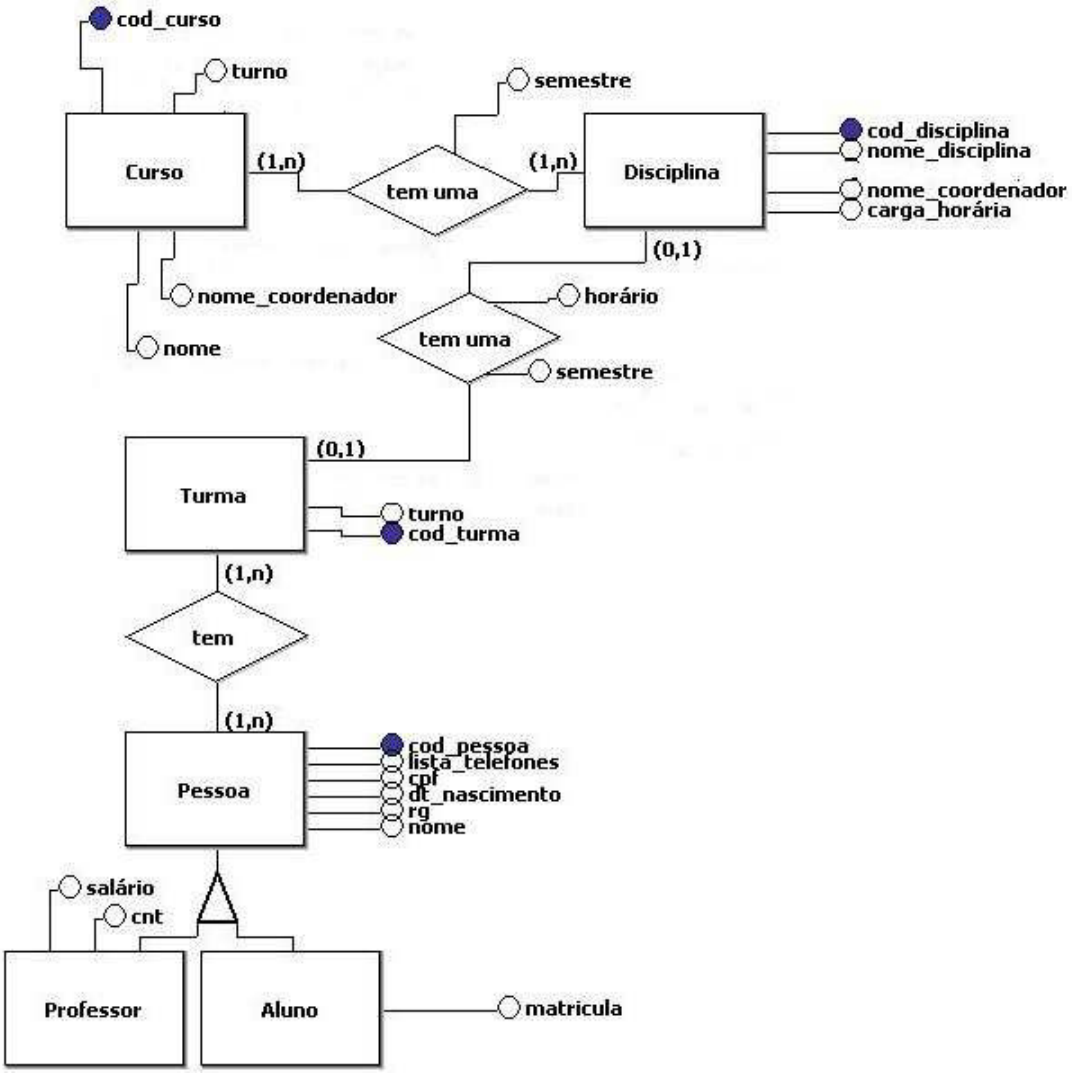


Figura 6.20

relacional



7. Dicionário de Dados

Após o modelo relacional ter sido descrito ou diagramado, é necessário criar o Dicionário de Dados, que tem por objetivo descrever as propriedades de uma tabela, sua estrutura física, e as restrições que cada atributo possui. Assim, o desenvolvedor que irá implementar o banco de dados saberá exatamente como a base de dados deve ser criada.

No Dicionário de dados, cada tabela do modelo relacional deverá ser descrita e deverá conter os seguintes campos: Nome do atributo, Descrição do atributo, tamanho, tipo, e restrições (valor nulo, regra de domínio, chaves, valor default, unique).

A tabela abaixo apresenta um modelo de dicionário de dados para a tabela tbAluno.

Nome	Descrição	Tipo	Tamanho	Nulo	Regra de Domínio	PK	Default	Unique
matricula_aluno	Armazena a matricula do aluno	Numérico	5	Não	-	-	-	Não
RG_aluno	Armazena o RG do aluno	Caracter	11	Não	-	-	-	Sim
nome_aluno	Armazena o nome do aluno	Caracter	100	Não	-	-	-	Não
data_nasc_aluno	Armazena a data de nascimento do aluno	Data	-	Não	-	-	-	Não
Cidade_aluno	Armazena a cidade do aluno	Caracter	20	Sim	-	-	Fortaleza	Não
matricula_aluno_repres	Armazena a matricula do aluno representante	Numérico	5	Sim	-	-	-	Não
Código_turma	Armazena o código da turma do aluno	Inteiro	-	Não	-	FK que referencia tbTurma	-	Não
sexo_aluno	Armazena o sexo a que o aluno pertence	Caracter	1	Não	(M)Masculino (F)Feminino	--	-	Não

É importante fazer algumas observações sobre a criação do dicionário de dados com relação as restrições dos atributos da tabela

1. Alguns tipos de dados não é possível definir o tamanho, como é o caso da data que já é predefinido pelo SGBD.
2. A restrição nulo define-se quando o atributo permite ou não um valor nulo, ou seja, define se o atributo será obrigatório ou não.
3. A Regra de Domínio define quais valores serão permitidos cadastrar para um atributo. No exemplo acima o atributo sexo, só é permitido guardar “M” para masculino e “F” para feminino.
4. As restrições chave permitem identificar a chave primária (PK) e as chaves estrangeiras da tabela. É importante indicar a referencia da chave estrangeira.
5. A restrição default permite que seja inserido um valor padrão caso o usuário não diga nada para o campo.
6. A restrição de unicidade, ou seja, não é permitido dois RGs iguais nos registros da tabela. Essa restrição só é utilizada para atributos que não são chave primária, tendo em vista que uma chave primária por si só já é única.

8. Normalização de Dados

8.1. Principais problemas e Anomalias nos Bancos de Dados Relacionais

O Projeto de um Banco de Dados como já percebemos não uma tarefa simples, e pode apresentar muitos problemas se não for projetado de forma correta, ocasionando redundância de dados, ou seja, informações repetidas e que irão consumir espaço de armazenamento e dificultar a atualização do banco, gerando assim inconsistência. Essas anomalias geralmente acontecem na inclusão, alteração, ou exclusão de registros. Observe as tabelas abaixo:

Cliente_id	Nome	Endereço	Cidade
1	João Silva	Rua Uruguaiana	Porto Velho
2	Maria Francisca	Rua México	Cacoal
3	Antonio José	Rua Piau	Porto Velho

Pedido_id	Preço	Data	Endereço	Nome	Cidade
8	23	01/05/05	Rua Uruguaiana	João Silva	Porto Velho
9	45	06/08/05	Rua México	Maria Francisca	Cacoal
10	67	04/07/05	Rua Uruguaiana	João Silva	Porto Velho

- **Anomalia de inclusão:** Este tipo de anomalia ocorre toda vez que quisermos adicionar um pedido, teremos que digitar novamente o nome, o endereço e a cidade de quem fez o pedido. Se um campo for digitado incorretamente ocorrerá inconsistência.
- **Anomalia de atualização:** Ocorrem quando, em um banco de dados mal projetado, ao modificarmos uma informação, ocorre inconsistência nos dados. Se quisermos modificar uma informação, pode ser que precisemos modifica-la várias vezes, como o nome 'João Silva'. Se quisermos modificar seu nome para 'João da Silva' teremos que modificá-lo tantas vezes quanto for o numero de seus pedidos. Se isso não for feito, ocorrerá inconsistência nos dados, ou seja, uma anomalia de modificação.
- **Anomalia de Exclusão:** Ocorrem quando ao excluir registros perdemos informações importantes para o banco, por exemplo, digamos que precisássemos excluir os registros de pedido 8 e 10, automaticamente perderíamos todas as informações do cliente João Silva.

8.2. Porque Normalizar?

Uma das formas de corrigir as anomalias em banco de dados é utilizar a normalização de dados. A normalização de dados é uma série de passos que se seguem no projeto de um banco de dados, que permitem um armazenamento consistente e um eficiente acesso aos dados em bancos de dados relacionais. Esses passos reduzem a redundância de dados e as chances dos dados se tornarem inconsistentes.

O processo de normalização geralmente é aplicado quando temos uma base de dados que foi criada antes da existência de um banco de dados relacional, ou foi desenvolvida sem considerar a existência de um banco de dados relacional. Ela também pode ser aplicada ao final do processo de modelagem de um banco para conferir se o nosso modelo de dados está normalizado e, caso não esteja, pode-se normaliza-lo antes da implementação da base de dados do SGBD.

O processo de normalização é constituído por um conjunto de formas normais. As formas normais especificam critérios que definem quando uma tabela está bem estruturada ou não. Assim, para saber se uma tabela está bem estrutura, você deve verificar se a estrutura da tabela satisfaz todas as formas normais.

8.3. Primeira Forma Normal (1FN)

Uma tabela está na Primeira Forma Normal (1FN) se e somente se todos os atributos contiverem apenas dados atômicos. Ou seja, cada atributo pode ter apenas um valor por registro (tupla).

A tabela abaixo não está na 1FN pois porque o atributo Telefones possui mais de um telefone por registro (tupla). Por exemplo, a agência 1524 possui três telefones.

NumAg	Endereço	Telefones
1524	Prudente de Morais, 12, RN	3605-5223, 3605-5141, 3605-5142
1550	Hermes da Fonseca, 15, RN	4225-5889, 4225-5890
2051	Eng. Roberto Freire, 20, RN	5223-8556, 5223-8557

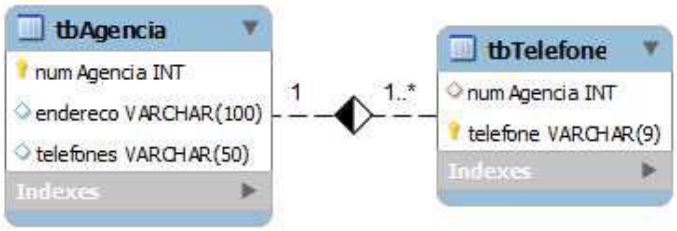
Para adequar uma tabela que não está na 1FN é necessário realizar os seguintes passos:

1. Criar uma tabela para conter os dados do atributo compostos, neste caso então deveríamos criar uma nova tabela tbTelefone;
2. Criar na nova tabela um atributo para conter o atributo composto da tabela original, neste caso criar um atributo telefone, na nova tabela tbTelefone;
3. Criar na nova tabela um atributo para conter a chave primária da tabela original, podemos criar um atributo numero_agencia na tabela tbTelefone;
4. Definir uma chave estrangeira para garantir a relação entre a nova tabela e a tabela original;
5. Definir a chave primária da nova tabela;
6. Remover o atributo não atômico da tabela original.

Ao aplicar a 1FN teríamos então:

NumAg	Endereço
1524	Prudente de Morais, 12, RN
1550	Hermes da Fonseca, 15, RN
2051	Eng. Roberto Freire, 20, RN

NumAg	Telefones
1524	3605-5223
1524	3605-5141
1524	3605-5142
1550	4225-5889
1550	4225-5890
2051	5223-8556
2051	5223-8557



8.4. Segunda Forma Normal (2FN)

Uma tabela está na Segunda Forma Normal (2FN) se e somente se ela estiver na 1FN e todos os atributos não chave primária puderem ser obtidos da combinação de todos os atributos que formam a chave primária.

A 2FN só é aplicável para tabelas que possuem uma chave primária composta e que, além disso, tenham outros atributos que não façam parte da chave primária.

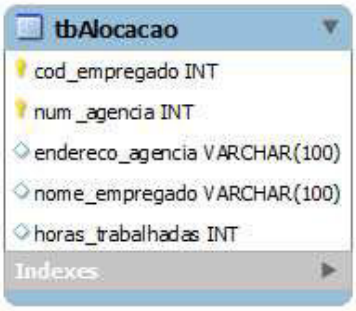
Para ilustrar o significado da 2FN, vamos olhar para a Figura xxx. Nesta figura, a tabela tbAlocação armazena as horas trabalhadas por funcionários

temporários em determinadas agências de um banco. A tabela possui a sua chave primária formada pelos atributos cod_empregado e numero_agencia. Além destes dois atributos, a tabela Alocação possui mais três atributos que não fazem parte da chave primária. São eles: endereço_agencia, nome_empregado e horas_trabalhadas.

Como você viu anteriormente, para estar na 2FN todos estes três atributos não chave (endereço_agencia, nome_empregado e horas_trabalhadas) devem ser obtidos através dos dois atributos chaves (cod_empregado e numero_agencia). Se for possível obter um atributo não chave primária através de apenas um dos atributos chave primária, então a tabela não está na 2FN. É o que acontece com os atributos nome_empregado, que pode ser obtido apenas de cod_empregado, e um outro exemplo é o caso do atributo agencia_endereco, que pode ser obtido apenas do atributo numero_agencia.

Assim a tabela tbAlocação não está na 2FN porque possui pelo menos um atributo que pode ser obtido de apenas um dos atributos que formam a chave primária.

O fato de que uma tabela não está na 2FN pode gerar as anomalias como para modificar o endereço da agência de número 1550 você teria que modificar dois registros: o segundo e quarto registro, como mostra na tabela.



Cod_empregado	Num_agencia	Agencia_endereco	Nome_empregado	Horas_trabalhadas
12	**1524**	Prudente de Morais, 12	Luiz	20
12	**1550**	Hermes da Fonseca, 15	Luiz	18
15	**1524**	Prudente de Morais, 12	Ricardo	10
15	**1550**	Hermes da Fonseca, 15	Ricardo	15

Para adequar uma tabela que não está na 2FN é necessário fazer os seguintes passos:

1. Criar duas novas tabelas para armazenar os dados dos campos redundantes, onde seus valores apresentam repetição de valores, neste caso criar as tabelas tbEmpregado e tbAgencia;

2. Remover os campos com valores redundantes da tabela original, neste caso retirar o endereço_agencia e colocar na tabela tbAgencia, e nomeEmpregado e colocar na nova tabela tbEmpregado;
3. Criar chaves primárias nas novas tabelas criadas com base na chave primária da tabela original, neste caso para a tabela tbEmpregado a chave primária seria cod_empregado, e para a tabela tbAgencia seria num_agencia;
4. Criar relações um-para-muitos entre as novas tabelas criadas e a tabela original.

Após a conclusão destes passos obteríamos este resultado:

Cod_empregado	Num_agencia	Horas_trabalhadas
12	**1524**	20
12	**1550**	18
15	**1524**	10
15	**1550**	15

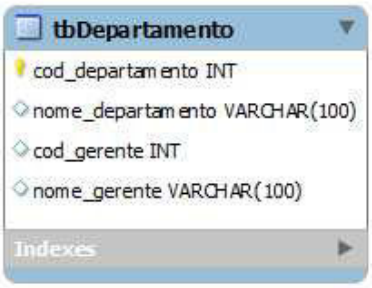
Num_agencia	Agencia_endereco
1524	Prudente de Morais, 12
1550	Hermes da Fonseca, 15
1524	Prudente de Morais, 12
1550	Hermes da Fonseca, 15

Cod_empregado	Nome_empregado
12	Luiz
12	Luiz
15	Ricardo
15	Ricardo

8.5. Terceira Forma Normal 3FN

Uma tabela está na Terceira Forma Normal (3FN) se e somente se ela estiver na 1FN e na 2FN e todos os atributos não chave primária puderem ser obtidos somente através da chave primária.

A tabela abaixo está na 1FN porque todos os seus atributos são atômicos, e está na segunda forma normal porque não possui chave composta. No entanto temos o atributo nome_gerente que depende do atributo código_gerente que não é chave e nem faz parte da chave primária.

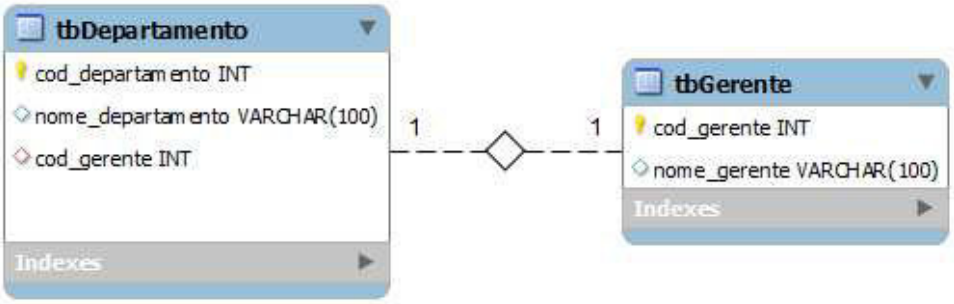


Cod_departamento	Nome_departamento	Cod_gerente	Nome_gerente
1	Vendas	101	Marcos
2	Recursos Humanos	102	Maria
3	Estoque	103	João
4	Financeiro	104	Matheus

Para que a tabela obedeça a 3FN, devemos criar uma nova tabela tbGerente que irá conter o atributo que depende de outro atributo não chave da tabela original, neste caso nome_gerente, mais o atributo do qual ele depende, neste caso o código_gerente. A figura abaixo mostra como ficaria este modelo.

Cod_departamento	Nome_departamento	Cod_gerente
1	Vendas	101
2	Recursos Humanos	102
3	Estoque	103
4	Financeiro	104

Cod_gerente	Nome_gerente
101	Marcos
102	Maria
103	João
104	Matheus



EXERCÍCIOS

1. Crie o dicionário de dados dos diagramas da atividade anterior.

2. Explique cada uma das restrições utilizadas na criação do dicionário de dados.

_____
_____
_____
_____
_____
_____

2. Explique o que é redundância de dados.

_____
_____
_____
_____
_____

3. Cite as principais características das anomalias de inserção, remoção e atualização.

_____
_____
_____

4. O que é normalização de dados, e qual sua finalidade?

_____
_____
_____

5. Dada as tabelas abaixo, verifique se estas tabelas estão normalizadas. Caso não estejam, faça o processo de adequação destas tabelas.

a)

CodProj	Tipo	Descr	CodEmp	Nome	Cat	Sal	DatIni	TempAl
LSC001	Novo Deseny.	Sistema de Estoque	2146	João	A1	4	1/11/91	24
LSC001	Novo Deseny.	Sistema de Estoque	3145	Silvio	A2	4	2/10/91	24
LSC001	Novo Deseny.	Sistema de Estoque	6126	José	B1	9	3/10/92	18
LSC001	Novo Deseny.	Sistema de Estoque	1214	Carlos	A2	4	4/10/92	18
LSC001	Novo Deseny.	Sistema de Estoque	8191	Mário	A1	4	1/11/92	12
PAG02	Manutenção	Sistema de RH	8191	Mário	A1	4	1/05/93	12
PAG02	Manutenção	Sistema de RH	4112	João	A2	4	4/01/91	24
PAG02	Manutenção	Sistema de RH	6126	José	B1	9	1/11/92	12

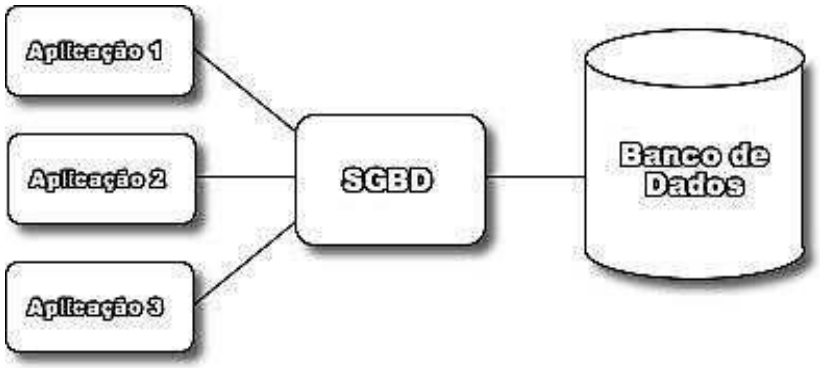
b)

Cód. Curso: INF001		**Nome Curso**: Projeto BD	**Cód. Área**: INF		**Descrição Área**: Informática	
Matrícula Funcionário	**Data de Admissão**	**Nome do Funcionário**	**Ano Concl.**	**Cód. Cargo**	**Nome do Cargo**	**Avaliação**
00129	01/03/1999	Alberto dos Santos	2000	001	Analista Junior	Regular
93821	05/03/1976	José da Silva	2002	002	Analista Sênior	Muito Bom
29841	09/09/2000	Maria José da Silva	2001	001	Analista Junior	Excelente
93820	08/07/1998	Rosa Maria	2000	003	Analista Pleno	Bom
00129	01/03/1999	Alberto dos Santos	2002	002	Analista Sênior	Muito Bom

FASE III : ARQUITETURA DE BANCO DE DADOS E LINGUAGEM SQL

9. CONCEITOS BÁSICOS

Banco de Dados pode ser entendido como um depósito de um conjunto de registros que são controlados por meio de um computador e que oferece aos usuários vários recursos como: inserção, eliminação e atualização dos dados. Um sistema gerenciador de banco de dados (SGBD) é basicamente o responsável por facilitar o processo de definição e manipulação do banco de dados.



Portanto, o SGBD não é o banco de dados, ele é o sistema que gerencia, ou seja, que controla o banco de dados (BD), ele é projetado para fazer o controle de grandes volumes de informações. Em um SGBD podemos ter vários bancos de dados. Nas nossas atividades usaremos o MySQL como

SGBD. Porém existem outros no mercado como: Oracle, PostgreSQL, DB2, Informix, SQL Server e outros.

Existem vários sites onde você pode baixar o MySQL, mas o ideal é visitar o site oficial

http://dev.mysql.com/downloads/mysql/

Após o sistema operacional preparar para a instalação, deverá abrir a tela com a apresentação do programa, clique em “NEXT” até finalizar a instalação.

Em seguida, abrirá outra tela que deverá iniciar as configurações. Clique em “NEXT” e finalize as configurações.

9.1. BANCOS DE DADOS RELACIONAIS

Os Bancos de Dados Relacionais foram desenvolvidos para facilitar o acesso aos dados. Pois enquanto em um banco de dados hierárquico os usuários precisam definir as questões de maneira mais específica, iniciando pela raiz, nos Bancos de Dados Relacionais os usuários podem fazer perguntas relacionadas através de vários pontos.

A arquitetura de um banco de dados relacional pode ser descrita usando os termos tabela, linha e coluna. Veja um exemplo;

Tabela Clientes	
Código do Cliente	Nome
00001	Antônio Silv
00002	Fábio Cost
00003	Maria Strim

Temos a “Tabela Clientes” que contém duas colunas que são o “Código do Cliente” e o “Nome”, Cada linha é formada por uma lista ordenada de colunas representando um registro.

Então um registro é uma instância de uma tabela, ou entidade. Uma entidade é

uma representação de um conjunto de informações sobre determinado conceito do sistema. Toda entidade possui atributos, que são as informações que referenciam a entidade.

Se houverem informações a serem armazenadas, você tem uma entidade. Exemplificando: Eu desejo armazenar os seguintes dados do livro: Título, Autor, Editora, Ano, Edição e Volume. Temos então a entidade Livro.

No exemplo acima “Antônio Silva” é uma instância (registro) da “Tabela Clientes”. As colunas de uma tabela são também chamadas de atributos. Então a coluna “Nome” é um atributo da “Tabela Clientes”.

As tabelas relacionam-se umas as outras através de chaves. Uma chave é um conjunto de um ou mais atributos que determinam a unicidade de cada registro. No nosso caso a chave da “Tabela Clientes” é o “Código do Cliente”, pois ele é único para cada registro.

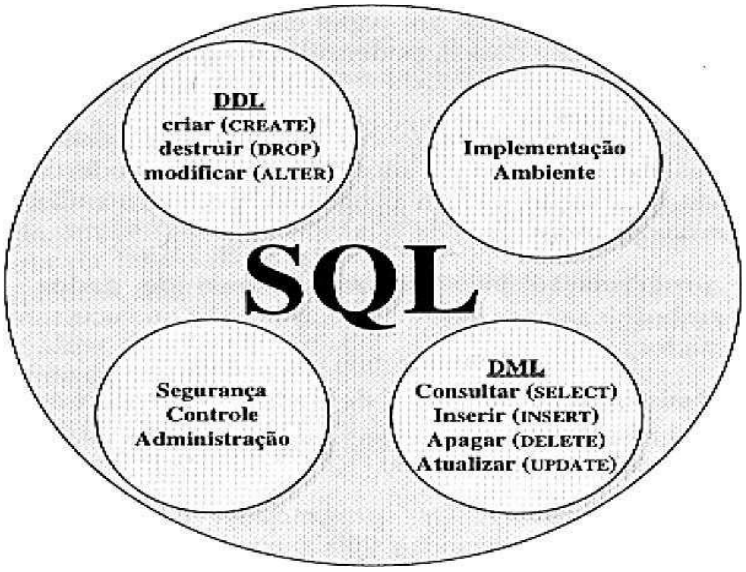
9.2. LINGUAGEM SQL

O nome “SQL” significa “Structured Query Language” que em português quer dizer “Linguagem de Consulta Estrutura”. Essa linguagem teve seus fundamentos no modelo relacional de Codd (1970). Sua primeira versão recebeu o nome de SEQUEL - “Structured English Query Language”, sendo definida principalmente por D.D.CHAMBERLIN, em 1974, nos laboratórios de pesquisa da IBM.

Em 1975, foi testado um protótipo de aplicação dessa nova linguagem. Entre 1976 e 1977, o SEQUEL foi revisado e ampliado, e teve seu nome alterado para “SQL”. Devido ao sucesso dessa nova forma de consulta a manipulação de dados, dentro de um ambiente de banco de dados, a utilização da SQL foi se tornando cada vez maior. Com isso uma grande quantidade de SGBD‟s foi tendo como linguagem básica a SQL.

A SQL se tornou um padrão de fato, no mundo dos ambientes de banco de dados relacionais. Em 1982, o American National Standard Institute (ANSI) tornou a SQL padrão oficial de linguagem em ambiente relacional.

Por ser uma linguagem de numerosas aplicações, a SQL pode manipular dados de diferentes bancos entre as funções de um SGBD.



9.3. COMPOSIÇÃO DOS BANCOS DE DADOS – DDL e DML

A linguagem SQL é dividida em subconjuntos de acordo com as operações que se deseja efetuar sobre um banco de dados. Os principais subconjuntos são:

DDL - Data Definition Language (Linguagem de Definição de Dados)

O conjunto de comandos da linguagem DDL é usado para a definição das estruturas de dados, fornecendo as instruções que permitem a criação, alteração e remoção de banco de dados, tabelas e etc.

- Principais comandos: CREATE, ALTER e DROP

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

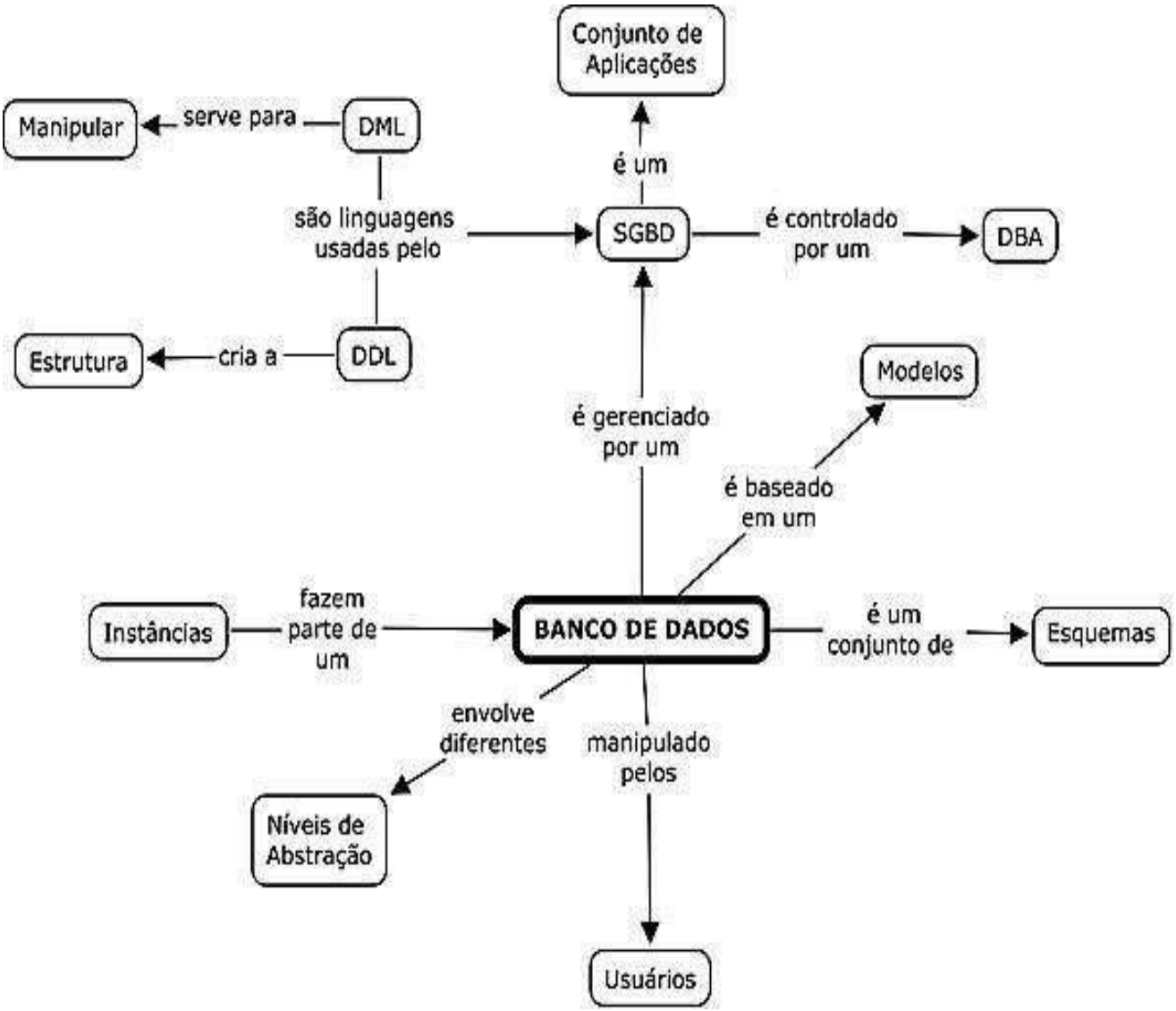
É o grupo de comandos dentro da linguagem SQL utilizado para a recuperação, inclusão, remoção e modificação de informações em bancos de dados.

- Principais comandos: SELECT, INSERT, UPDATE, DELETE, e outros.

COMANDO	DESCRIÇÃO	GRUPO
SELECT	Utilizado para extrair dados do banco	DML
INSERT	Introduzir novas linhas	DML
UPDATE	Alterar linhas já existentes	DML
DELETE	Apagar linhas já existentes	DML
CREATE	Criar estruturas no banco de dados	DDL
ALTER	Alterar estruturas do banco de dados	DDL
DROP	Apagar estruturas do banco de dados	DDL

EXERCICIO

1 – Com base no que você já aprendeu sobre banco de dados faça uma produção textual de no mínimo 5 linhas descrevendo o diagrama abaixo:


Conjunto de Aplicações
Manipular
serve para
DML
é um
são linguagens usadas pelo
SGBD
é controlado por um
DBA
Estrutura
cria a
DDL
Modelos
é gerenciado por um
é baseado em um
Instâncias
fazem parte de um
BANCO DE DADOS
é um conjunto de
Esquemas
envolve diferentes
manipulado pelos
Níveis de Abstração
Usuários

2- Marque V para verdadeiro e F para falso:

( ) CREATE, DROP, ALTER são comandos da Linguagem de Definição de Dados (DDL);

( ) É possível usarmos o comando CREATE para alterar estruturas da base de dados.

( ) O comando UPDATE realiza a exclusão de registros em uma tabela;

( ) Para apagar linhas da tabela tanto faz usar o comando DROP como o DELETE.

( ) O comando INSERT realiza a inclusão de um ou mais registros em uma tabela;

A sequência correta seria:

a) V - V - F – V - F;

b) V - F - F – V - V;

c) F - V - V – F - F;

d) V - F - F – F - V;

3 - Relacione a primeira coluna com a segunda e no final, indique a resposta correta:

1 - CREATE ( ) Comando responsável por inserir valores em uma tabela;

2 – INSERT ( ) - Responsável por retornar registros de uma tabela;

3 – DROP ( ) - Remove registros de uma tabela;

4 – SELECT ( ) - Tem o poder de remover estruturas do banco de dados;

5 – DELETE ( ) - Responsável por criar estruturas no banco de dados;

A sequência correta seria:

a) 4 - 2 - 1 - 3 - 5;

b) 2 - 4 - 5 - 3 - 1;

c) 3 - 5 - 4 - 2 - 1;

d) 2 - 1 - 3 - 4 - 5;

10. INTRODUÇÃO AO MySQL

1º Passo:

Após a Instalação do MySQL no seu computador, abra o MySQL Command Line Client. No meu caso eu baixei a versão 5.5 e sempre deixo fixado no Menu Iniciar, mas qualquer coisa é só você seguir o caminho (Iniciar-> Todos os Programas-> MySQL-> MySQL Server).

2º Passo:

Depois de Aberto o MySQL, ele pede para que você entre com o password, ou seja, com a senha que você definiu na hora da instalação.

```
Enter password: ***
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 5.5.30 MySQL Community Server (GPL)

Copyright (c) 2000, 2013, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

3º Passo:

Para conhecer quais os bancos de dados existentes dentro do MySQL basta usar o comando SHOW DATABASES. SHOW significa mostrar, apresentar, exibir e DATABASES significa bases de dados ou bancos de dados, ou seja, manda mostrar os bancos de dados existentes.

No meu caso eu já tenho sete bancos de dados criados, como você está acessando pela primeira vez deve ter pelo menos três que já vem junto da instalação que é o 'information_schema', o 'mysql' e o 'test' que é geralmente fornecido como um espaço de trabalho para usuários fazerem testes.

10.1. TIPOS DE DADOS

Strings	
Char	Este tipo de dados armazena uma string de tamanho fixo, com espaços à direita. O tamanho varia de 1 a 255 caracteres. Na consulta, o MySQL ignora os espaços que não foram ocupados.
Varchar	Armazena uma string de tamanho variável, com tamanho mínimo de 1 caracter e máximo de 255.
Tinyblob, Blob, Mediumblob, Longblob	Estes tipos de campo armazenam dados no formato binário.

Muito cuidado, pois o mysql tem como delimitador o ponto e vírgula (;), ou seja, se um ponto e vírgula (;) aparecer em meio a um código, o mysql entende que ali, o comando está encerrado e não há necessidade de continuidade. Muitos erros acontecem por falta do (ponto e vírgula) no final do comando.

Vai ai outra dica. No linux, o mysql é case-sensitive por padrão. No windows não, tanto faz letras maiúsculas como minúsculas. Para desabilitar a sensibilidade do Linux, habilite o parâmetro "lower_case_table_names" no MySQL e pronto.

Data/Hora	
Date	Este tipo de dados pode armazenar uma data no formato AAAA-MM-DD
Datetime	Pode armazenar uma data no formato AAAA-MM-DD e uma hora no formato HH:MM:SS
Timestamp	Temos a possibilidade de inserir automaticamente a data/hora atual. Para que isso acontece, basta que ao campo não seja atribuído nenhum valor no momento de inserção do registo.
Time	Armazena um valor horário, no formato HH:MM:SS.
Year	Armazena um numero anual de dois ou quatro dígitos

Numéricos	
Int	Tipo de dados que pode variar de -2147483648 a 2147483647 e de 0 a 4294967295, caso o parâmetro UNSIGNED seja utilizado.
Float	Armazena um número do tipo ponto flutuante de precisão simples. Varia de -3.402823466E+38 a -1.175494351E-38, 0, e 1.175494351E-38 a 3.402823466E+38
Double	Armazena um número do tipo ponto flutuante de precisão dupla. Varia de -1.7976931348623157E+308 a -2.2250738585072014E-308, 0, e 2.2250738585072014E-308 a 1.7976931348623157E+308.
Decimal	Tipo de dados que se comporta como o do tipo CHAR, ou seja, cada dígito ocupa 1 byte.

10.2. INSTRUÇÕES DDL – CREATE

Instrução que permite a criação de um banco de dados e também as demais estruturas de um banco. As duas situações onde o CREATE é utilizado com maior frequência são:

1. Criação de um novo banco de dados
2. Criação de uma nova tabela

1° Passo:

Abra o MySQL e digite o comando CREATE DATABASE <nome do banco> ;

```
mysql> CREATE DATABASE escola;
Query OK, 1 row affected (0.00 sec)
```

2º Passo:

Para que você possa ter acesso a qualquer informação do banco de dados, ou mesmo criar tabelas, inserir registro ou fazer uma consulta você precisa entrar dentro dele. O comando para isso é USE <nome do banco>;

```
mysql> USE escola;
Database changed
```

3º Passo:

Agora como você já tem o banco de dados e entrou nele, você já pode começar a criar as tabelas do seu banco. Para criar tabelas é preciso seguir um padrão básico:

CREATE TABLE <nome da tabela>(

<nome da coluna> <tipo de dados> (<quantidade de caracteres>),

<nome da coluna> <tipo de dados> (<quantidade de caracteres>));

```
mysql> CREATE TABLE aluno(
    -> nome VARCHAR(50),
    -> idade INT(2));
Query OK, 0 rows affected (0.04 sec)
```

Esse é um exemplo simples da criação de uma tabela do BD, você pode de acordo com a situação acrescentar mais campos e defini-los com mais características. Por exemplo, a coluna nome pode ser definida como NOT NULL, ou seja, não pode ser nulo(vazio).

Quando for inserido um novo aluno o nome dele vai ser obrigatório. Pense comigo, na matricula de um novo aluno o seu nome não pode faltar e o numero da sua matrícula é gerada automaticamente pelo sistema.

Para a criação do numero da matrícula automático é usado o código AUTO_INCREMENT, esse código permite que o número da matrícula seja acrescentado a cada novo registro inserido na tabela, ou seja, a cada novo aluno matriculado.

Como a matrícula é um número que não se repete e que é único para cada aluno ele pode ser declarado como chave primária. Em banco de dados relacionais, a chave primária é muito importante para quando for fazer o relacionamento entre as tabelas. Mais na frente iremos falar mais detalhado sobre chaves.

Ao ser definida uma coluna como chave primária, estamos dizendo para o banco de dados que não podem existir duas linhas nessa coluna com um mesmo valor. Geralmente quando falamos de chave primária usamos a sigla PK referente à palavra PRIMARY KEY.

```
mysql> CREATE TABLE novo_aluno(
    -> matricula INT(10) AUTO_INCREMENT,
    -> aluno VARCHAR(50) NOT NULL,
    -> idade INT(2),
    -> PRIMARY KEY (matricula));
Query OK, 0 rows affected (0.06 sec)
```

4º Passo:

Para ver como ficou a estrutura das tabelas criadas usamos o codigo DESCRIBE TABLE <nome da tabela>; ou somente DESC TABLE <nome da tabela>; Mais se você quer ver como foi criada a tabela pode usar o comando SHOW CREATE TABLE <nome da tabela>

Exemplo **DESCRIBE**:

```
mysql> DESCRIBE novo_aluno;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| matricula | int(10)     | NO   | PRI | NULL    | auto_increment |
| aluno     | varchar(50) | NO   |     | NULL    |                |
| idade     | int(2)      | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
3 rows in set (0.01 sec)
```

Exemplo **DESC**:

```
mysql> DESC novo_aluno;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| matricula | int(10)     | NO   | PRI | NULL    | auto_increment |
| aluno     | varchar(50) | NO   |     | NULL    |                |
| idade     | int(2)      | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
3 rows in set (0.01 sec)
```

Exemplo **SHOW CREATE**:

```
mysql> SHOW CREATE TABLE novo_aluno;
+------------+-----------------------------------------------------
-----------------------------------------------------------------+
| Table      | Create Table

                                                                 |
+------------+-----------------------------------------------------
-----------------------------------------------------------------+
| novo_aluno | CREATE TABLE `novo_aluno` (
  `matricula` int(10) NOT NULL AUTO_INCREMENT,
  `aluno` varchar(50) NOT NULL,
  `idade` int(2) DEFAULT NULL,
  PRIMARY KEY (`matricula`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+------------+-----------------------------------------------------
-----------------------------------------------------------------+
1 row in set (0.00 sec)
```

Veja agora como ficou o código criado e compare com o que você fez.

```
mysql> CREATE DATABASE escola;
Query OK, 1 row affected (0.00 sec)

mysql> USE escola;
Database changed
mysql> CREATE TABLE aluno(
    -> nome VARCHAR(50),
    -> idade INT(2));
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TABLE novo_aluno(
    -> matricula INT(10) AUTO_INCREMENT,
    -> aluno VARCHAR(50) NOT NULL,
    -> idade INT(2),
    -> PRIMARY KEY(matricula));
Query OK, 0 rows affected (0.01 sec)

mysql> DESCRIBE novo_aluno;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| matricula | int(10)     | NO   | PRI | NULL    | auto_increment |
| aluno     | varchar(50) | NO   |     | NULL    |                |
| idade     | int(2)      | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
3 rows in set (0.01 sec)

mysql> DESCRIBE aluno;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| nome  | varchar(50) | YES  |     | NULL    |       |
| idade | int(2)      | YES  |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
2 rows in set (0.01 sec)

mysql>
```

EXERCICIO

1. Com base no exemplo anterior crie um novo banco de dados chamado biblioteca.

- Criar a tabela livros contendo os seguintes campos:

✓ Código: inteiro com no máximo 8 dígitos
✓ Título: máximo de 40 caracteres
✓ Autor: máximo de 50 caracteres
✓ Editora: máximo de 20 caracteres
✓ Edição: inteiro com no máximo 2 dígitos

➢ O código é a chave primária e deve ser gerado automaticamente
➢ O Título e o Autor não podem ser vazios.

- Escreva o código da criação no quadro abaixo:

10.3. INSTRUÇÕES DDL – ALTER

Já aprendemos a criar tabelas no BD. Mas, como proceder para inserir, por exemplo, uma coluna em uma tabela já existente. Será que é necessário para isso excluir toda a tabela? A resposta para essa pergunta é não. Utilizando o comando ALTER, é possível que o usuário altere a estrutura de uma tabela, por exemplo, inserindo novas colunas.

Usando o comando ALTER, é possível realizar as seguintes alterações na estrutura de uma tabela:

1. Adicionar colunas;
2. Excluir colunas;
3. Alterar o tipo e o nome de uma coluna já existente.

Utilizando o banco de dados escola e a tabela novo_aluno vamos adicionar uma coluna chamada de endereco. Para adicionar colunas na tabela seguimos um padrão básico:

➔ **ALTER TABLE <nome da tabela> ADD <nome da coluna> <tipo de dado>;**

OBS: No comando ALTER aparece à cláusula ADD indicando que estamos adicionando uma coluna à tabela e definindo o seu tipo.

Exemplo **ADD**:

```
mysql> use escola;
Database changed
mysql> ALTER TABLE novo_aluno ADD endereco VARCHAR(150);
Query OK, 0 rows affected (0.63 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE novo_aluno;
+-----------+--------------+------+-----+---------+----------------+
| Field     | Type         | Null | Key | Default | Extra          |
+-----------+--------------+------+-----+---------+----------------+
| matricula | int(10)      | NO   | PRI | NULL    | auto_increment |
| aluno     | varchar(50)  | NO   |     | NULL    |                |
| idade     | int(2)       | YES  |     | NULL    |                |
| endereco  | varchar(150) | YES  |     | NULL    |                |
+-----------+--------------+------+-----+---------+----------------+
4 rows in set (0.05 sec)
```

Perceba que a coluna endereço foi adicionada na ultima posição, porém nem sempre queremos que a coluna que adicionarmos fique por ultimo, por isso Você pode utilizar palavras chaves como FIRST (primeiro) ou AFTER (após) para posicionar a nova coluna na posição que desejar na tabela.

Exemplo **FIRST**:

Usando o mesmo banco de dados, porém na tabela aluno vamos adicionar a coluna código como sendo a primeira coluna e chave primaria da tabela.

```
mysql> ALTER TABLE aluno ADD codigo INT(8) AUTO_INCREMENT PRIMARY KEY FIRST;
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> DESCRIBE aluno;
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| codigo | int(8)      | NO   | PRI | NULL    | auto_increment |
| nome   | varchar(50) | YES  |     | NULL    |                |
| idade  | int(2)      | YES  |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+
3 rows in set (0.01 sec)
```

Exemplo **AFTER**:

O FIRT insere na primeira coluna, já com o AFTER você escolhe a posição que deseja que a coluna fique, ou melhor, depois de quem a coluna vai ficar. Usando a mesma tabela aluno vamos inserir a coluna nascimento depois da coluna nome.

```
mysql> ALTER TABLE aluno ADD nascimento DATE AFTER nome;
Query OK, 1 row affected (0.07 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> DESCRIBE aluno;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| codigo     | int(8)      | NO   | PRI | NULL    | auto_increment |
| nome       | varchar(50) | YES  |     | NULL    |                |
| nascimento | date        | YES  |     | NULL    |                |
| idade      | int(2)      | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

Agora suponhamos que a coluna idade da tabela aluno seja desnecessária, então, neste caso, será possível também apagar a coluna da tabela. Para excluir uma coluna usamos o comando ALTER TABLE <nome da tabela> DROP <nome da coluna>; Perceba que agora usamos o DROP, em vez de ADD.

Exemplo **DROP**:

```
mysql> ALTER TABLE aluno DROP idade;
Query OK, 1 row affected (0.02 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SHOW FIELDS FROM aluno;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| codigo    | int(8)      | NO   | PRI | NULL    | auto_increment |
| nome      | varchar(50) | YES  |     | NULL    |                |
| nascimento| date        | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
3 rows in set (0.01 sec)
```

OBS: o comando SHOW FIELDS tem a mesma função do DESCRIBE.

Também podemos alterar o tipo e o nome de uma coluna. Usamos o comando:

➔ **ALTER TABLE <nome da tabela> CHANGE <nome da coluna> <nome da nova coluna> <tipo de dado da nova coluna>;**

Como exemplo suponha que a coluna código da tabela aluno não está no padrão das outras tabelas do banco, porque nas outras tabelas a PK é definida com o nome “id” e no máximo de 10 caracteres. Então em vez de excluir toda a tabela ou excluir toda a coluna podemos usar a cláusula CHANGE e renomear o campo.

Exemplo **CHANGE**:

```
mysql> ALTER TABLE aluno CHANGE codigo id INT(10) auto_increment;
Query OK, 1 row affected (0.02 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> DESCRIBE aluno;
+-----------+-------------+------+-----+---------+----------------+
| Field     | Type        | Null | Key | Default | Extra          |
+-----------+-------------+------+-----+---------+----------------+
| id        | int(10)     | NO   | PRI | NULL    | auto_increment |
| nome      | varchar(50) | YES  |     | NULL    |                |
| nascimento| date        | YES  |     | NULL    |                |
+-----------+-------------+------+-----+---------+----------------+
3 rows in set (0.01 sec)
```

10.4. INSTRUÇÕES DDL – DROP

O comando DROP é usado para excluir tabelas ou banco de dados, ele é bastante simples de se escrever, porém deve ser usado com prudência, pois uma vez excluído uma tabela ou um banco de dados a ação não pode ser desfeito. Por isso aconselho somente ser usado por quem criou o banco de dados. Com o comando DROP é possível, por exemplo:

1. **Excluir Tabelas**

Para que você possa excluir uma tabela é preciso que ela exista dentro do banco de dados. Normalmente primeiro usamos o comando SHOW TABLES, para saber as tabelas que existem dentro do BD. Após usamos o comando

➔ **DROP TABLE <nome da tabela>;**

Muitos se confundem entre o DELETE e o DROP. O comando DELETE é usado para manipulação e não para definição dos dados, ou seja, o DELETE remove os registros enquanto que o DROP remove a estrutura.

EXEMPLO

Vamos usar como exemplo o BD escola. Perceba que criamos duas tabelas uma se chama aluno e a outra novo_aluno, não é preciso que tenhamos essas duas tabelas em nosso banco então vamos excluir a tabela novo_aluno.

```
mysql> USE escola;
Database changed
mysql> SHOW TABLES;
+-----------------+
| Tables_in_escola |
+-----------------+
| aluno           |
| novo_aluno      |
+-----------------+
2 rows in set (0.04 sec)

mysql> DROP TABLE novo_aluno;
Query OK, 0 rows affected (0.18 sec)

mysql> SHOW TABLES;
+-----------------+
| Tables_in_escola |
+-----------------+
| aluno           |
+-----------------+
1 row in set (0.00 sec)
```

2. **Excluir Banco de Dados**

Assim como para excluir tabelas é preciso ver quais existem no BD, para excluir banco de dados é bom ver quais os bancos de dados existentes dentro do SGBD, o comando usado é SHOW DATABASES. Mas se você não quer perder tempo olhando os bancos de dados existentes é só acrescentar IF EXISTS após o comando DROP DATABASE. Assim:

➔ **DROP DATABASE IF EXISTS <nome do banco de dados>;**

OBS: IF EXISTS é utilizado para impedir a ocorrência de erros, se o banco de dados ou a tabela não existirem. Porém não é obrigatório o seu uso.

EXEMPLO

Vamos usar como exemplo o banco de dados test que veio junto da instalação. Esse BD é geralmente fornecido como um espaço de trabalho para usuários fazerem testes como não precisamos dele posso exclui-lo.

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| blog               |
| cad                |
| escola             |
| ext                |
| mysql              |
| performance_schema |
| sys_exemplo        |
| test               |
+--------------------+
9 rows in set (0.00 sec)

mysql> DROP DATABASE IF EXISTS test;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| blog               |
| cad                |
| escola             |
| ext                |
| mysql              |
| performance_schema |
| sys_exemplo        |
+--------------------+
8 rows in set (0.00 sec)
```

10.5. CHAVE PRIMÁRIA E CHAVE ESTRANGEIRA

Os Bancos de Dados Relacionais são formados de várias tabelas e utilizam chaves como forma de referenciar outras tabelas. Esse relacionamento entre as tabelas é feito através das chaves estrangeiras. Quando falamos de chave estrangeira usamos a sigla FK referente à palavra FOREIGN KEY.

Por exemplo, poderíamos abaixo fazer a tabela CLIENTE com os campos 'cliente_id, nome, endereço cidade' e a tabela PEDIDO com os campos 'pedido_id, cliente_id, preço, data' para armazenar os pedidos.

Os campos com sublinhado simples são as chaves primárias (PK) que dão uma identificação única a cada item, e o campo com pontilhado é uma chave estrangeira através da qual a tabela CLIENTE se relaciona com a tabela PEDIDO.

CLIENTE

Cliente_id	Nome	Endereço	Cidade
1	João Silva	Rua Uruguaiana	Porto Velho
2	Maria Francisca	Rua México	Cacoal
3	Antonio José	Rua Piau	Porto Velho

Tabela 1.a

PEDIDO

Pedido_id	Cliente_id	Preço	Data
8	3	23	01/05/05
9	1	45	06/08/05
10	3	67	04/07/05

Tabela 1.b

.

Esse tipo de representação evita que para cada pedido precisemos colocar a toda a identificação do cliente, basta armazenarmos o id do cliente na tabela de pedidos e através dele podemos identificar de quem é o pedido. Uma chave estrangeira sempre se relaciona com a chave primária da tabela do relacionamento.

Tratando de códigos, Vamos criar um banco de dados chamado VENDA e as duas tabelas CLIENTE e PEDIDO com os campos que foram definidos.

A definição da chave estrangeira segue o seguinte padrão:

➔ **FOREIGN KEY<nome da FK> REFERENCES < tabela da PK>(<nome da PK>)**

OBS: o nome e o tipo de dado da PK devem ser o mesmo da FK. A palavra REFERENCES é usada para dizer qual tabela a FK está referenciando.

```
mysql> CREATE DATABASE venda;
Query OK, 1 row affected (0.00 sec)

mysql> USE venda;
Database changed
mysql> CREATE TABLE cliente(
    -> cliente_id int(8) auto_increment,
    -> nome varchar(50) not null,
    -> endereco varchar(80) not null,
    -> cidade varchar(30) not null,
    -> primary key(cliente_id));
Query OK, 0 rows affected (0.59 sec)

mysql> CREATE TABLE pedido(
    -> pedido_id int(8) auto_increment,
    -> cliente_id int(8) not null,
    -> preco double not null,
    -> data date not null,
    -> primary key(pedido_id),
    -> foreign key(cliente_id) references cliente(cliente_id));
Query OK, 0 rows affected (0.11 sec)

mysql> DESC cliente;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| cliente_id | int(8)      | NO   | PRI | NULL    | auto_increment |
| nome       | varchar(50) | NO   |     | NULL    |                |
| endereco   | varchar(80) | NO   |     | NULL    |                |
| cidade     | varchar(30) | NO   |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
4 rows in set (0.56 sec)

mysql> DESC pedido;
+------------+--------+------+-----+---------+----------------+
| Field      | Type   | Null | Key | Default | Extra          |
+------------+--------+------+-----+---------+----------------+
| pedido_id  | int(8) | NO   | PRI | NULL    | auto_increment |
| cliente_id | int(8) | NO   | MUL | NULL    |                |
| preco      | double | NO   |     | NULL    |                |
| data       | date   | NO   |     | NULL    |                |
+------------+--------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

EXERCICIO

1) Comando SQL para criar um banco de dados chamado “locadora”
   a) create databases locadora;
   b) create locadora;
   c) create database locadora;
   d) use database locadora;

2) Comando SQL para apagar um banco de dados chamado "supermercado"
   a) delete database supermercado;
   b) drop table supermecado;
   c) drop supermercado;
   d) drop database supermercado;
   e) delete supermercado;

3) Comando SQL para entrar no banco de dados chamado "EEEP"
   a) use database EEEP;
   b) use table EEEP;
   c) create EEEP;
   d) usage EEEP;
   e) use EEEP;

4) Comando SQL para mostrar todos os bancos de dados existentes;
   a) show database;
   b) show tables;
   c) show databases;
   d) show BD;

5) Comando SQL para apagar a tabela "produtos" do BD "supermercado"
   a) drop table produtos;
   b) delete table produtos;
   c) drop supermercado;
   d) drop table supermercado;
   e) drop produtos;

6) Assinale a opção correta:

O comando DESCRIBE gera o mesmo resultado do comando _____ .

A opção que melhor completa a sentença é

a) SHOW DETAILS FROM;
b) SELECT DETAILS FROM;
c) SHOW FIELDS FROM;
d) SELECT FIELDS FROM;

7) Relacione a primeira coluna com a segunda, após, indique a sequência correta:

1 - ALTER TABLE ADD ( ) Usado para apagar uma tabela de um BD

2 - ALTER TABLE CHANGE ( ) Usado para trocar o nome e o tipo de uma coluna

3- DROP TABLE ( ) Usadas para adicionar uma coluna em uma tabela

A sequência correta seria:

a) 3 - 1 - 2;
b) 2 - 3 - 1;
c) 1 - 2 - 3;
d) 3 - 2 - 1;

8) Marque a opção em que a criação da tabela "filmes" esta feito corretamente:

a) create table filmes (
codigo int(20) note null,
titulo varchar int(30) note null);

b) create table filmes (
codigo int(20) not null,
titulo varchar(30) not null);

c) create tables filmes (
codigo inteiro(20) not nul,
titulo double not nul);

d) create table filmes (
codigo int(20) net null,
titulo String(30) net null);

9) Com base no exemplo que criamos chave primária(PK) e chave estrangeira(FK), crie um banco de dados EMPRESA e as duas tabelas DEPTO e EMP descritas abaixo com suas respectivas colunas.

DEPTO

CodigoDepto	NomeDepto
D1	Compras
D2	Engenharia
D3	Vendas

CodigoDepto em EMP é uma chave estrangeira em relação a tabela DEPTO

EMP

CodigoEmp	Nome	CodigoDepto	CategFuncional	CIC
E1	Souza	D1	-	132.121.331-20
E2	Santos	D2	C5	891.221.111-11
E3	Silva	D2	C5	341.511.775-45
E5	Soares	D1	C2	631.692.754-88

- Escreva o código da criação no quadro abaixo:

10.6. INSTRUÇÕES DML – INSERT

Já demonstramos como criar uma tabela no banco de dados, agora mostraremos as maneiras básicas de se inserir dados nessa tabela. O comando usado para inserir dados é o INSERT. A síntese básica do comando INSERT é a seguinte:

➔ **INSERT INTO <nome da tabela> (campo1, campo2) VALUES (valor1, valor2);**

Cada valor é inserido no campo que corresponde à posição do valor na lista: valor1 é inserido no campo1, valor2 no campo2 e assim por diante.

OBS: Os valores devem ser separados com uma vírgula e se o tipo do campo for texto deve está entre aspas duplas ou simples.

EXEMPLO

Vamos usar como exemplo a tabela "cliente" que criamos no banco de dados "venda", se você não criou essa tabela volte para página antes do exercício onde estão os códigos da criação. Mas antes de inserir os dados na tabela é bom ver quais os campos existentes, o comando usado pode ser o DESCRIBE.

Agora vamos inserir na coluna nome "Francisco" no endereço "Rua: 24 de Maio nº 324" e na cidade "Fortaleza". Observe que não é preciso inserir dados no campo "cliente_id", pois ele foi definido como AUTO_INCREMENT.

```
mysql> USE venda;
Database changed
mysql> DESCRIBE cliente;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| cliente_id | int(8)      | NO   | PRI | NULL    | auto_increment |
| nome       | varchar(50) | NO   |     | NULL    |                |
| endereco   | varchar(80) | NO   |     | NULL    |                |
| cidade     | varchar(30) | NO   |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

mysql> INSERT INTO cliente(nome,endereco,cidade)
    -> VALUES('Francisco','Rua: 24 de Maio nº 324','Fortaleza');
Query OK, 1 row affected (0.05 sec)
```

Temos também a opção de omitir as declarações dos campos. Essa sintaxe funciona somente se forem repassados valores para todas as colunas.

➔ **INSERT INTO <nome da tabela> VALUES (valor1, valor2, valor3,...);**

Cada valor é inserido no campo que corresponde a sequência das colunas na tabela, se a primeira coluna, por exemplo, for o nome então o valo1 deve ser o nome, se a segunda coluna, por exemplo, for idade então o valor2 deve ser a idade e assim por diante.

OBS: Se a coluna for declarada como AUTO_INCREMENT basta você colocar o valor que corresponde a essa coluna como sendo zero(0). O valor zero(0) não influencia em nada porem se ele for esquecido vai ser gerado um erro.

EXEMPLO

Vamos usar como exemplo a mesma tabela que usamos no exemplo anterior do banco de dados "venda" que é a tabela "cliente", porém não vai ser preciso mencionar os nomes dos campos basta repassar valores para todas as colunas.

Vamos inserir o cliente "Pedro" que mora na "Av. Augusto dos Anjos nº 2674" na cidade de "Fortaleza". Observe que o primeiro valor que iremos inserir será zero(0), isso porque a primeira coluna é "cliente_id" que foi declarada como AUTO_INCREMENT.

```
mysql> USE venda;
Database changed
mysql> DESC cliente;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| cliente_id | int(8)      | NO   | PRI | NULL    | auto_increment |
| nome       | varchar(50) | NO   |     | NULL    |                |
| endereco   | varchar(80) | NO   |     | NULL    |                |
| cidade     | varchar(30) | NO   |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

mysql> INSERT INTO cliente
    -> VALUES(0,'Pedro','Av. Augusto dos Anjos nº 2674','Fortaleza');
Query OK, 1 row affected (0.04 sec)
```

CLÁUSULAS

10.7. INSTRUÇÕES DML – SELECT

A forma mais simples de se fazer um SELECT é recuperando todos os dados de uma tabela. A síntese básica é:

➔ **SELECT * FROM <nome da tabela>;**

OBS: O * (asterisco) substitui os nomes de todas as colunas, e todas serão selecionadas para o resultado da consulta. A instrução FROM indica de qual tabela estamos buscando os dados.

Caso não fosse de nosso desejo mostrar todas as colunas no resultado da consulta, bastaria nomear as colunas que deveriam aparecer no lugar do * (asterisco) e separadas por vírgula.

➔ **SELECT <coluna1>, <coluna2> FROM <nome da tabela>;**

EXEMPLO

Para que possamos ver os dados dos clientes que inserimos no exemplo passado, basta usar o comando SELECT * FROM cliente. Se você não tem certeza que o nome da tabela é essa, use o comando SHOW TABLES e veja as tabelas existentes dentro do BD.

```
mysql> USE venda;
Database changed
mysql> SHOW TABLES;
+----------------+
| Tables_in_venda |
+----------------+
| cliente        |
| pedido         |
+----------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM cliente;
+------------+----------+--------------------------------+----------+
| cliente_id | nome     | endereco                       | cidade   |
+------------+----------+--------------------------------+----------+
|          1 | Francisco | Rua: 24 de Maio nº 324        | Fortaleza |
|          2 | Pedro     | Av. Augusto dos Anjos nº 2674 | Fortaleza |
+------------+----------+--------------------------------+----------+
2 rows in set (0.00 sec)
```

FROM	Utilizada para especificar a tabela que se vai selecionar os registros.
WHERE	Utilizada para especificar as condições que devem reunir os registros que serão selecionados.
GROUP BY	Utilizada para separar os registros selecionados em grupos específicos.
HAVING	Utilizada para expressar a condição que deve satisfazer cada grupo.
ORDER BY	Utilizada para ordenar os registros selecionados com uma ordem especifica.
DISTINCT	Utilizada para selecionar dados sem repetição.

Exemplo **WHERE**

Se quisermos ver somente o nome e o endereço do cliente, como já foi dito basta colocar os nomes das colunas depois do SELECT, porém quando temos muitos registros e precisamos de uma informação especifica usamos a condição WHERE que em português significado ONDE. Sempre a cláusula WHERE vem acompanhada de alguma condição, por exemplo:

➔ **SELECT nome, endereco FROM cliente WHERE cliente_id = 2 ;**

Podemos ler o comando SQL da seguinte forma:

Selecione o nome e o endereço da tabela cliente onde cliente_id for igual a 2.

```
mysql> USE venda;
Database changed
mysql> SHOW TABLES;
+----------------+
| Tables_in_venda |
+----------------+
| cliente        |
| pedido         |
+----------------+
2 rows in set (0.03 sec)

mysql> DESCRIBE cliente;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| cliente_id | int(8)      | NO   | PRI | NULL    | auto_increment |
| nome       | varchar(50) | NO   |     | NULL    |                |
| endereco   | varchar(80) | NO   |     | NULL    |                |
| cidade     | varchar(30) | NO   |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
4 rows in set (0.08 sec)

mysql> SELECT nome,endereco FROM cliente WHERE cliente_id = 2;
+-------+-------------------------------+
| nome  | endereco                      |
+-------+-------------------------------+
| Pedro | Av. Augusto dos Anjos nº 2674 |
+-------+-------------------------------+
1 row in set (0.05 sec)
```

Exemplo **ORDER BY**

Até o momento vimos como obter dados de uma tabela utilizando os comandos SELECT e WHERE. Porém, frequentemente precisamos listar os dados por uma ordem em particular. Pode ser por ordem ascendente ou descendente. Para isso podemos utilizar a cláusula ORDER BY para ordenar os dados. A sintaxe básica da cláusula ORDER BY é a seguinte:

➔ **SELECT < coluna > FROM < tabela > ORDER BY < coluna >;**

OBS: Por padrão o ORDER BY vem como ASC significa que os resultados serão apresentados por ordem ascendente, ou seja, do menor para o maior. Mais também pode ser DESC significa que os resultados serão apresentados por ordem descendente, para isso acontecer você precisa declara-lo.

EXEMPLO

Como exemplo irei usar uma tabela que criei anteriormente chamada produto:

```
mysql> select * from produto;
+----+----------+-------+
| id | nome     | preco |
+----+----------+-------+
|  1 | Arroz    |  6.50 |
|  2 | Farinha  |  1.80 |
|  3 | Sal      |  0.70 |
|  4 | Acucar   |  5.40 |
|  5 | Biscoito |  1.20 |
|  6 | Leite    |  2.35 |
|  7 | Vinagre  |  1.95 |
|  8 | Cafe     |  4.10 |
+----+----------+-------+
8 rows in set (0.00 sec)
```

Perceba que a tabela possui 8 produtos cadastrados, suponhamos que eu preciso ver os preços de forma ascendente, então iremos selecionar o nome e o preço ordenado pelo próprio preço, veja como ficou:

```
mysql> SELECT nome,preco FROM produto ORDER BY preco ASC;
+----------+-------+
| nome     | preco |
+----------+-------+
| Sal      |  0.70 |
| Biscoito |  1.20 |
| Farinha  |  1.80 |
| Vinagre  |  1.95 |
| Leite    |  2.35 |
| Cafe     |  4.10 |
| Acucar   |  5.40 |
| Arroz    |  6.50 |
+----------+-------+
8 rows in set (0.00 sec)
```

Podemos também ordenar os dados de outras maneiras, se eu quero ver os nomes e o preço dos produtos em ordem alfabética basta ordenar pelo nome de forma ascendente.

```
mysql> SELECT nome,preco FROM produto ORDER BY nome ASC;
+----------+-------+
| nome     | preco |
+----------+-------+
| Acucar   |  5.40 |
| Arroz    |  6.50 |
| Biscoito |  1.20 |
| Cafe     |  4.10 |
| Farinha  |  1.80 |
| Leite    |  2.35 |
| Sal      |  0.70 |
| Vinagre  |  1.95 |
+----------+-------+
8 rows in set (0.00 sec)
```

Também posso em vez de colocar o nome da coluna que eu quero ordenar basta colocar o numero dela na sequencia selecionada, por exemplo, se eu selecionei nome e preço então nome é 1 e preço é 2 e assim por diante:

```
mysql> SELECT nome,preco FROM produto ORDER BY 2 DESC;
+----------+-------+
| nome     | preco |
+----------+-------+
| Arroz    |  6.50 |
| Acucar   |  5.40 |
| Cafe     |  4.10 |
| Leite    |  2.35 |
| Vinagre  |  1.95 |
| Farinha  |  1.80 |
| Biscoito |  1.20 |
| Sal      |  0.70 |
+----------+-------+
8 rows in set (0.00 sec)
```

Perceba que agora usei o DESC na coluna 2 que é preço, então ficou de forma ascendente, ou seja, do maior para o menor, podemos usar a cláusula ORDER BY de várias maneiras em situações diferentes, basta saber como usa-lo.

10.8. INSTRUÇÕES DML – UPDATE

O comando para atualizar os dados é UPDATE, ele possui a seguinte sintaxe:

➔ **UPDATE < tabela > SET < campo > = "novo valor" WHERE < condição > ;**

- ✓ **tabela:** nome da tabela que será modificada
- ✓ **set:** define qual campo será alterado
- ✓ **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 inteira será atualizada
- ✓ **condição:** regra que impõe condição para execução do comando

EXEMPLO

Como exemplo usaremos a tabela cliente que foi usada nos exemplos anteriores. Suponhamos que o cliente de nome Francisco se mudou, então precisamos atualizar o seu registro colocando o seu novo endereço.

```
mysql> SELECT * FROM cliente;
+------------+-----------+-------------------------------+-----------+
| cliente_id | nome      | endereco                      | cidade    |
+------------+-----------+-------------------------------+-----------+
|          1 | Francisco | Rua: 24 de Maio nº 324        | Fortaleza |
|          2 | Pedro     | Av. Augusto dos Anjos nº 2674 | Fortaleza |
+------------+-----------+-------------------------------+-----------+
2 rows in set (0.42 sec)

mysql> UPDATE cliente SET endereco = "Av. C nº 687 4º Etapa Conjunto Ceará"
    -> WHERE nome = "Francisco";
Query OK, 1 row affected (0.07 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM cliente;
+------------+-----------+-----------------------------------------+-----------+
| cliente_id | nome      | endereco                                | cidade    |
+------------+-----------+-----------------------------------------+-----------+
|          1 | Francisco | Av. C nº 687 4º Etapa Conjunto Ceará    | Fortaleza |
|          2 | Pedro     | Av. Augusto dos Anjos nº 2674           | Fortaleza |
+------------+-----------+-----------------------------------------+-----------+
2 rows in set (0.00 sec)
```

Também podemos alterar mais de um campo de uma vez. Suponhamos que o cliente Pedro se mudou para outra cidade, precisamos alterar o endereço e a cidade atual, não precisamos criar dois UPDATES basta separa-los por vírgula.

➔ **UPDATE < tabela > SET < campo1 > = “valor1”, < campo2 > = “valor2” WHERE < condição > ;**

```
mysql> SELECT * FROM cliente;
+------------+----------+---------------------------------------+----------+
| cliente_id | nome     | endereco                              | cidade   |
+------------+----------+---------------------------------------+----------+
|          1 | Francisco | Av. C nº 687 4ª Etapa Conjunto Ceará | Fortaleza |
|          2 | Pedro    | Av. Augusto dos Anjos nº 2674         | Fortaleza |
+------------+----------+---------------------------------------+----------+
2 rows in set (0.00 sec)

mysql> UPDATE cliente SET endereco = "Rua Joaquim Mota nº 260" ,
    -> cidade = "Caucaia" WHERE nome = "Pedro";
Query OK, 1 row affected (0.08 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM cliente;
+------------+----------+---------------------------------------+----------+
| cliente_id | nome     | endereco                              | cidade   |
+------------+----------+---------------------------------------+----------+
|          1 | Francisco | Av. C nº 687 4ª Etapa Conjunto Ceará | Fortaleza |
|          2 | Pedro    | Rua Joaquim Mota nº 260               | Caucaia  |
+------------+----------+---------------------------------------+----------+
2 rows in set (0.00 sec)
```

OBS: Devemos passar sempre o WHERE, que é uma espécie de filtro em nossa tabela, porque senão o passarmos atualizaremos TODOS os dados da tabela e isso pode acarretar diversos problemas, dependendo do tamanho e da complexidade da sua tabela. Imagina se esquecermos de colocar uma condição em uma tabela de 1.000 registros e alterarmos todos os endereços dos clientes para um só.

10.9. INSTRUÇÕES DML – DELETE

A forma mais simples de se fazer um DELETE é excluindo todos os dados de uma tabela. A síntese básica é:

➔ **DELETE FROM < nome da tabela >;**

Se não for especificada nenhuma condição então serão excluídos todos os dados da tabela, porém se você quer excluir somente um registro é preciso usar a cláusula WHERE informando qual será a condição para deletar.

➔ **DELETE FROM < nome da tabela > WHERE < condição >;**

OBS: Lembre-se que este comando, assim como o UPDATE, pode ser perigoso em algumas situações, já que, uma vez executado esses comandos, não será possível desfazer a ação realizada. Portanto, devemos ficar atentos ao usar esses comandos em tabelas complexas.

EXEMPLO

Como exemplo vamos usar a mesma tabela que usamos nos comandos INSERT, SELECT e UPDATE, que é a tabela cliente do banco de dados venda. Suponhamos que o Francisco não é mais o nosso cliente, então devemos exclui-lo da nossa tabela, para isso usamos o comando DELETE.

```
mysql> DELETE FROM cliente WHERE cliente_id = 1;
Query OK, 1 row affected (0.12 sec)

mysql> SELECT * FROM cliente;
+------------+-------+--------------------------+---------+
| cliente_id | nome  | endereco                 | cidade  |
+------------+-------+--------------------------+---------+
|          2 | Pedro | Rua Joaquim Mota nº 260 | Caucaia |
+------------+-------+--------------------------+---------+
1 row in set (0.00 sec)
```

Perceba que usei a condição referenciando o cliente-id. Em banco de dados todo registro deve possuir o seu código, quando vamos alterar ou excluir um registro é bom que coloquemos como condição o código do registro, porque o nome pode ser que apareça outro igual, mais o código não.

FUNÇÕES DE AGREGAÇÃO	
COUNT	Utilizada para devolver o número de registros da seleção.
SUM	Utilizada para devolver a soma de todos os valores de um campo determinado.
MAX	Utilizada para devolver o valor mais alto de um campo especificado.
MIN	Utilizada para devolver o valor mais baixo de um campo especificado.
AVG	Utilizada para calcular a media dos valores de um campo determinado.

Como exemplo usaremos a tabela produto do banco de dados venda, que criamos em exemplos anteriores.

```
mysql> SELECT * FROM produto;
+----+----------+-------+
| id | nome     | preco |
+----+----------+-------+
|  1 | Arroz    |  6.50 |
|  2 | Farinha  |  1.80 |
|  3 | Sal      |  0.70 |
|  4 | Acucar   |  5.40 |
|  5 | Biscoito |  1.20 |
|  6 | Leite    |  2.35 |
|  7 | Vinagre  |  1.95 |
|  8 | Cafe     |  4.10 |
+----+----------+-------+
8 rows in set (0.00 sec)
```

Exemplo Contagem **(COUNT)**

➔ **SELECT COUNT(campo) FROM < nome da tabela > ;**

```
mysql> SELECT COUNT(preco) FROM produto;
+--------------+
| COUNT(preco) |
+--------------+
|            8 |
+--------------+
1 row in set (0.00 sec)
```

Exemplo SOMA **(SUM)**

➔ **SELECT SUM(campo) FROM < nome da tabela > ;**

```
mysql> SELECT SUM(preco) FROM produto;
+------------+
| SUM(preco) |
+------------+
|      24.00 |
+------------+
1 row in set (0.00 sec)
```

Exemplo Máximo **(MAX)**

➔ **SELECT MAX(campo) FROM < nome da tabela > ;**

```
mysql> SELECT MAX(preco) FROM produto;
+------------+
| MAX(preco) |
+------------+
|       6.50 |
+------------+
1 row in set (0.00 sec)
```

Exemplo Mínimo (**MIN)**

➔ **SELECT MIN(campo) FROM < nome da tabela > ;**

```
mysql> SELECT MIN(preco) FROM produto;
+------------+
| MIN(preco) |
+------------+
|       0.70 |
+------------+
1 row in set (0.00 sec)
```

Exemplo Média **(AVG)**

➔ **SELECT AVG(campo) FROM < nome da tabela > ;**

```
mysql> SELECT AVG(preco) FROM produto;
+-------------+
| AVG(preco) |
+-------------+
|   3.000000 |
+-------------+
1 row in set (0.00 sec)
```

Utilizando GROUP BY e HAVING

É possível dividir o conjunto em grupos e aplicar a função de agregação a cada grupo. Para executar essa ação, utilize uma cláusula GROUP BY na consulta. Aprendemos que a cláusula WHERE define uma condição de retorno de um comando SELECT. Os grupos também podem ser filtrados utilizando uma cláusula HAVING, que testa as propriedades de grupo envolvendo funções agregadas.

OBS: O HAVING é diferente do WHERE. O WHERE restringe os resultados obtidos sempre após o uso da cláusula FROM, ao passo que a cláusula HAVING filtra o retorno do agrupamento.

EXEMPLO

Como exemplo vamos usar a tabela a seguir:

```
mysql> select * from vendas;
+--------+-----------+-----------+------------+--------------+--------+
| Codigo | ClienteID | ProdutoID | ValorTotal | ValorParcial | Status |
+--------+-----------+-----------+------------+--------------+--------+
|      1 |        11 |         3 | 50,00      | 20           | F      |
|      2 |        11 |         3 | 50,00      | 30           | F      |
|      3 |        12 |         4 | 325,00     | 75           | F      |
|      4 |        12 |         4 | 325,00     | 75           | F      |
|      5 |        12 |         4 | 325,00     | 175          | F      |
|      6 |        12 |         5 | 110,00     | 60           | F      |
|      7 |        12 |         4 | 110,00     | 50           | F      |
|      8 |        13 |         5 | 70.00      | 50           | A      |
|      9 |        13 |         7 | 120.00     | 80           | A      |
|     10 |        13 |         6 | 30.00      | 20           | F      |
+--------+-----------+-----------+------------+--------------+--------+
10 rows in set (0.00 sec)
```

Suponhamos que precisamos saber o total de quanto cada cliente precisa pagar, para isso utilizamos o GROUP BY, veja como fica o código:

➔ **SELECT ClienteID, SUM(ValorTotal) FROM vendas GROUP BY ClienteID ;**

```
mysql> SELECT ClienteID, SUM(ValorTotal) FROM vendas GROUP BY ClienteID;
+-----------+-----------------+
| ClienteID | SUM(ValorTotal) |
+-----------+-----------------+
|        11 |             100 |
|        12 |            1195 |
|        13 |             220 |
+-----------+-----------------+
3 rows in set, 7 warnings (0.00 sec)
```

Agora se queremos saber somente os totais dos clientes que vão pagar um valor que seja maior que 200,00 R$ precisamos passar uma condição para o select utilizando a cláusula HAVING.

➔ **SELECT ClienteID, SUM(ValorTotal) FROM vendas GROUP BY ClienteID HAVING SUM(ValorTotal) > 200;**

```
mysql> SELECT ClienteID, SUM(ValorTotal) FROM vendas GROUP BY ClienteID HAVING
SUM(ValorTotal) > 200;
+-----------+-----------------+
| ClienteID | SUM(ValorTotal) |
+-----------+-----------------+
|        12 |            1195 |
|        13 |             220 |
+-----------+-----------------+
2 rows in set, 14 warnings (0.04 sec)
```

OBS: Aqui eu coloquei somente alguns exemplos, se eu fosse colocar todas as ações que podem ser feitas usando os comandos SQL. Acredito que não acabaríamos essa apostila tão cedo, espero que você não faça somente os exemplos mostrados na apostila, procure se aprofundar e com base nos exemplos, construa seus próprios comandos, não espere pelo professor. Estude e faça exemplos antes mesmo de o professor pedir.

OPERADORES RELACIONAIS	
<	Menor que
>	Maior que
< >	Diferente
<=	Menor ou igual que
>=	Maior ou igual que
=	Igual a
BETWEEN	Utilizado para especificar um intervalo de valores.
LIKE	Utilizado na comparação de um modelo e para especificar registros de um banco de dados."Like" + extensão % vai significar buscar todos resultados com o mesmo início da extensão.

OPERADORES LÓGICOS	
AND	Avalia as condições e devolve um valor verdadeiro caso ambos sejam corretos.
OR	Avalia as condições e devolve um valor verdadeiro se algum for correto.
NOT	Devolve o valor contrário da expressão.

Exemplo **LIKE**

Com este operador, podemos comparar Strings. O percentual (%) substitui nenhum, um ou mais caracteres, já a sublinha (_) substitui somente um caractere. Utilizando a combinação desses caracteres especiais com o que se quer localizar, pode-se conseguir uma variedade muito grande de expressões. Veja na tabela a seguir algumas possíveis combinações:

COMANDO	DESCRIÇÃO
LIKE 'A%'	Todas as palavras que iniciem com a letra A;
LIKE '%A'	Todas que terminem com a letra A;
LIKE '%A%'	Todas que tenham a letra A em qualquer posição;
LIKE 'A_'	String de dois caracteres que tenham a primeira letra A e o segundo caractere seja qualquer outro;
LIKE '_A'	String de dois caracteres cujo primeiro caractere seja qualquer um e a última letra seja A;
LIKE '_A_'	String de três caracteres cuja segunda letra seja A, independentemente do primeiro ou do último caractere;
LIKE '%A_'	Todos que tenham a letra A na panúltima posição e a última seja qualquer outro caractere;
LIKE '_A%'	Todos que tenham a letra A na segunda posição e o primeiro caractere seja qualquer um;

SELECT * FROM contatos WHERE nome LIKE ' < condição > ' ;

```
mysql> SELECT * FROM contatos;
+----+-----------+---------------------------+----------------+
| id | nome      | email                     | telefone       |
+----+-----------+---------------------------+----------------+
|  1 | Alice     | alicebarros@hotmail.com   | (85) 32945832  |
|  2 | Alexandre | alexandre@hotmail.com     | (85) 87948242  |
|  3 | Bruno     | bruno123@hotmail.com      | (85) 34892323  |
|  4 | Carla     | carlinha@gmail.com        | (85) 88533294  |
|  5 | Davi      | davi@gmail.com            | (85) 34895322  |
|  6 | Erica     | ericasouza@yahoo.com      | (85) 91432504  |
|  7 | George    | george_1998@hotmail.com   | (85) 87349232  |
|  8 | Laura     | laura@hotmail.com         | (85) 32954232  |
|  9 | Marcio    | marcio_123@gmail.com      | (85) 87426584  |
| 10 | Vitoria   | vitoria@hotmail.com       | (85) 85492504  |
+----+-----------+---------------------------+----------------+
10 rows in set (0.00 sec)

mysql> SELECT * FROM contatos WHERE nome LIKE 'A%';
+----+-----------+---------------------------+----------------+
| id | nome      | email                     | telefone       |
+----+-----------+---------------------------+----------------+
|  1 | Alice     | alicebarros@hotmail.com   | (85) 32945832  |
|  2 | Alexandre | alexandre@hotmail.com     | (85) 87948242  |
+----+-----------+---------------------------+----------------+
2 rows in set (0.04 sec)

mysql> SELECT * FROM contatos WHERE nome LIKE '%A';
+----+---------+---------------------------+----------------+
| id | nome    | email                     | telefone       |
+----+---------+---------------------------+----------------+
|  4 | Carla   | carlinha@gmail.com        | (85) 88533294  |
|  6 | Erica   | ericasouza@yahoo.com      | (85) 91432504  |
|  8 | Laura   | laura@hotmail.com         | (85) 32954232  |
| 10 | Vitoria | vitoria@hotmail.com       | (85) 85492504  |
+----+---------+---------------------------+----------------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM contatos WHERE nome LIKE '%A%';
+----+-----------+---------------------------+----------------+
| id | nome      | email                     | telefone       |
+----+-----------+---------------------------+----------------+
|  1 | Alice     | alicebarros@hotmail.com   | (85) 32945832  |
|  2 | Alexandre | alexandre@hotmail.com     | (85) 87948242  |
|  4 | Carla     | carlinha@gmail.com        | (85) 88533294  |
|  5 | Davi      | davi@gmail.com            | (85) 34895322  |
|  6 | Erica     | ericasouza@yahoo.com      | (85) 91432504  |
|  8 | Laura     | laura@hotmail.com         | (85) 32954232  |
|  9 | Marcio    | marcio_123@gmail.com      | (85) 87426584  |
| 10 | Vitoria   | vitoria@hotmail.com       | (85) 85492504  |
+----+-----------+---------------------------+----------------+
8 rows in set (0.00 sec)
```

Utilizando AND / OR

O operador **AND** exibe os registros se tanto a primeira condição como a segunda condição for verdadeira. O operador **OR** exibe os registros se a primeira condição ou a segunda for verdadeira. Esses operadores são usados para filtrar registros com base em mais de uma condição.

```
mysql> SELECT * FROM convidados;
+----+---------+-----------+-------------------------+-----------+----------+
| id | nome    | sobrenome | endereco                | cidade    | telefone |
+----+---------+-----------+-------------------------+-----------+----------+
|  1 | Alberto | Silva     | Castelo Branco 123      | fortaleza | 32584930 |
|  2 | Bruno   | Sousa     | Vicente Carvalho        | Fortaleza | 87954738 |
|  3 | Carlos  | Moura     | Pedro Pereira 438       | Fortaleza | 34869353 |
|  4 | Davi    | Ferreira  | Castro Silva 754        | Fortaleza | 34869353 |
|  5 | Eliana  | Silva     | Joaquim Mota 78         | Caucaia   | 87594537 |
|  6 | Marcio  | Tavares   | Fernando Pinto 654      | Caucaia   | 32864930 |
|  7 | Bruno   | Ferreira  | Oliveira Paiva          | Fortaleza | 34895842 |
|  8 | Vitor   | Silva     | Conjunto Ceara Av.C 453 | Fortaleza | 87947435 |
|  9 | Carlos  | Andrade   | Santa Adélia 274        | Eusebio   | 91058249 |
| 10 | Maria   | Betania   | Tiburcio Targino        | Aquiraz   | 86950353 |
| 11 | Carlos  | Silva     | Imperador 453           | Fortaleza | 34895472 |
+----+---------+-----------+-------------------------+-----------+----------+
11 rows in set (0.03 sec)

mysql> SELECT * FROM convidados WHERE nome = 'Carlos' AND sobrenome = 'Silva';
+----+--------+-----------+--------------+-----------+----------+
| id | nome   | sobrenome | endereco     | cidade    | telefone |
+----+--------+-----------+--------------+-----------+----------+
| 11 | Carlos | Silva     | Imperador 453 | Fortaleza | 34895472 |
+----+--------+-----------+--------------+-----------+----------+
1 row in set (0.00 sec)

mysql> SELECT * FROM convidados WHERE nome = 'Carlos' OR sobrenome = 'Silva';
+----+---------+-----------+-------------------------+-----------+----------+
| id | nome    | sobrenome | endereco                | cidade    | telefone |
+----+---------+-----------+-------------------------+-----------+----------+
|  1 | Alberto | Silva     | Castelo Branco 123      | fortaleza | 32584930 |
|  3 | Carlos  | Moura     | Pedro Pereira 438       | Fortaleza | 34869353 |
|  5 | Eliana  | Silva     | Joaquim Mota 78         | Caucaia   | 87594537 |
|  8 | Vitor   | Silva     | Conjunto Ceara Av.C 453 | Fortaleza | 87947435 |
|  9 | Carlos  | Andrade   | Santa Adélia 274        | Eusebio   | 91058249 |
| 11 | Carlos  | Silva     | Imperador 453           | Fortaleza | 34895472 |
+----+---------+-----------+-------------------------+-----------+----------+
6 rows in set (0.00 sec)
```

Exemplo **BETWEEN**

O comando BETWEEN permite fazer a seleção de um intervalo, entre um e outro. A sintaxe da cláusula BETWEEN é a seguinte:

➔ **SELECT * FROM alunos WHERE idade BETWEEN 10 AND 20;**

Este comando irá selecionar todas as linhas cuja coluna tiver um valor entre 10 e 20. Os valores podem ser números, texto ou datas. Poderíamos de outra forma obter o mesmo resultado que seria:

➔ **SELECT * FROM alunos WHERE idade > = 10 AND idade < = 20;**

EXEMPLO

```
mysql> SELECT * FROM alunos;
+----+---------+-------+-------------+----------+
| id | nome    | idade | responsavel | contato  |
+----+---------+-------+-------------+----------+
|  1 | Daniel  |    15 | Maria       | 87945252 |
|  2 | Rafael  |    18 | Francisca   | 32853952 |
|  3 | Juliana |    23 | Graça       | 91583206 |
|  4 | Hugo    |    16 | Dores       | 87935725 |
|  5 | Emerson |    25 | Eunice      | 34895325 |
|  6 | Lucas   |    17 | João        | 32853925 |
|  7 | Sara    |    14 | Maria       | 87936306 |
|  8 | Natalia |    28 | Leonardo    | 91328536 |
+----+---------+-------+-------------+----------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM alunos WHERE idade BETWEEN 10 AND 20;
+----+--------+-------+-------------+----------+
| id | nome   | idade | responsavel | contato  |
+----+--------+-------+-------------+----------+
|  1 | Daniel |    15 | Maria       | 87945252 |
|  2 | Rafael |    18 | Francisca   | 32853952 |
|  4 | Hugo   |    16 | Dores       | 87935725 |
|  6 | Lucas  |    17 | João        | 32853925 |
|  7 | Sara   |    14 | Maria       | 87936306 |
+----+--------+-------+-------------+----------+
5 rows in set (0.46 sec)

mysql> SELECT * FROM alunos WHERE idade >= 10 AND idade <= 20;
+----+--------+-------+-------------+----------+
| id | nome   | idade | responsavel | contato  |
+----+--------+-------+-------------+----------+
|  1 | Daniel |    15 | Maria       | 87945252 |
|  2 | Rafael |    18 | Francisca   | 32853952 |
|  4 | Hugo   |    16 | Dores       | 87935725 |
|  6 | Lucas  |    17 | João        | 32853925 |
|  7 | Sara   |    14 | Maria       | 87936306 |
+----+--------+-------+-------------+----------+
5 rows in set (0.00 sec)
```

EXERCICIO

Dada a tabela **Alunos** a seguir, escreva os comandos **SQL** que:

Matricula	Nome	Sexo	Idade
1	Marcelo Medeiros	M	35
2	Ana Paula Berlim	F	25
3	Lucas Silva	M	7
4	Caroline Silva	F	19
5	Djalma Medeiros	M	65
6	Artur Paes	M	5
7	Eduarda Duda	F	8

1. Crie a tabela alunos acima

2. Insira os dados corretamente.

3. Liste todos os alunos do sexo masculino

4. Liste todos os alunos que possuem o sobrenome Medeiros, ordenados por idade.

5. Liste a média de idade dos alunos.

6. Mostre a maior idade dos alunos.

7. Liste a quantidade de alunos com idade menor que 20 anos.

8. Liste a Matricula e a idade do aluno chamado ‘Lucas Silva’.

9. Informe o nome e a idade do aluno mais jovem.

10. Liste a quantidade de alunos com idade entre 10 e 20 anos que sejam do sexo masculino.

11. Liste a quantidade de mulheres cadastradas na tabela Alunos.

12. Informe o sexo e a quantidade de ocorrências agrupadas por sexo.

13. Liste os nomes dos alunos que começam com a letra 'A' e de sexo masculino.

14. Altere a idade da aluna Caroline Silva para 20 anos.

15. Exclua a aluna Djalma Medeiros da tabela.

FASE IV : DESENVOLVIMENTO DO PROJETO

APOSTILA DE BANCO DE DADOS
CRIANDO SISTEMA DE VIDEO LOCADORA

INTRODUÇÃO

Nesse módulo faremos um sistema para gerenciamento de vídeo locadora em Java, visando tornar mais rápido o atendimento a seus clientes, atendendo a necessidade de cadastros de filmes e clientes e o gerenciamento da locação e devolução de filmes. Para isso vamos utilizar o banco de dados Mysql para guardar os dados, e a IDE Netbeans para desenvolver o projeto com uma interface amigável para o usuário.

O Objetivo geral desse módulo é fazer com que estudantes de Java possam aprender a fazer conexão com banco de dados e realizar as principais funcionalidades (cadastrar, consultar, excluir e alterar). Espero que com o término desse módulo você possa ter aprendido o bastante para poder desenvolver seus próprios sistemas e saber interagir a sua aplicação com qualquer banco de dados.

As chamadas vídeo locadoras são locais que se disponibilizam locações de vários filmes, oferecidos na locadora, para clientes cadastrados. Os filmes são divididos em várias categorias (ação, aventura, romance, etc.), cada um dos filmes contém a sua classificação e seu respectivo valor do aluguel. Em uma locadora podemos ter vários DVDs de um mesmo filme, então na verdade é alugado o DVD e não o filme, por isso foi dividido o cadastro de filme e o cadastro de DVD.

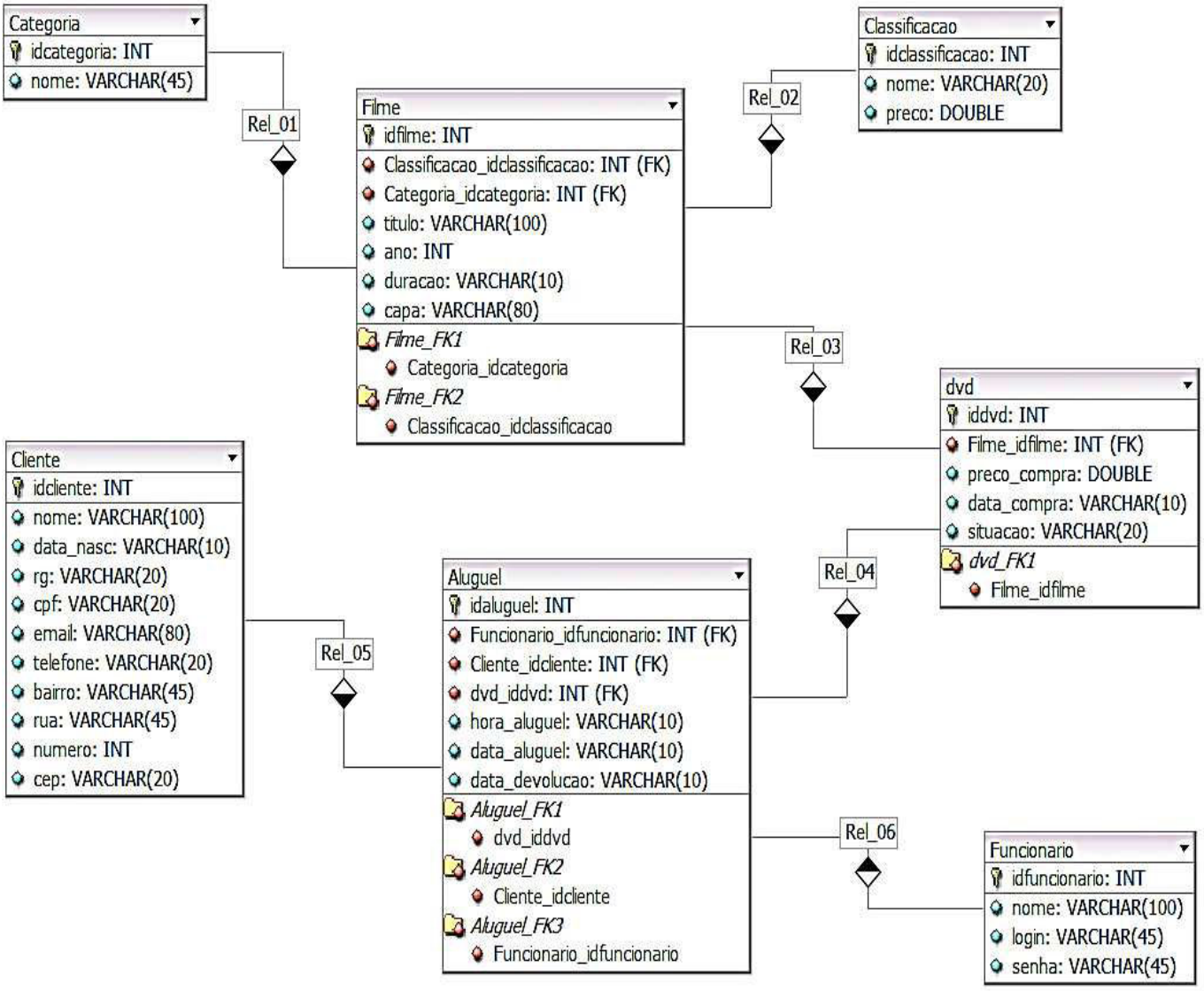
Locações, devoluções e cadastros, deixarão de ser feitos em blocos de papel, passando a ser executados em um sistema próprio para locadora, melhorando o desempenho dos funcionários e o atendimento aos clientes. Assim, será possível buscar um determinado filme para saber se está ou não na locadora ou até mesmo pesquisar as locações realizadas, através de pesquisas que poderão ser realizadas no sistema rapidamente.

Como serão muitas informações não vou me prender em detalhes, como já foi dito o objetivo principal é ensinar a manipular dados do banco, então caso fale pouco sobre algum assunto é porque esse não é o foco. Então se você tiver alguma dúvida procure ajuda em outras e peça a Judá ao seu professor, pois poderá precisar desse conhecimento mais na frente.

MODELAGEM DOS DADOS

A figura abaixo ilustra o modelo de dados do nosso sistema de vídeo locadora, preste atenção a cada detalhe e relacionamentos porque será através desse diagrama que vamos criar o nosso banco de dados e as referidas chaves estrangeiras nas tabelas.

Por exemplo, a tabela Filme possui relacionamento com outras três tabelas (Categoria, Classificação e dvd), onde as chaves primarias(PK) das tabelas Categoria e Classificação são chaves estrangeiras(FK) na tabela Filme, e a chave primaria(PK) da tabela Filme é chave estrangeira(FK) na tabela dvd e assim por diante acontece com as outras tabelas.



Modelagem do Banco de Dados Locadora

CRIAÇÃO DO BANCO DE DADOS

Faremos agora o script do banco de dados que intitulamos de locadora, com base no que já foi ensinado sobre criação de banco de dados e tabelas, crie os seguintes comandos SQL.

- **Criação do banco de dados locadora**

```
mysql> create database locadora;
Query OK, 1 row affected (0.00 sec)
```

- **Usando o banco locadora**

```
mysql> use locadora;
Database changed
```

- **Criação da tabela funcionário**

```
mysql> create table funcionario(
    -> idfuncionario int auto_increment primary key,
    -> nome varchar(100) not null,
    -> login varchar(45) not null,
    -> senha varchar(45) not null);
Query OK, 0 rows affected (0.01 sec)
```

- **Criação da tabela cliente**

```
mysql> create table cliente(
    -> idcliente int auto_increment primary key,
    -> nome varchar(100) not null,
    -> data_nasc varchar(10) not null,
    -> rg varchar(20),
    -> cpf varchar(20),
    -> email varchar(80),
    -> telefone varchar(20) not null,
    -> bairro varchar(45) not null,
    -> rua varchar(45) not null,
    -> numero int not null,
    -> cep varchar(20));
Query OK, 0 rows affected (0.02 sec)
```

- **Criação da tabela categoria**

```
mysql> create table categoria(
    -> idcategoria int auto_increment primary key,
    -> nome varchar(45) not null);
Query OK, 0 rows affected (0.00 sec)
```

- **Criação da tabela classificação**

```
mysql> create table classificacao(
    -> idclassificacao int auto_increment primary key,
    -> nome varchar(20) not null,
    -> preco double not null);
Query OK, 0 rows affected (0.00 sec)
```

- **Criação da tabela filme**

```
mysql> create table filme(
    -> idfilme int auto_increment primary key,
    -> titulo varchar(100) not null,
    -> ano int,
    -> duracao varchar(10),
    -> idcategoria int not null,
    -> idclassificacao int not null,
    -> capa varchar(80),
    -> foreign key(idcategoria) references categoria(idcategoria),
    -> foreign key(idclassificacao) references classificacao(idclassificacao));
Query OK, 0 rows affected (0.01 sec)
```

- **Criação da tabela dvd**

```
mysql> create table dvd(
    -> iddvd int auto_increment primary key,
    -> idfilme int not null,
    -> preco_compra double,
    -> data_compra  varchar(10),
    -> situacao varchar(20) not null,
    -> foreign key(idfilme) references filme(idfilme));
Query OK, 0 rows affected (0.04 sec)
```

- **Criação da tabela aluguel**

```
mysql> create table aluguel(
    -> idaluguel int auto_increment primary key,
    -> iddvd int not null,
    -> idcliente int not null,
    -> hora_aluguel varchar(10) not null,
    -> data_aluguel varchar(10) not null,
    -> data_devolucao varchar(10) not null,
    -> foreign key(idcliente) references cliente(idcliente),
    -> foreign key(iddvd) references dvd(iddvd));
Query OK, 0 rows affected (0.01 sec)
```

- **Inserir o 1º funcionário Administrador**

```
mysql> insert into funcionario values(0,'Administrador','qwe','123');
Query OK, 1 row affected (0.00 sec)
```

CRIAÇÃO DO PROJETO

Vamos agora criar um novo projeto no Netbeans chamado SistemaLocadora com os seguintes pacotes descritos abaixo:

- **Pacote DAO**

Tudo relacionado ao banco de dados se encontra nesse pacote, as classes estão divididas de acordo com o modelo para que fiquem mais organizadas.

- **Pacote Imagens**

Todas as imagens do nosso projeto se encontram dentro desse pacote, tudo isso para nível de organização.

- **Pacote Locacao**

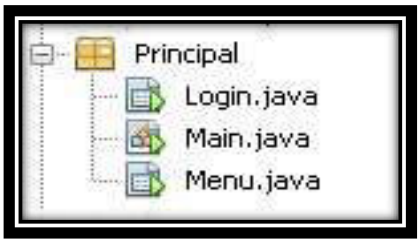
Esse pacote faz parte da interface com usuário(JFrame), porém ele foi separado do pacote Visão por motivo de um melhor controle das suas classes.

- **Pacote Modelo**

Dentro do pacote modelo temos todas as classes que possuem as variáveis encapsuladas com os métodos getters e setters, chamada também de lógica de negócios.

- **Pacote Principal**

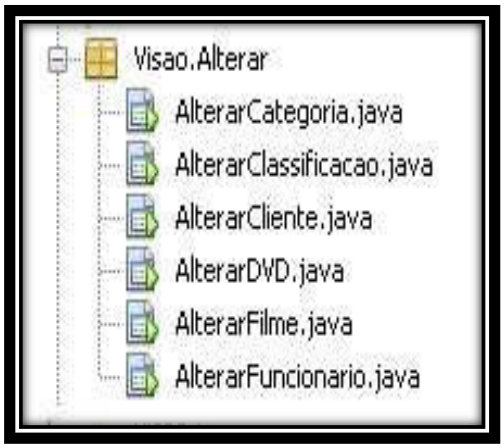
Nesse pacote temos a classe que possui o método Main, essa classe é responsável por chamar a tela de Login que por sua vez é responsável por chamar a tela do Menu.



- **Pacotes de Visao**

Foi dividido os pacotes que possuem telas (JFrame) de forma bastante intuitiva, ou seja, todos possuem o nome Visao seguido da sua ação (Visao.Alterar, Visao.Cadastrar, Visao.Consultar, Visao.Excluir)

Visao.Alterar



Visao.Cadastrar

Visao.Consultar

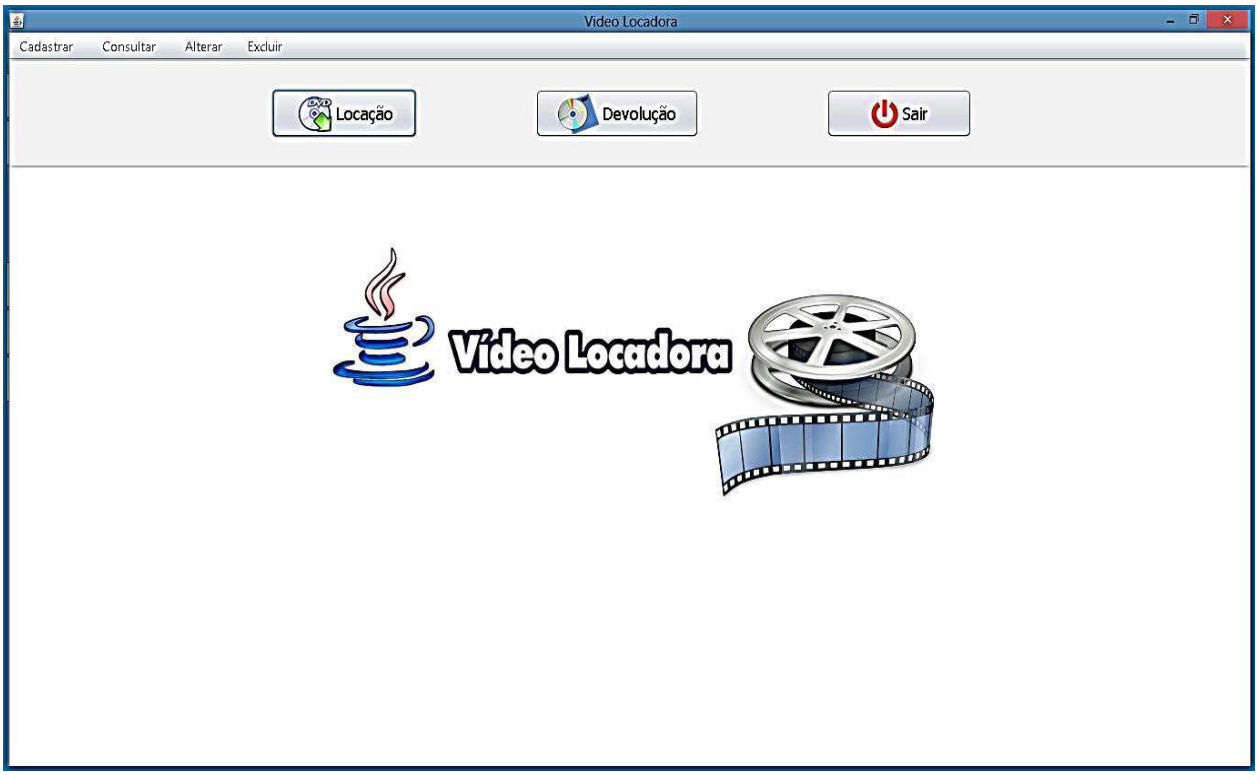
Visao.Excluir

APRESENTAÇÃO DAS TELAS

A interface foi toda desenvolvida com o auxílio da paleta do netbeans, porém procure sempre ver os códigos que estão sendo criado a partir da ação de puxar e arrastar os componentes. Agora iremos mostrar as telas do sistema, procure fazer as suas o mais próximo possível, sem esquecer nenhum dos componentes.

- **Tela de Login**

- **Menu Principal**



- **Cadastro de Funcionário**

- **Cadastro de Cliente**

- **Cadastro de Categoria**

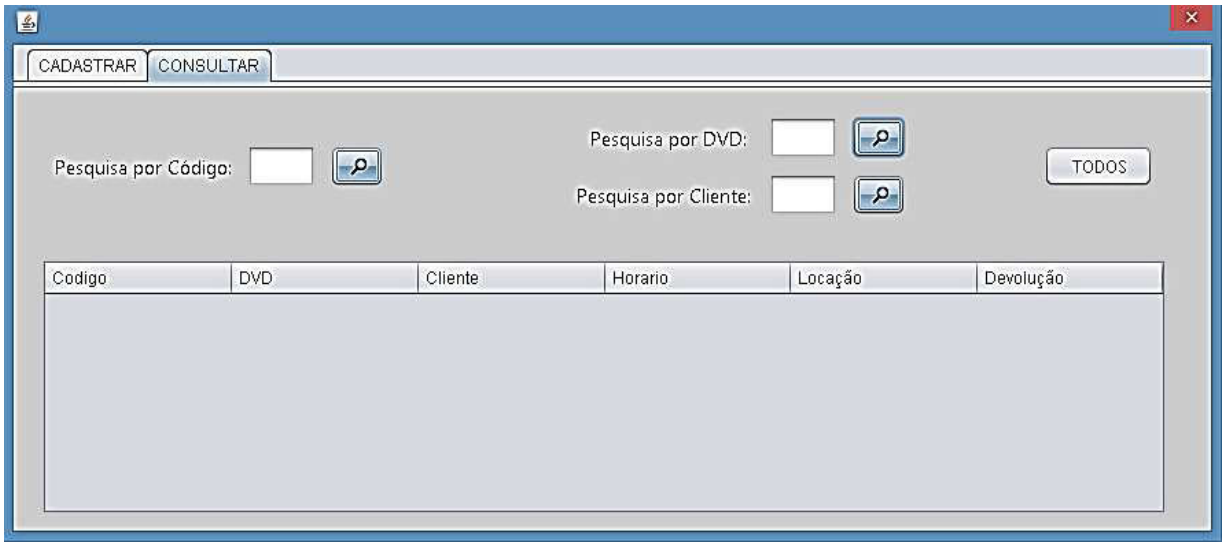
- **Cadastro de Classificação**

- **Cadastro de Filme**

- **Cadastro de DVD**

- **Cadastro de Locação**

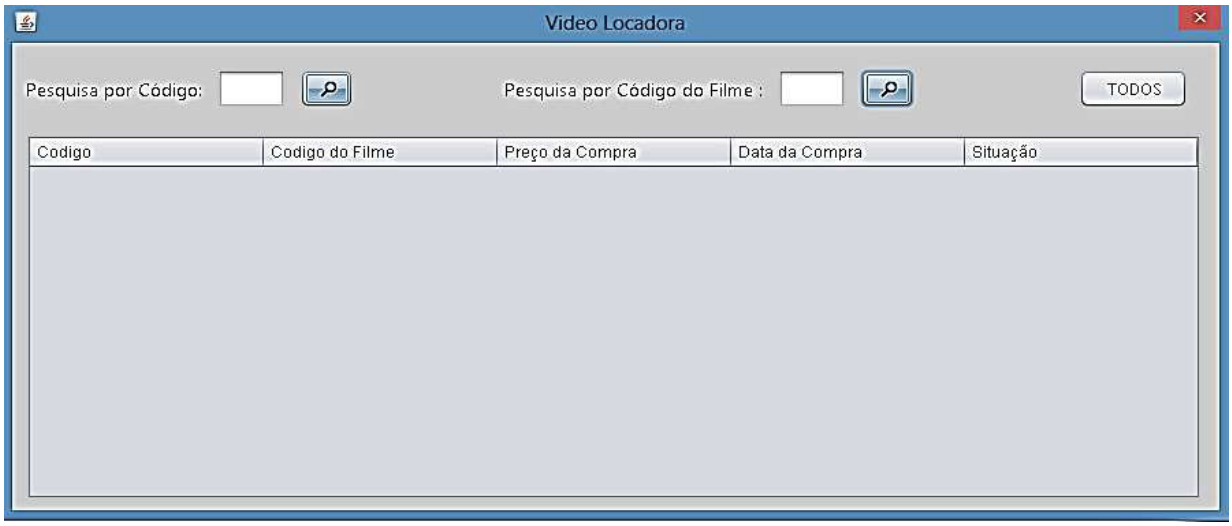
- **Consulta de Locação**



- **Consulta de Cliente**

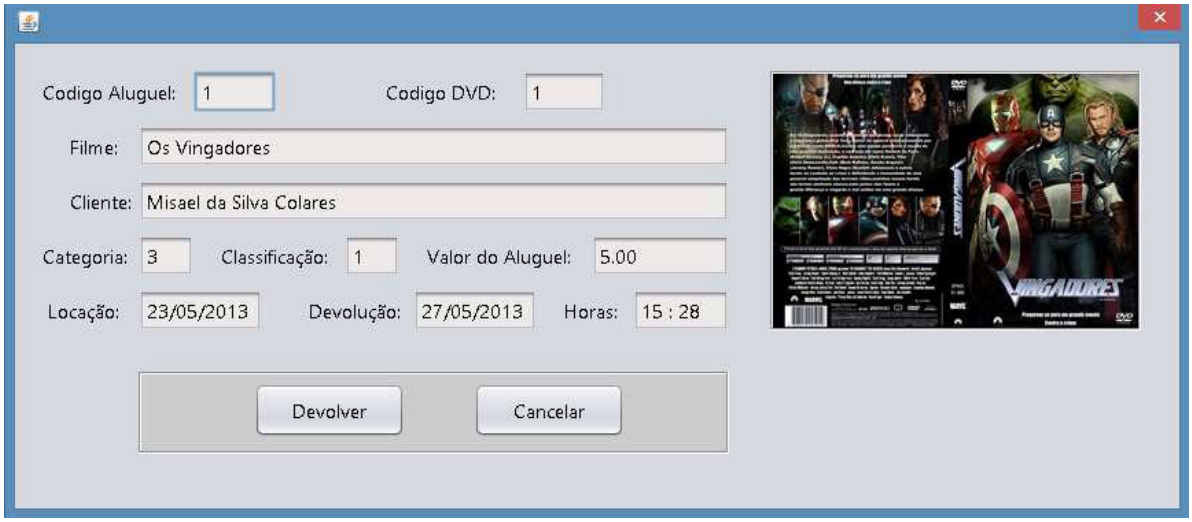
- **Consulta de Filme**

- **Consulta de DVD**



- **Consulta para Devolução**

- **Exemplo de Realizar Devolução**



- **Alterar e Excluir**

As telas de alterar seguem o mesmo padrão das de cadastros, porém acrescenta um campo para digitar o código que deseja que sofra alteração, por esse motivo vou mostrar apenas o de alterar funcionário como exemplo. E as telas de excluir seguem um padrão para todas, mudando apenas detalhes, por esse motivo vou mostrar apenas o de excluir cliente como exemplo.

DEFINIÇÃO DAS CLASSES DE MODELO

Sabemos que encapsular significa proteger o acesso direto aos atributos de uma classe**,** usando para a manipulação desses atributos os métodos **set** e **get**. Agora iremos mostrar os atributos de cada classe para que você crie os métodos get e set. Para gerar os métodos o netbeans tem um atalho que ajuda, Alt + Insert.

```
public class Categoria{

    //atributos
    private int codigo;
    private String nome;

    //metodos getters e setters
```

```
public class Classificacao {

    //atributos
    private int codigo;
    private String nome;
    private double preco;

    //metodos getters e setters
```

```
public class DVD {

    //atributos
    private int codigo;
    private int cod_filme;
    private String situacao;
    private double preco;
    private String data_compra;

    //metodos getters e setters
```

```
public class Funcionario {

    //atributos
    private int cod;
    private String nome;
    private String login;
    private String senha;

    //metodos getters e setters
```

```
public class Filme {

    //atributos
    private int codigo;
    private String titulo;
    private int ano;
    private String duracao;
    private int cod_categoria;
    private int cod_classificao;
    private String capa;

    //metodos getters e setters
```

```
public class Cliente {

    //atributos
    private int Codigo;
    private String Nome;
    private String Nascimento;
    private String RG;
    private String CPF;
    private String Telefone;
    private String Email;
    private String Bairro;
    private String Rua;
    private int Numero;
    private String CEP;

    //metodos getters e setters
```

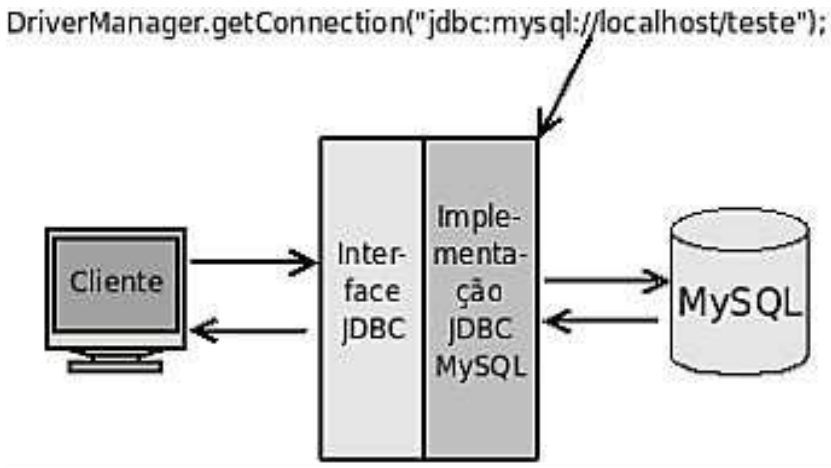
```
public class Aluguel {

    //atributos
    private int cod;
    private int coddvd;
    private int codcliente;
    private String data_aluguel;
    private String horario;
    private String data_devolucao;

    //metodos getters e setters
```

CONEXÃO COM BANCO DE DADOS

Conectar-se a um banco de dados com Java é bastante simples. O Java possui um único conjunto de interfaces muito bem definidas que devem ser implementadas. Esse conjunto de interfaces fica dentro do pacote java.sql e nos referiremos a ela como **JDBC** (Java Database Connectivity). Sem um JDBC não é possível fazer uma conexão a um banco de dados**.**



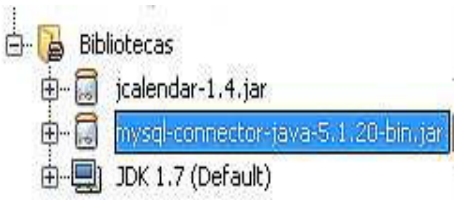
Definição do Driver

A classe DriverManager é a responsável por se comunicar com todos os drivers que você deixou disponível. Para isso, invocamos o método estático getConnection com uma String que indica a qual banco desejamos nos conectar. Essa String é chamada de **String de conexão JDBC**, a que utilizaremos para acessar o nosso banco de dados tem a seguinte forma:

- ✓ **jdbc:mysql://localhost/locadora**

Agora precisamos adicionar a biblioteca do nosso sistema o conector MySQL que possibilitar a conexão com o Java, sem ele nada feito, o download do conector está disponível no site oficial do MySQL: http://dev.mysql.com/downloads/. No site, clique em Connectors e depois em Connector/J. Faça o download e descompacte o arquivo. Serão descompactados vários arquivos, porém o arquivo que nos interessa é o que tem extensão JAR.

Agora no projeto do Netbeans click com o botão direito na pasta Bibliotecas e escolha Adicionar JAR/Pasta… procure entre os arquivos descompactados um que tenha extensão.JAR e o adicione. Nesse exemplo utilizamos a versão do conector mysql-connector-java-5.1.20-bin.jar



- **Criação da Classe Conexão**

```
1   package DAO;
2
3   import java.sql.*;
4   import javax.swing.JOptionPane;
5
6   public class Conexao {
7
8       public static Connection AbrirConexao() {
9           Connection con = null;
10          try {
11              Class.forName("com.mysql.jdbc.Driver");
12              String url = "jdbc:mysql://localhost/locadora";
13              con = DriverManager.getConnection(url, "root", "123");
14
15          } catch (Exception e) {
16              JOptionPane.showMessageDialog(null, "Erro na Conexão com o Banco",
17                      "Video Locadora", JOptionPane.ERROR_MESSAGE);
18              e.getMessage();
19          }
20          return con;
21      }
22
23      public static void FecharConexao(Connection con) {
24          try {
25              con.close();
26
27          } catch (Exception e) {
28              System.out.println(e.getMessage());
29          }
30      }
31  }
```

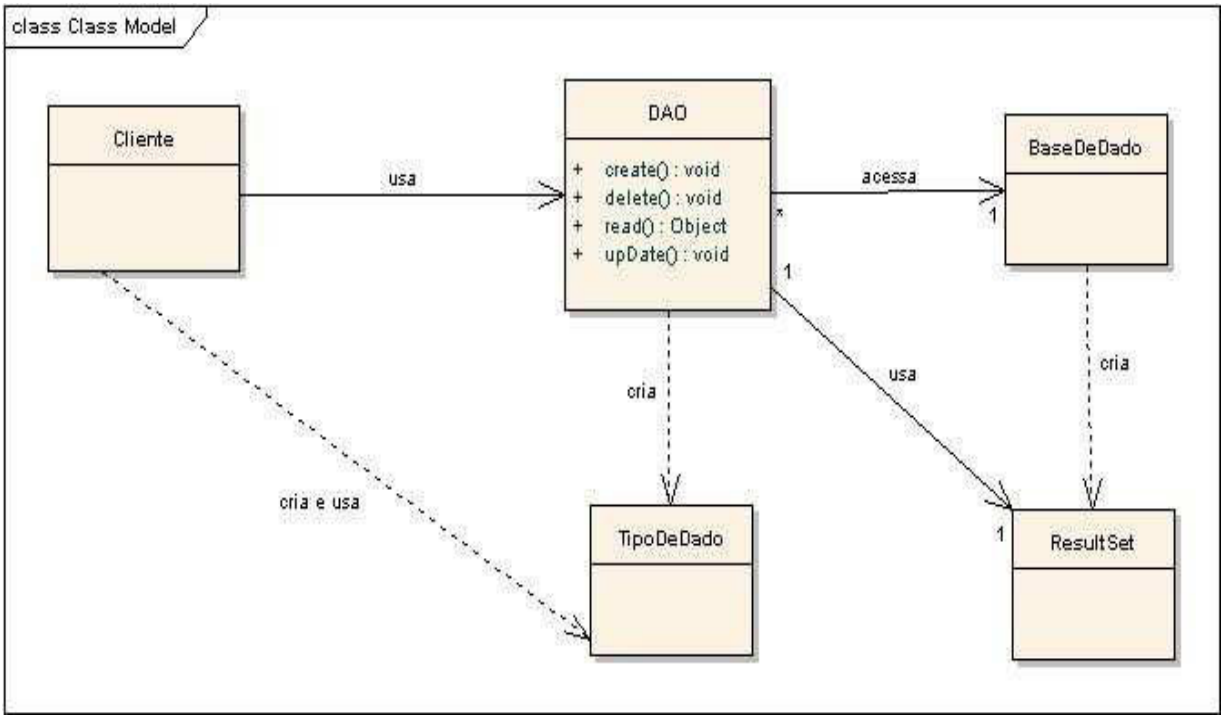
A classe de Conexão

- ➔ **Linha 3:** importa as classes do pacote sql necessárias para conexão com banco de dados.
- ➔ **Linha 8:** cria o método AbrirConexao() do tipo Connection.
- ➔ **Linha 9:** cria o objeto con da classe Connection.
- ➔ **Linha 10:** o tratamento de exceções em Java se dá através dos blocos try, catch. Tudo que estiver dentro do bloco try será executado até que alguma exceção seja lançada, ou seja, até que algo dê errado.
- ➔ **Linha 11:** carrega o driver que será usado pela aplicação Java para realizar a comunicação com o banco de dados.
- ➔ **Linha 12:** declara a url necessária para conexão passando o nome do banco de dados que iremos usar.
- ➔ **Linha 13:** estabelece uma conexão por meio do objeto con usando a String url e passando o login e a senha que foram definidas no momento da instalação do Mysql.
- ➔ **Linha 15:** caso a conexão não possa ser realizada por qualquer motivo ocorre a exceção onde passa a executar o bloco catch.
- ➔ **Linha 16, 17:** envia uma mensagem ao usuário informando que houve erro na conexão com o banco de dados.
- ➔ **Linha 20:** retorna o objeto con, se tudo ocorrer bem o objeto con recebeu a conexão na linha 13.
- ➔ **Linha 23:** cria o método FecharConexao() recebendo por parâmetro um objeto da classe Connection.
- ➔ **Linha 25:** contém o método close() que encerra a conexão criada.

- **O Padrão Data Access Object (DAO)**

Misturar a lógica de persistência com a lógica de aplicação cria uma dependência direta entre a implementação da aplicação e do armazenamento persistente. Tal dependência de código nos componentes torna difícil migrar a aplicação de um tipo de fonte de dados para outro.

O principal objetivo de um DAO é encapsular o acesso e a manipulação de dados em uma camada separada, por isso organizamos os recursos de lógica de acesso a dados e encapsulamos recursos proprietários para Facilitar a capacidade de Manutenção e a portabilidade.



A estrutura do DAO

- **Cliente** – o cliente é um objeto que requer acesso à fonte de dados para obter e armazenar dados.
- **DAO** – o DAO abstrai a implementação de acesso a dados para o cliente a fim de permitir um acesso transparente a fonte dedados.
- **BaseDeDado** – representa uma implementação de fonte de dados.
- **ResultSet** – representa os resultados de uma execução de consulta.
- **TipoDeDado** – representa um objeto de transferência usado como um carregador de dados.

CRIAÇÃO DA SUPERCLASSE ExecuteSQL.java.....

Levando em consideração que já sabemos o que é herança no Java, podemos dizer que uma subclasse herda métodos e atributos de sua superclasse; apesar disso, pode escrevê-los novamente para uma forma mais específica de representar o comportamento do método herdado. Na nossa aplicação todas as classes DAO herdam da classe ExecuteSQL.java que possui os métodos getCon e setCon, isso para evitar repetições desses métodos em todas as outras classes.

```
1   package DAO;
2
3   import java.sql.*;
4
6   public class ExecuteSQL {
7
8       private Connection con;
9
10      public ExecuteSQL(Connection con) {
11          setCon(con);
12      }
13
14      public Connection getCon() {
15          return con;
16      }
17
18      public void setCon(Connection con) {
19          this.con = con;
20      }
21
22  }
```

A superclasse ExecuteSQL.java

➔ **Linha 7:** declara o objeto con da classe Connection como private, ou seja, não pode ser acessado diretamente por outras classes.

➔ **Linha 9:** cria o método construtor da classe, um construtor tem sempre o mesmo nome da classe e é onde tem as instruções que serão executadas sempre que for instanciado um objeto desta classe.

➔ **Linha 13, 17:** cria os métodos getCon e setCon, para acessar o objeto con definido como private na linha 7, os métodos são public, ou seja, são vistos por qualquer outra classe do projeto.

- **Criação do Método Logar()**

```
5   package DAO;
6
8   import java.sql.*;
8   import Modelo.Funcionario;
9
10  public class FuncionarioDAO extends ExecuteSQL {
11
12      public FuncionarioDAO(Connection con) {
13          super(con);
14      }
15      public boolean Logar(String login, String senha) {
16          boolean finalResult = false;
17          try {
18              String consulta = "select login, senha from funcionario "
19            + "where login = '" + login + "' and senha = '" + senha + "'";
20              PreparedStatement ps = getCon().prepareStatement(consulta);
21              ResultSet rs = ps.executeQuery();
22
23              if (rs != null) {
24                  while (rs.next()) {
25                      Funcionario a = new Funcionario();
26                      a.setLogin(rs.getString(1));
27                      a.setSenha(rs.getString(2));
28                      finalResult = true;
29                  }
30              }
31          } catch (SQLException ex) {
32              ex.getMessage();
33          }
34          return finalResult;
35      }
```

Método Logar() da classe FuncionarioDAO.java

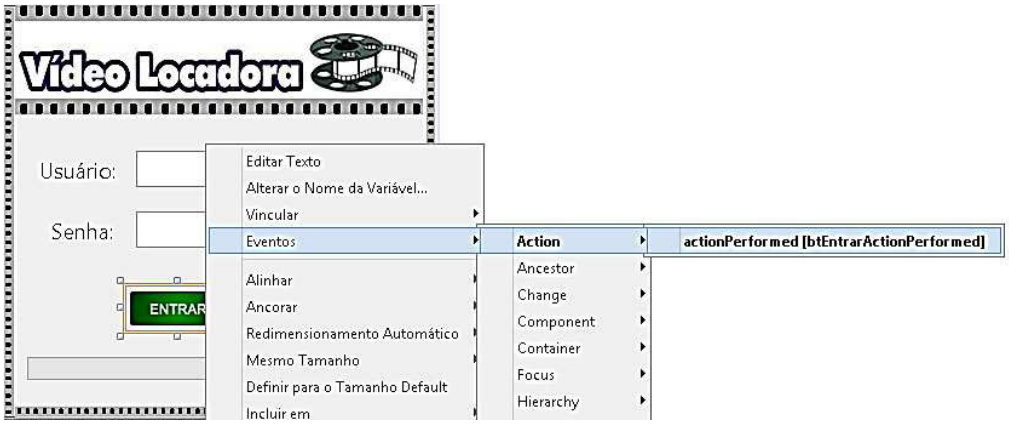
➔ **Linha 8:** importa a classe Funcionário do pacote Modelo necessária para o uso dos métodos getters e setters.

➔ **Linha 10:** a classe FuncionarioDAO é subclasse da ExecuteSQL.java. 'extends' é uma palavra reservada do Java, para definir uma relação de herança entre classes.

➔ **Linha 13:** A palavra super representa uma chamada de método ou acesso a um atributo da superclasse, por isso tem esse nome. No nosso caso, estamos usando o super para invocar construtor da superclasse ExecuteSQL.java.

- ➔ **Linha 15:** cria o método Logar() do tipo boolean, ou seja, irar retornar true ou false. O método recebe dois valores por parâmetro que será os valores de login e senha digitada pelo usuário no momento de entrar no sistema.
- ➔ **Linha 16:** cria a variável finalResult do tipo boolean recebendo como valor false.
- ➔ **Linha 18, 19:** cria o script de comandos em SQL que será usado na consulta ao banco de dados e o armazena na String consulta. O comando manda basicamente selecionar o login e a senha da tabela funcionário onde login e senha forem iguais aos valores das variáveis passadas para o método.
- ➔ **Linha 20:** cria um objeto chamado ps a partir da interface PreparedStatement que possibilita a execução de um script SQL pelo método getCon() herdado da classe executeSQL.java.
- ➔ **Linha 21:** cria um objeto chamado rs a partir da interface ResultSet. Como o próprio nome sugere, o objeto rs será usado para armazenar o resultado gerado pelo script SQL por meio do método executeQuery. Pode-se dizer que o objeto rs armazena o resultado da ação efetuada pelo script SQL.
- ➔ **Linha 23:** verifica se o objeto rs é diferente (!=) de nulo, ou seja, se o objeto armazenou o resultado gerado pelo script SQL (linha 21), se não for nulo ele passa para estrutura de repetição while (linha 24).
- ➔ **Linha 24:** realiza a varredura de todos os registros armazenados no objeto rs usando while(rs.next()), isto é, o loop é executado enquanto existirem registros no objeto rs. Desta forma, todos os registros do objeto rs são recuperados pelo método getString (linha 26 e 27).
- ➔ **Linha 25:** instancia um novo objeto da classe Funcionario.java que chamamos de “a”. Pelo objeto podemos acessar os métodos da classe, que no caso iremos usar os métodos setLogin e setSenha (linha 26 e 27).
- ➔ **Linha 28:** se tudo ocorrer bem a variável booleana finalResult passará a receber um novo valor, ou seja, de false ela passa a ser true, isso servirá para quando formos usar esse método na ação do botão entrar.

AÇÃO DE O BOTÃO ENTRAR




```
149     private void btEntrarActionPerformed(java.awt.event.ActionEvent evt) {
150         Connection con = Conexao.AbrirConexao();
151         FuncionarioDAO sql = new FuncionarioDAO(con);
152         String login = tfUsuario.getText();
153         String senha = pfSenha.getText();
154         if (login.equalsIgnoreCase("") || senha.equalsIgnoreCase("")) {
155             JOptionPane.showMessageDialog(null, "Nenhum campo pode esta Vazio",
156                     "Video Locadora", JOptionPane.WARNING_MESSAGE);
157             tfUsuario.setText("");
158             pfSenha.setText("");
159         } else {
160             if (sql.Logar(login, senha) == true) {
161                 new Thread() {
                        public void run() {
163                         for (int i = 0; i < 101; i++) {
164                             jProgressBar.setValue(i);
165                             try {
166                                 Thread.sleep(35);
167                             } catch (Exception ex) {
168                                 ex.getMessage();
169                             }
170                         }
171                         new Menu().setVisible(true);
172                         dispose();
173                     }
174                 }.start();
175             } else {
176                 JOptionPane.showMessageDialog(null, "Usuário ou Senha Invalidos",
177                         "Video Locadora", JOptionPane.ERROR_MESSAGE);
178                 tfUsuario.setText("");
179                 pfSenha.setText("");
180             }
181         }
```

Ação do botão entrar

➔ **Linha 150:** o objeto con da classe Connection recebe o retorno do método AbrirConexao() da classe Conexao.java.

➔ **Linha 151:** instância um novo objeto da classe FuncionarioDAO.java chamado "sql" e passa para o método construtor da classe o objeto con.

➔ **Linha 152, 153:** cria as variáveis login e senha do tipo String para receber os valores digitado pelo usuário nos campos tfUsuario e tfsenha, os nomes dos campos foram dados seguindo o padrão textfield = tf + nome.

➔ **Linha 154:** verifica se as variáveis estão vazias, se algum dos campos no momento do click do botão estiver vazio emite uma mensagem para o usuário (linhas 155, 156) informando que "nenhum dos campos pode estar vazio".

➔ **Linha 160:** verifica se o retorno do método Logar() é igual a true, ou seja, nesse momento pelo objeto sql é chamado o método Logar() passando as variáveis login e senha definidas nas linhas(152 e 153) e no mesmo instante já verifica o retorno do método.

➔ **Linha 161:** depois de verificado se o login e a senha do usuário estão corretas(linha 160) passa a carregar a barra de progresso, Em Java, usamos a classe Thread do pacote java.lang para criarmos *linhas de execução* paralelas. A classe Thread recebe como argumento um objeto com o código que desejamos rodar. No nosso caso, a barra de progresso.

➔ **Linha 162:** a tarefa a ser executado pelo thread deverá ser descrita pelo método run(). Ou seja, toda tarefa de carregar a barra de progresso e abrir a tela do menu principal após o carregamento, se encontra dentro desse método.

➔ **Linha 163:** cria a estrutura de repetição for, incrementando a variável i e enviando para o método setValue() da barra de progresso(linha 164) , isso faz com que a barra fique enchendo.

➔ **Linha 166:** define a velocidade que será cheia a barra de progresso, através do método sleep() da classe Thread.

➔ **Linha 174:** após definido a ação que será realizada pela Thead dentro do método run() é chamado o método start(), ou seja, é nesse momento que o que foi definido será executado.

➔ DECLARANDO O MÉTODO MAIN

Como já é de nosso conhecimento a classe que possui o método main é a primeira a ser executada no projeto, basicamente definimos essa classe para abrir a tela de login. Ela poderia ter qualquer nome mais escolhemos chamá-la de Main.java.

```
1  package Principal;
2
3  public class Main {
4
5      public static void main(String[] args) {
6          new Login().setVisible(true);
7      }
8  }
```

A classe Main.java

➔ **Linha 6**: A tela de Login chama o método setVisible() da classe JFrame passando por valor true, ou seja, a tela de Login é aberta nesse momento.

BARRA DE MENUS

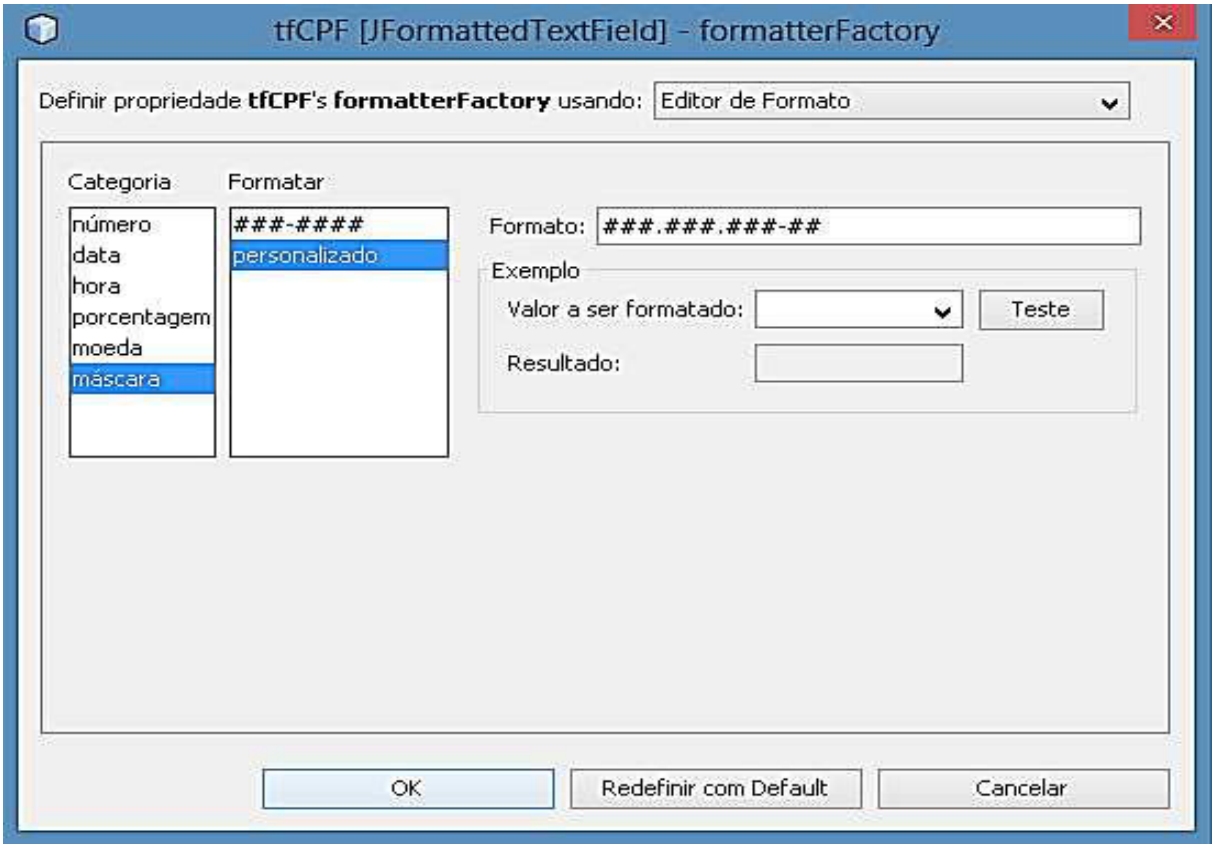
Foi criada uma barra de menus com as respectivas ações que serão realizadas (Cadastrar, Consultar, Alterar, Excluir) todas possuem os mesmos itens (Cliente, Filme, DVD, Categoria, Classificação, Funcionário), então a única ação é chamar as telas de cada um dos itens.

Barra de Menu

INSERINDO MÁSCARAS

Existem varias formas de inserir mascaras em campos de texto para que o usuário não digite, por exemplo, letras em datas ou até mesmo RG e CPF fora do padrão nacional. Vamos ensinar da maneira mais fácil, que é usando as propriedades do NetBeans.

Primeiramente é necessário que use o componente jFromattedTextField (campo de texto formatado). Depois de selecionado o componente, clique em propriedades e procure formatterFactory Vamos usar como exemplo o campo CPF do cadastro de cliente.java. Formato: ###.###.###-##



Formato da máscara CPF

CADASTRO DE CLIENTE

- **Criação do Método Inserir_Cliente( )**

```
26      public String Inserir_Cliente(Cliente a) {
27          String sql = "insert into cliente values(0,?,?,?,?,?,?,?,?,?,?)";
28          try {
29              PreparedStatement ps = getCon().prepareStatement(sql);
30
31              ps.setString(1, a.getNome());
32              ps.setString(2, a.getNascimento());
33              ps.setString(3, a.getRG());
34              ps.setString(4, a.getCPF());
35              ps.setString(5, a.getEmail());
36              ps.setString(6, a.getTelefone());
37              ps.setString(7, a.getBairro());
38              ps.setString(8, a.getRua());
39              ps.setInt(9, a.getNumero());
40              ps.setString(10, a.getCEP());
41
42
43              if (ps.executeUpdate() > 0) {
44                  return "Inserido com sucesso.";
45              } else {
46                  return "Erro ao inserir";
47              }
48          } catch (SQLException e) {
49              return e.getMessage();
50          }
51
52      }
```

Método Inserir_Cliente() da classe ClienteDAO.java

➔ **Linha 27:** cria o script de comandos em SQL que será usado na consulta ao banco de dados e o armazena na String sql. Um ponto a observar é o uso do caractere de interrogação(?), que será substituído por parâmetros durante a execução do método.

➔ **Linha 31 – 40:** cada um dos parâmetros usado no script em SQL deve ser passado ao objeto ps por meio de métodos apropriados. Existem diversos métodos que podem ser usados, dependendo do tipo de valor, por exemplo: setString, setInt, setDouble etc. na linha 31 usamos os parâmetros 1(referente a primeira interrogação da linha 27) e a.getNome(referente ao conteúdo que será colocado no lugar do caractere de interrogação).

- **Ação de o Botão Cadastrar**

```
429    private void btSalvarActionPerformed(java.awt.event.ActionEvent evt) {
430        String nome = jTF_Nome.getText();
431        String nascimento = jTF_Nascimento.getText();
432        String cep = jTF_CEP.getText();
433        String rua = jTF_Rua.getText();
434        String numero = jTF_Numero.getText();
435        String bairro = jTF_Bairro.getText();
436        String email = jTF_Email.getText();
437        String fone = jTF_Telefone.getText();
438        String cpf = tfCPF.getText();
439        String rg = jTF_RG.getText();
440        if (nome.equals("") || nascimento.equals("") || cep.equals("")
441                || rua.equals("") || numero.equals("") || bairro.equals("")
442                || fone.equals("") || cpf.equals("") || rg.equals("")) {
443            JOptionPane.showMessageDialog(null, "nenhum campo pode estar vazio",
444                    "Video Locadora", JOptionPane.WARNING_MESSAGE);
445        } else {
446            Connection con = Conexao.AbrirConexao();
447            ClienteDAO sql = new ClienteDAO(con);
448            int n = Integer.parseInt(numero);
449            Cliente a = new Cliente();
450
451            a.setNome(nome);
452            a.setNascimento(nascimento);
453            a.setCEP(cep);
454            a.setRua(rua);
455            a.setNumero(n);
456            a.setBairro(bairro);
457            a.setEmail(email);
458            a.setTelefone(fone);
459            a.setCPF(cpf);
460            a.setRG(rg);
461
462            sql.Inserir_Cliente(a);
463            Conexao.FecharConexao(con);
464
465            jTF_Nome.setText("");
466            jTF_CEP.setText("");
467            jTF_Numero.setText("");
468            jTF_Bairro.setText("");
469            jTF_Email.setText("");
470            jTF_Telefone.setText("");
471            jTF_Rua.setText("");
472            jTF_Nascimento.setText("");
473            jTF_RG.setText("");
474            tfCPF.setText("");
475            JOptionPane.showMessageDialog(null, "Cadastro Realizado com Sucesso",
476                    "Video Locadora", JOptionPane.INFORMATION_MESSAGE);
477            dispose();
478
479        }
```

Ação do Botão Cadastrar

- ➔ **Linha 430 – 439:** guarda nas variáveis do tipo String os dados digitado pelo usuário nas TextFields, o método usado para pegar o que foi digitado é getText();
- ➔ **Linha 440 – 444:** verifica se nenhuma das variáveis estão vazias através do método equals(“ ”), se alguma estiver vazia é lançada uma mensagem informando que nenhum campo pode estar vazio.
- ➔ **Linha 448:** cria a variável n do tipo inteiro para armazenar a conversão da variável numero do tipo String, isso porque na tabela do banco de dados foi definido que o valor de numero seria do tipo inteiro.
- ➔ **Linha 451 – 460:** através do objeto “a” da classe Cliente.java é enviado para o método set as variáveis que armazenaram os dados digitados pelo usuário.
- ➔ **Linha 462:** através do objeto “sql” da classe ClienteDAO.java é enviado para o método Inserir_Cliente() o objeto “a” da classe Cliente.java.
- ➔ **Linha 463:** chama o método FecharConexao( ) da classe Conexao.java, é importante sempre que abrir uma conexão com banco de dados lembrar também de fechar.

USANDO ARRAYLIST

O Java, por padrão, possui uma série de recursos prontos (APIs) para que possamos tratar de estrutura de dados, também chamados de coleções (collections). Podemos dizer que ArrayList é uma classe para coleções. Uma classe genérica para ser mais exato. Coleções de qualquer tipo de dados e não somente de tipos primitivos.

Você pode criar seus objetos através de uma classe e agrupá-los através de ArrayList e realizar, nessa coleção, várias operações, como: adicionar e retirar elementos, ordená-los, procurar por um elemento específico, apagar um elemento específico, limpar o ArrayList dentre outras possibilidades.

Nós vamos precisar usar o ArrayList para agrupar os dados vindos do banco, para realizar os comandos INSERT, UPDATE ou DELETE não é preciso usar um

ArrayList porque nenhum desses comandos retorna registros, diferente de um comando SELECT onde vamos precisar agrupar os dados selecionados.

CONSULTA DE CLIENTE

- **Criação do Método ListarCliente().**

```
54      public List<Cliente> ListarCliente() {
55          String sql = "select idcliente,nome,rg,cpf,telefone,email from cliente";
56          List<Cliente> lista = new ArrayList<>();
57          try {
58              PreparedStatement ps = getCon().prepareStatement(sql);
59              ResultSet rs = ps.executeQuery();
60
61              if (rs != null) {
62                  while (rs.next()) {
63                      Cliente a = new Cliente();
64                      a.setCodigo(rs.getInt(1));
65                      a.setNome(rs.getString(2));
66                      a.setRG(rs.getString(3));
67                      a.setCPF(rs.getString(4));
68                      a.setTelefone(rs.getString(5));
69                      a.setEmail(rs.getString(6));
70
71                      lista.add(a);
72                  }
73                  return lista;
74              } else {
75                  return null;
76              }
77          } catch (SQLException e) {
78              return null;
79          }
80      }
```

Método ListarCliente() da classe ClienteDAO.java

➔ **Linha 54:** cria o método ListarCliente() do tipo List<Cliente>, isso porque o método retornará um ArrayList do tipo Cliente.

➔ **Linha 56:** cria um ArrayList do tipo Cliente com o nome lista.

➔ **Linha 71:** enquanto existirem registros no objeto rs(linha 62) o objeto "a" da classe Cliente será adicionado a lista através do método add(), esse método é usado para adicionar um elemento ao ArrayList.

➔ **Linha 73:** se o objeto “rs” for diferente de null (linha 61) então o método retornará a lista com os registros adicionados(linha 71) e devidamente agrupados.

- **Criação do Método AtualizaTable().**

O método AtualizaTable() deve ser o primeiro a ser chamado na tela de ConsultarCliente.java, se o primeiro método a ser executado é o método construtor então basta chamar o método AtualizaTable() dentro do construtor. Para identificar onde deve chamar o método basta procurar o initComponents() que é um método gerado pelo NetBeans designer de swing que fica dentro do construtor. Ele estabelece os componentes e define seus valores padrão.

```
43          initComponents();
44
45          setTitle("Vídeo Locadora");
46          setSize(970, 380);
47          AtualizaTable();
```

➔ **Linha 45:** define o titulo que aparecerá no JFrame através do método setTitle().

➔ **Linha 46:** define o tamanho do JFrame através do método setSize()

➔ **Linha 47:** chama o método AtualizaTable()

```
323      private void AtualizaTable() {
324
325          Connection con = Conexao.AbrirConexao();
326          ClienteDAO bd = new ClienteDAO(con);
327          List<Cliente> lista = new ArrayList<>();
328          lista = bd.ListarCliente();
329          DefaultTableModel tbm =
330                  (DefaultTableModel) jTable.getModel();
331          while (tbm.getRowCount() > 0) {
332              tbm.removeRow(0);
333          }
334          int i = 0;
335          for (Cliente tab : lista) {
336              tbm.addRow(new String[i]);
337              jTable.setValueAt(tab.getCodigo(), i, 0);
338              jTable.setValueAt(tab.getNome(), i, 1);
339              jTable.setValueAt(tab.getRG(), i, 2);
340              jTable.setValueAt(tab.getCPF(), i, 3);
341              jTable.setValueAt(tab.getTelefone(), i, 4);
342              jTable.setValueAt(tab.getEmail(), i, 5);
343              i++;
344          }
345          Conexao.FecharConexao(con);
346      }
```

Método AtualizaTable()

➔ **Linha 328:** o ArrayList de nome lista recebe o retorno do método ListarCliente() da classe ClienteDAO.java.

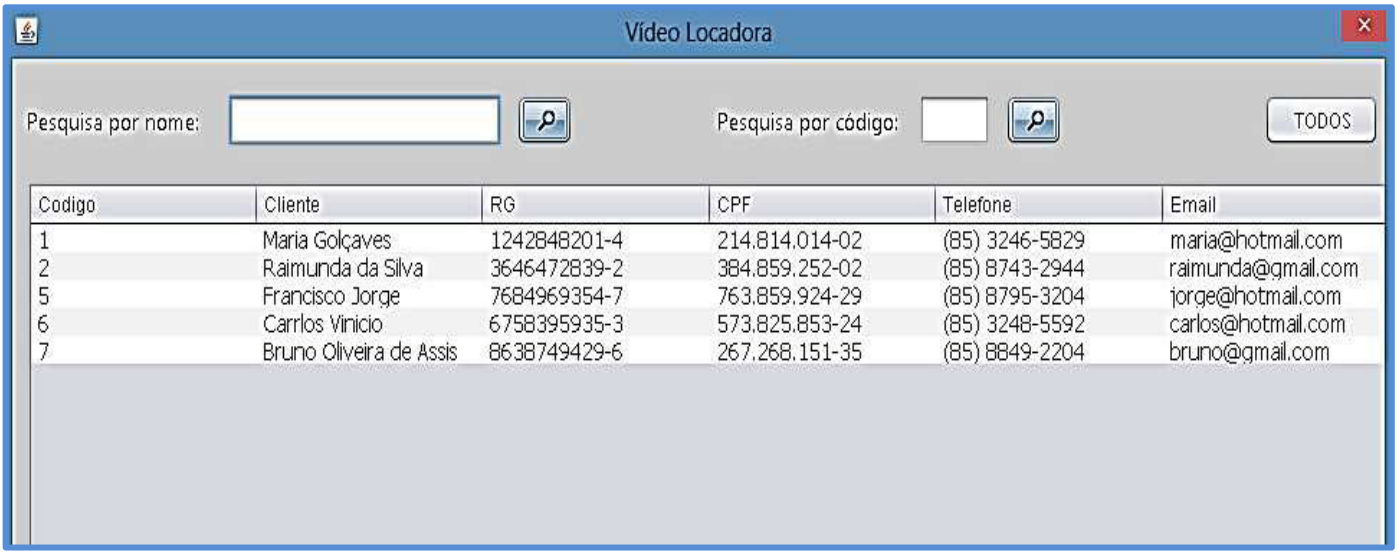
➔ **Linha 329, 330:** cria um objeto de nome “tbm” do tipo DefaultTableModel para trabalhar com um modelo que possa inserir deletar ou atualizar os dados de uma tabela, que em nosso caso a tabela se chama “jTable”, por isso que usamos o método getModel() (linha 330) para poder manipular a tabela jTable.

➔ **Linha 331 – 333:** cria a estrutura de repetição while para apagar os dados da tabela antes de inserir novos, o método getRowCount() retorna a quantidade total de linhas, então se a quantidade de linhas for maior que 0 ele entra na estrutura e usa o método RemoveRow() passando o valor 0 para apagar a tabela.

➔ **Linha 336:** como os dados serão preenchidos nas linhas da tabela dinamicamente é usado o método addRow() para adicionar linha no jTable passando o valor da variável “i”(linha 334) que será incrementada a cada novo registro(linha 343).

➔ **Linha 337 – 342:** é usado o método setValueAt() para inserir os dados na tabela passando o valor através do objeto “tab” da classe Cliente que chama o método get() se referindo ao valor que será inserido na linha dinamicamente e na referida coluna. Então o método setValueAt() recebe o valor, a linha e a coluna.

✓ **Exemplo**



Tela de Consultar Cliente

➢ **Criação do Método Pesquisar_Nome_Cliente()**

```
82      public List<Cliente> Perquisar_Nome_Cliente(String nome) {
83          String sql = "select idcliente, nome,  RG, CPF, Telefone, Email "
84                  + "from cliente where nome Like '" + nome + "%'";
```

Método Pesquisar_Nome_Cliente() da classe ClienteDAO.java

➔ **Linha 83, 84:** a única coisa que muda em relação ao método ListarCliente() é a String sql que agora tem uma condição de seleciona pelo nome. Por isso acredito que não seja necessário colocar todo o método, basta você como um bom programar com base no método ListarCliente() fazer o restante do método.

➢ **Criação do Método Pesquisar_Cod_Cliente()**

```
110    public List<Cliente> Perquisar_Cod_Cliente(int cod) {
111        String sql = "select idcliente, Nome, RG, CPF, Telefone, Email "
112                + "from Cliente where idcliente = '" + cod + "'";
```

Método Pesquisar_Cod_Cliente() da classe ClienteDAO.java

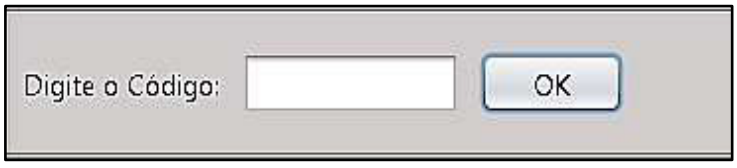
➔ **Linha 111, 112:** assim como no método Pesquisar_Nome_Cliente() a única coisa que irá mudar em relação ao método ListarCliente() é a String sql, que agora o comando faz uma condição de selecionar aonde o valor recebido por parâmetro da variável "cod" seja igual ao idcliente da tabela.

Ação de Pesquisar Cliente

Assim como o método AtualizaTable(), as ações dos botões de pesquisar irão implementar a mesma estrutura mudando apenas o chamado do método da classe ClienteDAO. Acredito que não seja necessário colocar o código fonte dos métodos, porque a única coisa que vai ser preciso mudar em relação ao método AtualizaTable() é a criação de uma variável pra receber o valor digitado no TextField, passando por parâmetro para um dos método que criamos a poucos instantes. Você como um bom programador é capaz de realizar essa tarefa.

ALTERAÇÃO DE CLIENTE

As telas de alteração seguem o mesmo padrão das telas de cadastro, mudando apenas que foi colocado mais um campo para digitar o código do cliente e um botão ok para consultar no banco de dados e listar as informações cadastradas colocando em cada campo o seu valor.



Digitar o código do cliente

- **Criação do Método Testar_Cliente()**

```
287      public boolean Testar_Cliente(int cod) {
288         boolean Resultado = false;
289         try {
290
291             String sql = "select * from cliente where idcliente = " + cod + "";
292             PreparedStatement ps = getCon().prepareStatement(sql);
293             ResultSet rs = ps.executeQuery();
294
295             if (rs != null) {
296                 while (rs.next()) {
297                     Resultado = true;
298                 }
299             }
300         } catch (SQLException ex) {
301             ex.getMessage();
302         }
303         return Resultado;
304     }
```

Método Testar_Cliente() da classe ClienteDAO.java

Esse método serve para verificar se existe algum cliente com o idcliente igual ao valor digitado pelo usuário, se o objeto “rs” for diferente de nulo(linha 295) então a variável “resultado” passará a receber true como valor(linha 297). Esse método é parecido com o método que criamos Testar_Funcionário() que verificava se existe funcionário cadastrado com aquele login e senha.

- **Criação do Método CapturarCliente()**

```
256      public List<Cliente> CapturarCliente(int cod) {
257          String sql = "select * from cliente where idcliente =" + cod + " ";
258          List<Cliente> lista = new ArrayList<>();
259          try {
260              PreparedStatement ps = getCon().prepareStatement(sql);
261              ResultSet rs = ps.executeQuery();
262              if (rs != null) {
263                  while (rs.next()) {
264                      Cliente a = new Cliente();
265                      a.setCodigo(rs.getInt(1));
266                      a.setNome(rs.getString(2));
267                      a.setNascimento(rs.getString(3));
268                      a.setRG(rs.getString(4));
269                      a.setCPF(rs.getString(5));
270                      a.setEmail(rs.getString(6));
271                      a.setTelefone(rs.getString(7));
272                      a.setBairro(rs.getString(8));
273                      a.setRua(rs.getString(9));
274                      a.setNumero(rs.getInt(10));
275                      a.setCEP(rs.getString(11));
276                      lista.add(a);
277                  }
278                  return lista;
279              } else {
280                  return null;
281              }
282          } catch (SQLException e) {
283              return null;
284          }
285      }
```

Método CapturarCliente() da classe ClienteDAO.java

Esse método é responsável por selecionar tudo da tabela cliente aonde o idcliente for igual ao código passado para a variável “cod” (linha 257). Você pode até pensar, “porque estou criando vários métodos parecidos?”, perceba que usamos a mesma lógica para todos os comandos select, e não somente para os comandos select, mais para todos os outros, insert, update e delete. Então acredito que com base nos exemplos de cliente você consegue criar as mesmas ações para todos os outros.

- **Criação do Método Alterar_Cliente()**

```
306     public String Alterar_Cliente(Cliente a) {
307         String sql = "update cliente set nome = ? ,data_nasc = ? ,rg = ? "
308                 + ",cpf = ? ,email = ? ,telefone = ? ,bairro = ?,rua = ? "
309                 + ",numero = ?,cep = ? where idcliente = ? ";
310         try {
311             PreparedStatement ps = getCon().prepareStatement(sql);
312             ps.setString(1, a.getNome());
313             ps.setString(2, a.getNascimento());
314             ps.setString(3, a.getRG());
315             ps.setString(4, a.getCPF());
316             ps.setString(5, a.getEmail());
317             ps.setString(6, a.getTelefone());
318             ps.setString(7, a.getBairro());
319             ps.setString(8, a.getRua());
320             ps.setInt(9, a.getNumero());
321             ps.setString(10, a.getCEP());
322             ps.setInt(11, a.getCodigo());
323             if (ps.executeUpdate() > 0) {
324                 return "Atualizado com sucesso.";
325             } else {
326                 return "Erro ao Atualizar";
327             }
328         } catch (SQLException e) {
329             return e.getMessage();
330         }
331     }
```

Método Alterar_Cliente() da classe ClienteDAO.java

➔ **Linha 307 – 309:** cria o comando update, para alterar os dados do cliente. O objetivo desse modulo não é explicar comandos sql, tendo em vista que todos que estão estudando conexão com banco de dados em Java, já passaram pelo modulo de banco de dados, mas se você tem duvida sobre comandos sql, não somente copie o comando mais entenda o que ele faz, esse é um principio que todo programador deve ter.

- **Criação do Método InserirDados()**

```
649     private void InserirDados(int cod) {
650
            Connection con = Conexao.AbrirConexao();
652         ClienteDAO sql = new ClienteDAO(con);
653         List<Cliente> lista = new ArrayList<>();
654         lista = sql.CapturarCliente(cod);
655
656         for (Cliente a : lista) {
657
658             jTF_Codigo.setText("" + a.getCodigo());
659             jTF_Nome.setText(a.getNome());
660             jTF_CEP.setText(a.getCEP());
661             jTF_Numero.setText("" + a.getNumero());
662             jTF_Bairro.setText(a.getBairro());
663             jTF_Email.setText(a.getEmail());
664             jTF_Telefone.setText(a.getTelefone());
665             jTF_Rua.setText(a.getRua());
666             jTF_Nascimento.setText(a.getNascimento());
667             jTF_RG.setText(a.getRG());
668             jTF_CPF.setText(a.getCPF());
669
670         }
671
672         Conexao.FecharConexao(con);
673     }
```

Método InserirDados()

Esse método recebe por parâmetro a variavel “cod” do tipo inteiro e envia para o método CapturarCliente() guardando em um ArrayList o retorno do método(linha 654). A criação desse método é feita dentro da tela de alterar cliente, ele será chamado no momento do click do botão ok.

Acredito que não seja necessário comentar linha por linha sendo que os mesmos códigos já foram usados e comentados em exemplos anteriores, se você não sabe o que está acontecendo em alguma linha de código procure em exemplos saber o que esse comando faz. Basicamente esse comando é responsável por preencher os campos com informações do cliente que pertence o código digitado.

- **Ação do Botão OK**

```
571     private void btOKActionPerformed(java.awt.event.ActionEvent evt) {
572         String codigo = jTF_cod.getText();
            Connection con = Conexao.AbrirConexao();
574         ClienteDAO sql = new ClienteDAO(con);
575         int cod = Integer.parseInt(codigo);
576         if (sql.Testar_Cliente(cod) == false) {
577             JOptionPane.showMessageDialog(null, "Codigo não Encontrado no Banco",
578                     "Video Locadora", JOptionPane.ERROR_MESSAGE);
579             Conexao.FecharConexao(con);
580         }
581         if (codigo.equals("")) {
582             JOptionPane.showMessageDialog(null, "Digite um Codigo para Atualizar",
583                     "Video Locadora", JOptionPane.WARNING_MESSAGE);
584         }
585         jTF_Codigo.setText("");
586         jTF_Nome.setText("");
587         jTF_CEP.setText("");
588         jTF_Numero.setText("");
589         jTF_Bairro.setText("");
590         jTF_Email.setText("");
591         jTF_Telefone.setText("");
592         jTF_Rua.setText("");
593         jTF_Nascimento.setText("");
594         jTF_RG.setText("");
595         jTF_CPF.setText("");
596
597         InserirDados(cod);
598         jTF_cod.setText("");
599     }
```

Ação do botão OK

➔ **Linha 576 – 578:** verifica se o retorno do método Testar_Cliente() enviando a variável “cod” é igual a false, ou seja, se o código digitado pelo usuário não estiver cadastrado no banco de dados o retorno do método é false e conseqüentemente é aberta uma caixa de mensagem dizendo “código não encontrado no banco”.

➔ **Linha 597:** nesse momento é chamado o método InserirDados() passando a variável “cod” que recebeu o valor digitado pelo usuário, depois de verificar se existe o cliente com aquele código(linha 576) e o campo para digitar não está vazio(linha 581), limpa todos os campos(linhas 585 - 595) e insere os dados pelo método InserirDados() que criamos a poucos instantes.

- **Ação de o Botão Alterar**

```
477     private void btalterarActionPerformed(java.awt.event.ActionEvent evt) {
478         String codigo = jTF_Codigo.getText();
479         String nome = jTF_Nome.getText();
480         String nascimento = jTF_Nascimento.getText();
481         String cep = jTF_CEP.getText();
482         String rua = jTF_Rua.getText();
483         String numero = jTF_Numero.getText();
484         String bairro = jTF_Bairro.getText();
485         String email = jTF_Email.getText();
486         String fone = jTF_Telefone.getText();
487         String cpf = jTF_CPF.getText();
488         String rg = jTF_RG.getText();
489         if (nome.equals("")) {
490             JOptionPane.showMessageDialog(null, "nenhum campo pode estar vazio"
491                     , "Video Locadora", JOptionPane.WARNING_MESSAGE);
492         } else {
                Connection con = Conexao.AbrirConexao();
494             ClienteDAO sql = new ClienteDAO(con);
495             int num = Integer.parseInt(numero);
496             int cod = Integer.parseInt(codigo);
497             Cliente a = new Cliente();
498
499             a.setCodigo(cod);
500             a.setNome(nome);
501             a.setNascimento(nascimento);
502             a.setRG(rg);
503             a.setCPF(cpf);
504             a.setNumero(num);
505             a.setBairro(bairro);
506             a.setCEP(cep);
507             a.setRua(rua);
508             a.setEmail(email);
509             a.setTelefone(fone);
510
511             sql.Alterar_Cliente(a);
512             Conexao.FecharConexao(con);
513
514             jTF_Nome.setText("");
515             jTF_CEP.setText("");
516             jTF_Numero.setText("");
517             jTF_Bairro.setText("");
518             jTF_Email.setText("");
519             jTF_Telefone.setText("");
520             jTF_Rua.setText("");
521             jTF_Nascimento.setText("");
522             jTF_RG.setText("");
523             jTF_CPF.setText("");
524             JOptionPane.showMessageDialog(null,"Cadastro Realizado com Sucesso",
525                     "Video Locadora", JOptionPane.INFORMATION_MESSAGE);
526             dispose();
527
528         }
```

Ação do botão alterar

➔ **Linha 478 – 788:** guarda nas variáveis do tipo String os dados dos TextFields através do método getText().

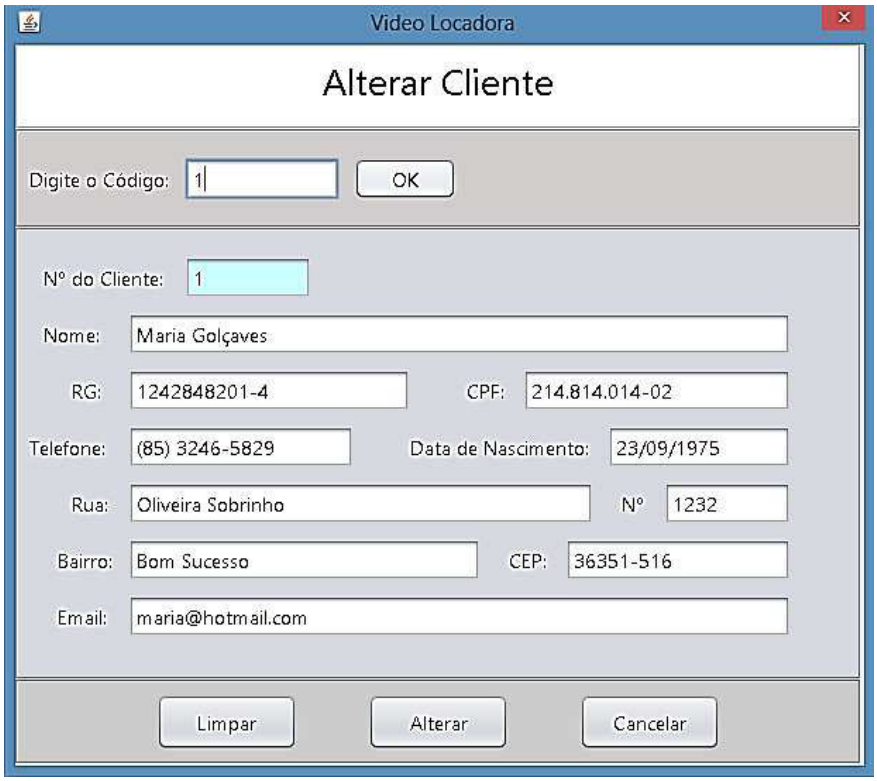
➔ **Linha 495, 496:** converte para o tipo int as variáveis que recebem o código e o numero, porque na criação da tabela os campos recebem do tipo inteiro.

➔ **Linha 499 – 509:** envia para os métodos set da classe Cliente.java através do objeto da classe(linha 497) as variáveis que receberam os dados atualizados.

➔ **Linha 511:** envia para o método Alterar_Cliente() da classe ClienteDAO.java o objeto "a" da classe Cliente.java. A ação de alterar é parecida com a ação de cadastrar mudando algumas linhas de código.

✓ **Exemplo**

Vamos supor que a cliente "Maria Gonçalves" mudou de endereço e precisa alterar as informações do seu cadastro. Então basta digitar o código do seu cadastro na tela de alteração e modificar o que for necessário, depois basta clicar no botão Alterar e as suas informações serão atualizadas, porém se não lembrar o código basta fazer uma consulta rápida pelo nome para saber o código.



Exemplo de Alterar Cliente

EXCLUSÃO DE CLIENTE

A tela de exclusão é bastante simples, ela segue um modelo para todas as outras, sabendo fazer a ação de uma, as outras ficam bastante simples. Algo novo que ainda não vimos em nenhum exemplo até agora, é o uso do componente combobox. Preste atenção com colocar dados do banco dentro de uma combobox porque isso será muito usado.

- **Criação do Método ListarComboCliente()**

Esse método é responsável por pegar os nomes dos clientes para listar dentro da combobox. Perceba que usamos a clausula “order by nome” (linha 164) para listar os nomes dos clientes em ordem alfabética.

```
162     public List<Cliente> ListarComboCliente() {
163
164         String sql = "select nome from cliente order by nome ";
165         List<Cliente> lista = new ArrayList<>();
166         try {
167             PreparedStatement ps = getCon().prepareStatement(sql);
168             ResultSet rs = ps.executeQuery();
169
170             if (rs != null) {
171                 while (rs.next()) {
172
173                     Cliente a = new Cliente();
174                     a.setNome(rs.getString(1));
175                     lista.add(a);
176                 }
177                 return lista;
178             } else {
179                 return null;
180             }
181
182         } catch (Exception e) {
183             return null;
184         }
185     }
186
```

Método ListarComboCliente() da classe ClienteDAO.java

- **Criação do Método AtualizaCombo()**

Esse método é o primeiro a ser chamado dentro do método construtor da tela de excluir cliente, ele é o responsável por adicionar ao combobox que foi nomeado como jCB_Nome a lista contendo os clientes cadastrados.

```
263     private void AtualizaCombo() {
264         Connection con = Conexao.AbrirConexao();
265         ClienteDAO sql = new ClienteDAO(con);
266         List<Cliente> lista = new ArrayList<>();
267         lista = sql.ListarComboCliente();
268         jCB_Nome.addItem("");
269
270         for (Cliente b : lista) {
271
272             jCB_Nome.addItem(b.getNome());
273         }
274         Conexao.FecharConexao(con);
275     }
```

Método AtualizaCombo()

➔ **Linha 268:** para adicionar itens na combobox usamos o método addItem(), para que o primeiro cliente da lista não apareça no momento de carregar a tela, adicionamos uma String vazia antes de adicionar os nomes.

➔ **Linha 272:** nesse momento são listados dentro da combobox os nomes dos clientes em ordem alfabética. É preciso ter cuidado para não se esquecer de chamar o método AtualizaCombo() dentro do método initComponents() para que assim que abrir o Jframe os nomes seja adicionados na combobox.

- ✓ **Exemplo**

- **Criação do Método ConsultaCodigoCliente()**

```
187     public List<Cliente> ConsultaCodigoCliente(String nome) {
188
189         String sql = "select idcliente from cliente where nome  = '" + nome + "'";
190         List<Cliente> lista = new ArrayList<>();
191         try {
192             PreparedStatement ps = getCon().prepareStatement(sql);
193             ResultSet rs = ps.executeQuery();
194
195             if (rs != null) {
196                 while (rs.next()) {
197
198                     Cliente a = new Cliente();
199                     a.setCodigo(rs.getInt(1));
200                     lista.add(a);
201                 }
202                 return lista;
203             } else {
204                 return null;
205             }
206
207         } catch (Exception e) {
208             return null;
209         }
210
211     }
```

Método ConsultaCodigoCliente() da classe ClienteDAO.java

- **Ação de Selecionar na combobox**

```
    private void jCB_NomeActionPerformed(java.awt.event.ActionEvent evt) {
279         Connection con = Conexao.AbrirConexao();
280         ClienteDAO sql = new ClienteDAO(con);
281         List<Cliente> lista = new ArrayList<>();
282         String nome = jCB_Nome.getSelectedItem().toString();
283
284         lista = sql.ConsultaCodigoCliente(nome);
285
286         for (Cliente b : lista) {
287             int a = b.getCodigo();
288             jTF_codigo.setText("" + a);
289         }
290         Conexao.FecharConexao(con);
291     }
```

Ação de Selecionar na combobox

➔ **Linha 282:** para pegar o valor digitado em uma TextField usamos o método getText(), mas para pegar o valor selecionado na combobox é preciso usar o método getSelectedItem().toString().

➔ **Linha 284:** o ArrayList de nome lista recebe o retorno do método ConsultaCodigoCliente(), perceba que passamos como parâmetro para o método a variável "nome" que guardou o cliente selecionado na combobox.

- **Criação do Método Excluir_Cliente()**

```
237     public String Excluir_Cliente(Cliente a) {
238         String sql = "delete from cliente where idcliente = ? and nome = ? ";
239
240         try {
241             PreparedStatement ps = getCon().prepareStatement(sql);
242             ps.setInt(1, a.getCodigo());
243             ps.setString(2, a.getNome());
244             if (ps.executeUpdate() > 0) {
245                 return "Excluído com sucesso.";
246             } else {
247                 return "Erro ao excluir";
248             }
249         } catch (SQLException e) {
250             return e.getMessage();
251         }
252     }
253 }
```

Método Excluir_Cliente() da classe ClienteDAO.

➔ **Linha 238:** cria o comando sql para deletar o cliente que possui o idcliente e o nome de acordo com os valores passados(linhas 242 e 243).

➔ **Linha 244:** verifica se o método executeUpdate() é maio que zero, isso para saber se alguma linhas da tabela foi afetadas, se for maior que zero retorna uma String informando excluído com sucesso.

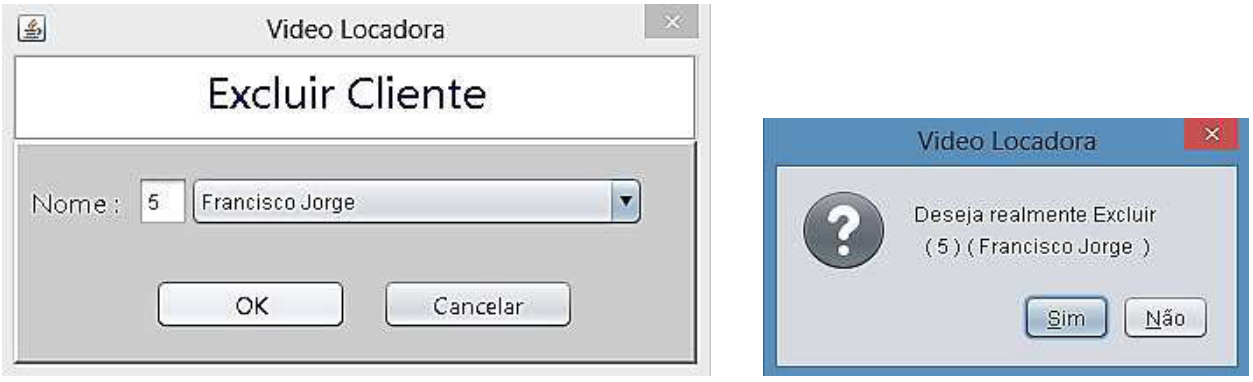
- **Ação de o Botão Excluir**

```
174     private void btExcluirActionPerformed(java.awt.event.ActionEvent evt) {
175         String codigo = jTF_codigo.getText();
176         String nome = jCB_Nome.getSelectedItem().toString();
177 
178         Connection con = Conexao.AbrirConexao();
179         ClienteDAO sql = new ClienteDAO(con);
180         Cliente a = new Cliente();
181         if (nome.equals("")) {
182             JOptionPane.showMessageDialog(null, "Nenhum Nome Selecionado",
183                     "Video Locadora", JOptionPane.WARNING_MESSAGE);
184         } else {
185         int b = JOptionPane.showConfirmDialog(null, "Deseja realmente Excluir"
186                + " \n  ( " + codigo + " ) ( " + nome + " ) ", "Video Locadora",
187                 JOptionPane.YES_NO_OPTION, JOptionPane.QUESTION_MESSAGE);
188             if (b == 0) {
189                 int cod = Integer.parseInt(codigo);
190                 a.setNome(nome);
191                 a.setCodigo(cod);
192                 sql.Excluir_Cliente(a);
193                 Conexao.FecharConexao(con);
194                 dispose();
195 
196             }
197         }
198     }
```

Ação do botão excluir

- **Exemplo**

Depois de selecionado o nome do cliente e apertado no botão OK aparece uma mensagem perguntando se realmente deseja excluir o cliente.



Exemplo de Exclusão de Cliente

CADASTRO DE FILME

Com base no que já foi ensinado usando o exemplo de clientes, acredito que não será preciso explicar novamente o mesmo processo para todos os cadastros, então vamos nos prender naquilo que ainda não ensinamos, como por exemplo, carregar a capa do filme. E você como um bom programador irá fazer o restante dos cadastros, consultas, alterações e exclusões.

```
476     private void btCapaActionPerformed(java.awt.event.ActionEvent evt) {
477         try {
478             JFileChooser foto = new JFileChooser();
479             foto.setCurrentDirectory(new File("/C:/Video Locadora/Pictures/"));
480             foto.setDialogTitle("Carregar Capa");
481             foto.showOpenDialog(this);
482             String a = "" + foto.getSelectedFile().getName();
483             tfCapa.setText(a);
484             lbCapa.setIcon(new ImageIcon
485                     ("/C:/Video Locadora/Pictures/"+ tfCapa.getText() + "/"));
486         } catch (Exception e) {
487            JOptionPane.showMessageDialog(null,"Não foi possivel carregar capa");
488         }
489     }
```

Ação do botão de carregar capa

- **Linha 478:** cria um objeto da classe JFileChooser chamado “foto”, O FileChooser, permite que você escolha um arquivo. Ele abre uma caixinha com cara de salvar, mas sua tarefa se resume a apenas escolher um arquivo de acordo com o caminho especificado.
- **Linha 479:** basicamente define o diretório inicial para um JFileChooser através no método setCurrentDirectory(), ou seja, especifica o local que será aberto para escolher as capas de filmes.
- **Linha 480:** define através do método setDialogTitle() qual será o titulo que aparecerá na caixa de seleção.

➔ **Linha 481:** nesse momento abre a caixa de dialogo através do método showOpenDialog(this).

➔ **Linha 482, 483:** guarda na variável "a" o nome do arquivo selecionado através do método getSelectedFile.getName() e mostra no tfCapa (linha 483)

➔ **Linha 484, 485:** usa o método setIcon do lbcapa para mostrar a imagem, de forma que especifica o caminho do arquivo. Mais além de especificar o caminho do arquivo é preciso dizer o nome dele, podemos saber o nome do arquivo através do tfcapa (linha 483), então basta usar o método getText().

✓ **Exemplo**

Exemplo de Cadastro de Filme

CADASTRO DE LOCAÇÃO

Espero que você possa estar entendo os códigos e conseguido realizar as tarefas de cadastrar, consultar, alterar e excluir (Cliente, Filme, DVD, Categoria, Classificação, Funcionário). Se você ainda não terminou essa parte sugiro que não prossiga e se concentre nessas principais funcionalidades.

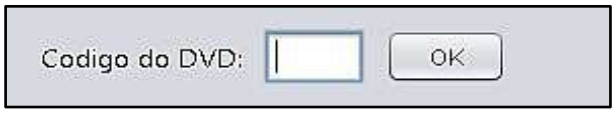
- **Criação do Método Testar_DVD()**

```
192     public boolean Testar_DVD(int cod) {
193         boolean teste = false;
194         try {
195
196             String sql = "select iddvd from dvd where iddvd =" + cod + "";
197            PreparedStatement ps = getCon().prepareStatement(sql);
198             ResultSet rs = ps.executeQuery();
199
200             if (rs != null) {
201                 while (rs.next()) {
202                     teste = true;
203                 }
204             }
205
206         } catch (SQLException ex) {
207         }
208         return teste;
209     }
```

- **Criação do Método Testar_Situacao()**

```
119 public boolean Testar_Situacao(int cod) {
120         boolean teste = false;
121         try {
122
123             String sql = "select iddvd from dvd where iddvd =" + cod + ""
124                     + " and situacao = 'Disponivel'";
125            PreparedStatement ps = getCon().prepareStatement(sql);
126            ResultSet rs = ps.executeQuery();
127
128             if (rs != null) {
129                 while (rs.next()) {
130                     teste = true;
131                 }
132             }
133
134         } catch (SQLException ex) {
135         }
136         return teste;
137     }
```

Os métodos criados serão usados no momento de digitar o código do DVD para alocação, porque é preciso verificar se existe o código cadastrado através do método Testar_DVD() e se o DVD esta emprestado através do método Testar_Situacao().




```
603     private void btOKActionPerformed(java.awt.event.ActionEvent evt) {
604         String pesquisa = jTF_CodDVD.getText();
605         Connection con = Conexao.AbrirConexao();
606         if (pesquisa.equals("")) {
607             JOptionPane.showMessageDialog(null, "Digite o codigo do DVD",
608                     "Video Locadora", JOptionPane.WARNING_MESSAGE);
609         } else {
610 
611             DVDDAO sql = new DVDDAO(con);
612             int cod = Integer.parseInt(pesquisa);
613             if (sql.Testar_DVD(cod) == false) {
614             JOptionPane.showMessageDialog(null, "Codigo do DVD não Encontrado",
615                     "Video Locadora", JOptionPane.ERROR_MESSAGE);
616                 jTF_CodDVD.setText("");
617                 jTF_Titulo.setText("");
618                 jTF_Valor.setText("");
619                 jTF_Categoria.setText("");
620                 jTF_Classificacao.setText("");
621                 jLbFoto.setIcon(new ImageIcon(""));
622                 jTF_Codigo.setText("");
623 
624             }else
625             if (sql.Testar_Situacao(cod) == false) {
626                 JOptionPane.showMessageDialog(null, "o DVD de codigo ( "+cod+" )"
627                 + " está Emprestado", "Video Locadora", JOptionPane.INFORMATION_MESSAGE);
628                 jTF_CodDVD.setText("");
629                  jTF_Titulo.setText("");
630                 jTF_Valor.setText("");
631                 jTF_Categoria.setText("");
632                 jTF_Classificacao.setText("");
633                 jLbFoto.setIcon(new ImageIcon(""));
634                 jTF_Codigo.setText("");
635             } else {
636 
637                 InserirDados(cod);
638                 jTF_Codigo.setText(pesquisa);
639                 jTF_CodDVD.setText("");
640             }
641         }
642         Conexao.FecharConexao(con);
643     }
```

Ação do botão OK

- **Criação do Método AtualizaDate()**

Esse método é responsável por preencher os campos de data da locação e o horário atual, de acordo com a data e a hora do relógio do computador. Ele deve ser o primeiro método chamado ao ser aberto a tela.

```
922     public void AtualizaDate() {
923
924         Date date = new Date();
925         SimpleDateFormat data = new SimpleDateFormat("dd/MM/yyyy");
926         SimpleDateFormat hora = new SimpleDateFormat("hh:mm");
927         jTF_DataLocacao.setText(data.format(date));
928         jTF_Horas.setText(hora.format(date));
929
930     }
```

Método AtualizaDate()

- **Linha 924:** instância um objeto da classe Date(). A classe Date armazena uma data e hora, que internamente é armazenada com um inteiro long que é o numero de milissegundos.
- **Linha 925, 926:** instância os objetos da classe SimpleDateFormat passando os formatos de queremos para data e hora. A classe SimpleDateFormat pode facilmente converter um objeto da classe Date para uma String no formato que a gente quiser.
- **Linha 927, 928:** através dos objetos da classe SimpleDateFormat chamamos o método format() passando o objeto date(linha 924), ou seja, nesse momento irá aparecer nos campos a hora e a data, obedecendo o formato que foi especificado para cada objeto.

- ✓ **Exemplo**

String	Data formatada	Comentário
dd/MM/yyyy	25/12/2010	Padrão brasileiro
MM/dd/yyyy	12/25/2010	Padrão americano
yyyy-MM-dd	2010-12-25	Padrão de alguns bancos de dados
dd MMMM yyyy	25 Dezembro 2010	Quando tem mais de 2 caracteres 'M', o resultado é o nome do mês
hh:mm:ss	16:23:34	Hora…

- **Criação do Método ListarCodFilme()**

```
73      public List<DVD> ListarCodFilme(int cod) {
74          String sql = "select idfilme from dvd where iddvd = " + cod + " ";
75          List<DVD> lista = new ArrayList<>();
76          try {
77              PreparedStatement ps = getCon().prepareStatement(sql);
78              ResultSet rs = ps.executeQuery();
79
80              if (rs != null) {
81                  while (rs.next()) {
82                      DVD a = new DVD();
83                      a.setCod_filme(rs.getInt(1));
84
85                      lista.add(a);
86                  }
87                  return lista;
88              } else {
89                  return null;
90              }
91          } catch (SQLException e) {
92              return null;
93          }
94      }
```

- **Criação do Método Pesquisar_Cod_Filme()**

```
199     public List<Filme> Perquisar_Cod_Filme(int cod) {
200     String sql = "select idfilme,titulo,ano,duracao,idcategoria,idclassificacao,"
201                 + "capa from filme where idfilme = '" + cod + "'";
202         List<Filme> lista = new ArrayList<>();
203         try {
204             PreparedStatement ps = getCon().prepareStatement(sql);
205             ResultSet rs = ps.executeQuery();
206
207             if (rs != null) {
208                 while (rs.next()) {
209                     Filme a = new Filme();
210                     a.setCodigo(rs.getInt(1));
211                     a.setTitulo(rs.getString(2));
212                     a.setAno(rs.getInt(3));
213                     a.setDuracao(rs.getString(4));
214                     a.setCod_categoria(rs.getInt(5));
215                     a.setCod_classificao(rs.getInt(6));
216                     a.setCapa(rs.getString(7));
217                     lista.add(a);
218                 }
219                 return lista;
220             } else {
221                 return null;
222             }
223         } catch (SQLException e) {
224             return null;
225         }
226     }
```

Os métodos criados serão usados pelo método InserirDados(), porque é preciso saber o idfilme através do método ListarCodFilme() passando o código do DVD digitado, para que então possamos obter os dados da tabela filme através do método Pesquisar_Cod_Filme() passando o código obtido na consulta anterior.

Espero que você esteja entendendo a lógica usada, porque em uma locadora podemos ter vários DVDs de um mesmo filme, então na verdade é alugado o DVD e não o filme, por isso foi dividido o cadastro de filme e o cadastro de DVD, e consequentemente tivemos que criar esses métodos.

- **Criação do Método InserirDados()**

```
891     private void InserirDados(int cod) {
892
893         Connection con = Conexao.AbrirConexao();
894         DVDDAO dvd = new DVDDAO(con);
895         FilmeDAO filme = new FilmeDAO(con);
896         List<DVD> listaDVD = new ArrayList<>();
897         List<Filme> listaFIL = new ArrayList<>();
898         listaDVD = dvd.ListarCodFilme(cod);
899         for (DVD a : listaDVD) {
900             int codigo = a.getCod_filme();
901             listaFIL = filme.Perquisar_Cod_Filme(codigo);
902         }
903         for (Filme a : listaFIL) {
904             jTF_Titulo.setText(a.getTitulo());
905             jTF_Categoria.setText("" + a.getCod_categoria());
906             jTF_Classificacao.setText("" + a.getCod_classificao());
907             jLbFoto.setIcon(new ImageIcon("/C:/Video Locadora/Pictures/"
908                     + a.getCapa() + "/"));
909         }
910         ClassificacaoDAO cla = new ClassificacaoDAO(con);
911         List<Classificacao> listaCLA = new ArrayList<>();
912         String b = jTF_Classificacao.getText();
913         int codigo = Integer.parseInt(b);
914         listaCLA = cla.ListarPrecoClassificacao(codigo);
915         for (Classificacao a : listaCLA) {
916             double preco = a.getPreco();
917             jTF_Valor.setText("" + preco + "0");
918         }
919         Conexao.FecharConexao(con);
920     }
```

Método InserirDados()

- **Ação de o Botão Cadastrar**

```
683    private void btCadastrarActionPerformed(java.awt.event.ActionEvent evt) {
684        String dvd = jTF_Codigo.getText();
685        String cliente = jTF_CodCliente.getText();
686        String horario = jTF_Horas.getText();
687        String aluguel = jTF_DataLocacao.getText();
688        if (dvd.equals("") || cliente.equals("") || jDateDevolucao.getDate() == null) {
689            JOptionPane.showMessageDialog(null, "nenhum campo pode estar vazio",
690                    "Video Locadora", JOptionPane.WARNING_MESSAGE);
691        } else {
692        String devolucao = new SimpleDateFormat("dd/MM/yyyy").format(jDateDevolucao.getDate());
693            Connection con = Conexao.AbrirConexao();
694            AluguelDAO sql = new AluguelDAO(con);
695            int coddvd = Integer.parseInt(dvd);
696            int codcli = Integer.parseInt(cliente);
697            Aluguel a = new Aluguel();
698            a.setCoddvd(coddvd);
699            a.setCodcliente(codcli);
700            a.setHorario(horario);
701            a.setData_aluguel(aluguel);
702            a.setData_devolucao(devolucao);
703            sql.Inserir_Aluguel(a);
704            String situacao = "Emprestado";
705            sql.Atualizar_Situacao(situacao, coddvd);
706            Conexao.FecharConexao(con);
707
708            JOptionPane.showMessageDialog(null,"Cadastro Realizado com Sucesso",
709                    "Video Locadora", JOptionPane.INFORMATION_MESSAGE);
710            dispose();
711        }
712    }
```

- **Exemplo**

REALIZAR DEVOLUÇÃO

Para realizar devolução o usuário escolhe o nome do cliente na combobox e a tabela lista todas as locações que pertencem a esse cliente e clicando na linha da tabela abre uma nova tela para devolver o DVD. Com base nos exemplos criados vocês sabem listar os nomes dos clientes dentro da combobox e também sabem consultar e listar na tabela através do código e do nome do cliente.

Acredito que não seja necessário mostrar novamente os códigos, volte na parte que excluímos os clientes e também na de consultar e use a mesma lógica. Vamos mostrar agora o evento do click na tabela. Siga o caminho de clicar com o botão direito na tabela -> Eventos -> Mouse -> MouseClicked.

- **Criação da classe Listar.java**

```
6   package Modelo;
7
8   public class Listar {
9
10      static private int coddvd;
11      static private int codaluguel;
12      static private int codcliente;
13
14      //metodos getters e setters
```

- **Criação do evento de clicar na tabela**

```
168     private void jTableMouseClicked(java.awt.event.MouseEvent evt) {
169         Integer linha = jTable.getSelectedRow();
170         Integer idaluguel = (Integer) jTable.getValueAt(linha, 0);
171         Integer idcliente = (Integer) jTable.getValueAt(linha, 1);
172         Integer iddvd = (Integer) jTable.getValueAt(linha, 2);
173         Listar a = new Listar();
174         a.setCoddvd(iddvd);
175         a.setCodaluguel(idaluguel);
176         a.setCodcliente(idcliente);
177
178         new EfetuarDevolucao().setVisible(true);
179
180     }
```

- ✓ **Exemplo**

Exemplo de consultar locações

Exemplo de devolução

CONCLUSÃO

Espero que você tenha conseguido realizar todas as operações mostradas nesse modulo, o principal objetivo era que você conseguisse realizar as principais operações no banco de dados (Cadastrar, Consultar, Alterar, Excluir), se você conseguiu parabéns.

Agora nada impede com que você evolua o sistema realizando mais funcionalidades como, por exemplo, fazer reservas de DVD, calcular multas por atraso, melhorar a forma de devolução, mostrar o valor que o cliente está devendo não deixando alocar novos DVD antes de pagar a divida, etc.

Cabe a cada um de vocês terem a criatividade e com base nos exemplos mostrados e buscando em outras fontes de estudo mais operações que podem ser realizadas em uma locadora, ou até mesmo criarem seus próprios sistemas, porque a cada sistema ou funcionalidade criado, você aprende algo novo no mundo da programação em Java.

Hino Nacional

Ouviram do Ipiranga as margens plácidas
De um povo heróico o brado retumbante,
E o sol da liberdade, em raios fúlgidos,
Brilhou no céu da pátria nesse instante.

Se o penhor dessa igualdade
Conseguimos conquistar com braço forte,
Em teu seio, ó liberdade,
Desafia o nosso peito a própria morte!

Ó Pátria amada,
Idolatrada,
Salve! Salve!

Brasil, um sonho intenso, um raio vívido
De amor e de esperança à terra desce,
Se em teu formoso céu, risonho e límpido,
A imagem do Cruzeiro resplandece.

Gigante pela própria natureza,
És belo, és forte, impávido colosso,
E o teu futuro espelha essa grandeza.

Terra adorada,
Entre outras mil,
És tu, Brasil,
Ó Pátria amada!
Dos filhos deste solo és mãe gentil,
Pátria amada,Brasil!

Deitado eternamente em berço esplêndido,
Ao som do mar e à luz do céu profundo,
Fulguras, ó Brasil, florão da América,
Iluminado ao sol do Novo Mundo!

Do que a terra, mais garrida,
Teus risonhos, lindos campos têm mais flores;
"Nossos bosques têm mais vida",
"Nossa vida" no teu seio "mais amores."

Ó Pátria amada,
Idolatrada,
Salve! Salve!

Brasil, de amor eterno seja símbolo
O lábaro que ostentas estrelado,
E diga o verde-louro dessa flâmula
- "Paz no futuro e glória no passado."

Mas, se ergues da justiça a clava forte,
Verás que um filho teu não foge à luta,
Nem teme, quem te adora, a própria morte.

Terra adorada,
Entre outras mil,
És tu, Brasil,
Ó Pátria amada!
Dos filhos deste solo és mãe gentil,
Pátria amada, Brasil!

Hino do Estado do Ceará

Poesia de Thomaz Lopes
Música de Alberto Nepomuceno
Terra do sol, do amor, terra da luz!
Soa o clarim que tua glória conta!
Terra, o teu nome a fama aos céus remonta
Em clarão que seduz!
Nome que brilha esplêndido luzeiro
Nos fulvos braços de ouro do cruzeiro!

Mudem-se em flor as pedras dos caminhos!
Chuvas de prata rolem das estrelas...
E despertando, deslumbrada, ao vê-las
Ressoa a voz dos ninhos...
Há de florar nas rosas e nos cravos
Rubros o sangue ardente dos escravos.
Seja teu verbo a voz do coração,
Verbo de paz e amor do Sul ao Norte!
Ruja teu peito em luta contra a morte,
Acordando a amplidão.
Peito que deu alívio a quem sofria
E foi o sol iluminando o dia!

Tua jangada afoita enfune o pano!
Vento feliz conduza a vela ousada!
Que importa que no seu barco seja um nada
Na vastidão do oceano,
Se à proa vão heróis e marinheiros
E vão no peito corações guerreiros?

Se, nós te amamos, em aventuras e mágoas!
Porque esse chão que embebe a água dos rios
Há de florar em meses, nos estios
E bosques, pelas águas!
Selvas e rios, serras e florestas
Brotem no solo em rumorosas festas!
Abra-se ao vento o teu pendão natal
Sobre as revoltas águas dos teus mares!
E desfraldado diga aos céus e aos mares
A vitória imortal!
Que foi de sangue, em guerras leais e francas,
E foi na paz da cor das hóstias brancas!

GOVERNO DO
ESTADO DO CEARÁ

Secretaria da Educação